2014

Moderní učebnice informačních a komunikačních technologií

Jan Petrásek

Úvodem

Počítače jsou nedílnou součástí našich životů již leta a jejich význam v životě každého z nás velmi rychle roste. Počítače potkáme všude, doma, v práci, ve škole, v obchodě, jsou prostě všude. Nedílnou součástí jejich světa jsou i mobilní telefony, ať chytré, nebo klasické. Naše doba je přetechnizovaná a nutí člověka, aby uměl alespoň základní operace s PC bez ohledu na pohlaví, vzdělání, postavení i cíle člověka jako osobnosti. Bohužel se blíží doba, kdy se bez počítače neobejdeme ani v důchodu.

Sám jsem počítačový samouk, v rodině jsem neměl nikoho, kdy by mi mohl podat pomocnou ruku, a dotáhl jsem to po několika letech pilné práce (nikoli hraní her a chatování) až k získání prestižních certifikátů (řízení jakosti SW, Systémové a aplikační integrace, platforma .NET a další). I přes tyto pro většinu lidí až záhadně vysoké úspěchy se pořád považuji za IT začátečníka. Bohužel spousta lidí neumí základní věci, kolikrát je o ně učitelé informatiky okradli, a dávají různým servisům nemalé peníze za to, co si mohou udělat sami. Neříkám, že se má člověk hrabat v počítači, který si koupil a má ještě záruku, problémy, na něž se vztahuje záruka, tak samozřejmě přenechme výrobcům, za co bychom je jinak platili, ale pak jsou věci, které si může každý během chvilky udělat sám. Můžu říct, že pokud si počítač člověk postaví z komponentů a osadí celý sám, tak je to stroj, který dokonale vyhovuje jeho potřebám a přáním, ale hlavně člověka hřeje dobrý pocit z toho, že to dokázal.

Když jsem přemýšlel, jak bych mohl pomoci více lidem dříve, nežli dokončím učitelství na přírodovědecké fakultě, tak mi řekla jedna kamarádka, kterou jsem učil s počítačem, sepiš ty svoje prezentace a udělej z toho volně šiřitelnou knihu v elektronické podobě. Touto publikací bych chtěl lidem podat onu pomocnou ruku, která mně chyběla, když jsem před lety sám začínal. Její nápad se mi velice zalíbil, a tak vznikla tato knížka, i když se při jejím psaní sám dále vzdělávám a objevuji spoustu nových věcí, tak, vám čtenářům, chci ukázat, jak vypadá svět počítačů a ušetřit vám čas s „objevováním Ameriky“. Také bych chtěl trošku nastínit, jak fungují mobilní zařízení, ať chytré, nebo klasické mobilní telefony, jak funguje taková telefonní síť a jak je možné například hledat ztracené telefony a nejen to. Tak už dost povídání a pojďme na to…

Poděkování

Poděkovat bych chtěl v první řadě paní RNDr. Petruši Danielové, která mě jako můj metodik ICT naučila, jak připravovat informace pro druhé a také mi rozšířila do té doby moje velmi úzce zaměřené znalosti z oblasti ICT.

Další velký dík patří panu Mgr. Davidu Mikolášovi, který mě seznámil s didaktikou a s přípravou materiálů a jejich následném zpracování, nejen v oblasti IT.

Poděkování patří také slečně Heidi Wiesnerové, která se jako počítačový laik, ale velmi zdatná metodička anglického jazyka podílela přímo na vzniku publikace. Kromě korekcí anglických výrazů také připomínkovala srozumitelnost knihy pro lajky.

Velké díky za korekce patří také paní Bc. Lucii Obstové, která se podílela především na kontrole jazykové stánky a připomínkovala srozumitelnost textu běžnému uživateli.

Také nesmím zapomenout na svého skoro bratra Jana Plachého, který se jako odborník na SW, především Windows 7 a XP, podílel radou na vzniku některých pasáží.

Za spolupráci ohledně Linuxu děkuji Štěpánu Kolářovi v licenci CC-BY-SA 4.0.

V neposlední řadě nesmím zapomenout na kamarádky Niky a Marianu, které mi v době psaní byly velkou psychickou oporou. Při aktualizaci knihy i její online verze v roce 2017 mi byla velkou oporou moje přítelkyně Veronika, které tímto patří obrovský dík, její zásluhou se kniha dočkala plánovaných rozšíření a aktualizací.

Obsah

ÚVODEM 1

PODĚKOVÁNÍ 1

OBSAH 2

1. POČÍTAČ 25

1.1. Co je počítač? 25

1.2. Jak počítač funguje? 25

1.3. Z čeho se počítač skládá? 26

1.4. Hygiena 26

1.5. Abychom si rozuměli… 28
1.5.1. Software 28
1.5.2 Hardware 28
1.5.2.1. Jednotky užívané v IT 29
1.5.3. Číselné soustavy 29
1.5.3.1. Binární soustava 29
Příklad převodu pomocí dělení dvěma 30
Příklad převodu pomocí vah 30
Příklad váhového převodu 2 31
Příklad váhového převodu 31
1.5.3.2. Hexadecimální soustava 31
Příklad převodu 32
1.5.4. Shrnutí kapitoly 32

1.6 Historie 32
1.6.1. Předchůdci 33
1.6.1.1. Abakus 33
1.6.1.2. Mechanické kalkulátory 33
1.6.1.3. Logaritmické pravítko 34
1.6.2. Nultá generace 34
1.6.2.1. Z1 35
1.6.2.2. Z2 a Z3 35
1.6.2.3. Mark I 35
1.6.2.4. Mark II 36
1.6.2.5. SAPO 36
1.6.3. První generace 36
1.6.3.1. ENIAC a MANIAC 37
1.6.4. Druhá generace 37
1.6.5. Třetí generace 37
1.6.6. Třiapůltá generace 38
1.6.7. Čtvrtá generace 38

1.7. Ženy v Historii IT 38

1.8. Souhrnem… 41

2. HARDWARE 42

2.1. Skříň (Case) 42

2.2. Periferie 43
2.2.1. Vstupní zařízení 43
2.2.1.1. klávesnice 43
2.2.1.2. Myš 45
2.2.1.2.1. Snímače CCD 46
2.2.1.3. Scanner 47
2.2.1.3.1. Základní parametry 47
2.2.1.3.2. Princip 47
CCD snímání 47
CIS snímání (Contact Image Senzor) 48
2.2.1.3.3. Filmové scannery 48
2.2.1.3.4. Bubnové 49
2.2.1.3.5. Stolní 50
2.2.1.3.6. Ruční 50
2.2.1.3.7. 3D skenování 51
2.2.1.3.8. OCR 52
2.2.1.4. Mikrofon 53
2.2.1.4.1. Kondenzátorový mikrofon 53
2.2.2. Výstupní zařízení 54
2.2.2.1. Monitor 54
2.2.2.1.1. CRT 54
2.2.2.1.2. LCD – Liquid Crystal Display, displej s tekutými krystaly 55
2.2.2.1.3. PDP – plazmové monitory 58
2.2.2.1.4. OLED display 58
2.2.2.1.5. Jak vybrat monitor 59
2.2.2.2. Tiskárny a plotry 59
2.2.2.2.1. inkoustové 60
2.2.2.2.2. Laserové 60
2.2.2.2.3. Jehličkové 61
2.2.2.2.4. Plotry 61
2.2.2.3. Reproduktory 63
2.2.2.3.1. Elektrodynamický reproduktor přímo vyzařující 63
2.2.2.3.2. Elektrodynamický reproduktor nepřímo vyzařující 64
2.2.2.3.3. Vícepásmové reproduktorové soustavy 64

2.3. Obsah skříně 64
2.3.1. Základová deska 64
2.3.1.1. Sběrnice 66
2.3.1.1.1. Základní parametry 66
2.3.1.1.2. Režimy práce 66
2.3.1.1.3. Části 67
2.3.1.1.4. Sloty a porty 67
2.3.1.1.4.1. Sériové rozhraní 67
2.3.1.1.5. ISA 68
2.3.1.1.6. MCA 68
2.3.1.1.7. EISA 68
2.3.1.1.8. VESA VL-BUS 68
2.3.1.1.9. PCI 69
2.3.1.1.10. AGP (accelerated graphics port) 69
2.3.1.1.11. CNR (comunication network riser) 69
2.3.1.1.12. PCI-E 69
2.3.1.2. Další konektory 70

2.3.2 CPU (Procesor) 70
2.3.2.1. Strojový kód 70
2.3.2.1.1. Strojová instrukce 71
2.3.2.2. Dělení procesorů 71
2.3.2.3. Součásti 71
2.3.2.4. Mechanismus stránkování 71
2.3.2.5. Tabulka základních parametrů 72
2.3.2.6. Rychlost procesoru 72
2.3.2.7. Patice 73
2.3.3. Zdroj 73
2.3.4. Grafické karty 75
2.3.4.1. Základní parametry 75
2.3.4.2. Základní součásti 75
2.3.4.3. Užívané sběrnice 77
2.3.4.4. Rozlišení pro 16:9 78
2.3.4.5. Chlazení 78
2.3.4.6. Omezení grafických karet 78
2.3.4.7. Výrobci 78
2.3.5. Zvuková karta 79
2.3.6. Optické mechaniky 80
2.3.7. Síťová karta 80
2.3.8. Kabely 81
2.3.9. Chlazení 81
2.3.10. Paměti 82
2.3.10.1. Kapacita 83
2.3.10.2. Dynamičnost 83
2.3.10.3. Energetická závislost 83
2.3.10.4. Přístup k paměti 83
2.3.10.5. Vnitřní paměti energeticky nezávislé 83
2.3.10.6. Vnitřní paměti energeticky závislé 85
2.3.10.6.1. SIMM 85
2.3.10.6.2. DIMM 85
2.3.10.6.3. Cache 85
2.3.10.6.4. CMOS 86
2.3.10.6.5. BIOS 86
2.3.10.7. Magnetický záznam dat 87
2.3.10.8. Vnější paměti – Disketa 87
2.3.10.9. Vnější paměti - ZIP disk 88
2.3.10.9. Vnější paměti - Bernoulliho disk 88
2.3.10.10. Vnější paměti - Magneto-optický disk 88
2.3.10.11. Vnější paměti - Páskové paměti (streamer) 89
2.3.10.12. HDD 90
2.3.10.12.1. Technologie Raid 91
2.3.10.12.2. Řadič IDE 91
2.3.10.12.3. Řadič SATA 91
2.3.10.12.3. Řadič SCI 91
2.3.10.12.4. Formátování 92
2.3.10.12.5. Alokační tabulka 92
2.3.10.13. Vnější paměti - Paměťové karty 92
2.3.10.13.1. Programování paměti 92
2.3.10.13.2. Mazání paměti 93
2.3.10.13.3. Zapojení pamětí 93
2.3.10.14. Optické disky 93
2.3.10.14.1. CD (Compact Disc) 93
2.3.10.14.2. DVD (Digital Video Disc) 94
2.3.310.14.3. BD (Blu-ray Disc) 95

2.3.10.14.4. Vrstvy média 95

2.4. Jak postavit počítač? 95
2.4.1. Sestavení 96

2.5. Virtualizace 97
2.5.1. Hyper-V krok za krokem (Windows 8 a 8.1 Pro) 98
2.5.2. Proč Virtualizovat? 107

3. OPERAČNÍ SYSTÉMY 109

3.1. Rozdělení 109

3.2. Obecná struktura 109

3.3. Terminologie 109

3.4. Stavy procesoru 111

3.5. Operační paměť 111
3.5.1. Přidělování OP po blocích 111
3.5.2. Virtuální paměť 112
3.5.2.1. Virtualizace adres 112
3.5.3. Ochrana paměti 113

3.6. Obsluha požadavků na přístup k HDD 113

3.7. Časovač 113

3.8. Něco málo z historie 113
3.8.1. CP/M 114
3.8.2. MS-DOS 114
3.8.3. Rodina Windows obecně 115
3.8.4. Windows 1.0 115
3.8.5. Windows 2.0 116
3.8.6. Windows 3.1 116
3.8.7. Windows 3.11 117
3.8.8. Windows NT 3.1 117
3.8.9. Windows NT 4 118
3.8.10. Windows 95 118
3.8.10.1. Systémové požadavky 119
3.8.11. Windows 98 119
3.8.11.1. Systémové požadavky 120
3.8.11.2. Nejvýkonnější sestava pro Windows 98SE 120
3.8.12. Windows 2000 (NT5) 120
3.8.12.1. Systémové požadavky 121
3.8.13. Windows Me 121
3.8.14. Windows XP (NT5.1) 122
3.8.14.1. V praxi 124
3.8.15. Windows Server 2003 124
3.8.16. Windows Vista (NT6) 125
3.8.16.1. Základní technologie 126
3.8.16.2. Novinky 126
3.8.16.3. Praxe 127
3.8.17. Windows Server 2008 127
3.8.18. Windows 7 (NT 6.1) 127

3.8.18.1.Přehled edic 128
3.8.18.2. Novinky 128
3.8.19. Windows 8 (NT6.2) 128
3.8.19.1. Minimální požadavky 129
3.8.19.2. Novinky 129
3.8.19.2.1. Modern UI 129
3.8.19.2.2. Přihlašovací obrazovka 130
3.8.19.2.3. Obrázkové heslo 130
3.8.19.2.4. Průzkumník 130
3.8.19.2.5. Správce úloh 136
3.8.19.2.6. Modern UI App 138
3.8.19.2.6. Connected standby 138
3.8.19.3. Klady 138
3.8.19.4. Nevýhody 139
3.8.20. Windows 8.1 (NT6.3) 139
3.8.20.1. Minimální požadavky 139
3.8.20.2. Novinky 139
3.8.20.3. Spouštění Windows 139
3.8.20.4. Tlačítko Start 140
3.8.20.4. Přizpůsobení 140
3.8.20.5. Rozšíření 140
3.8.20.6. Update1 140
3.8.20.7. Další informace 140
3.8.21. Windows 10 140
3.8.22. Systém 6 142
3.8.23. Systém 7 142
3.8.24. Systém 8 143
3.8.25. MacOS X 143
3.8.25.1. Snow Leopard 143

3.9. Linux 144

3.10. Mobilní OS 146
3.10.1. Android 146
3.10.2. IOS 147
3.10.3. Symbian 147
3.10.4. Windows mobile 148
3.10.4.1. 6.x 148
3.10.5. Windows Phone 149
3.10.5.1. WP 8.1 150

3.11. Přehled alternativních OS 151

3.12. Závěrem 151

4. MICROSOFT OFFICE 2013 152

4.1. Word 153
4.1.1. Krátké představení 153
4.1.2. Formální úprava dokumentů 157
4.1.2.1. části práce (seminární, diplomová apod.) 157
4.1.2.2. Sazba textu 158
4.1.2.2.1. Okraje stránky 158
4.1.2.2.2. Číslování stránek 158
4.1.2.2.3. Volba typu písma 158
4.1.2.2.4. Řádkování 159

4.1.2.2.5. Dělení slov 159
4.1.2.2.6. Mezery 159
4.1.2.2.7. Interpunkční a další znaménka 159
4.1.2.3. Bibliografické citace 160
4.1.2.3.1. Struktura bibliografických citací 160
4.1.2.4. Odkazy 161
4.1.2.5. Použitá literatura pro vytvoření tohoto návodu 162
4.1.3. Nastavení stylů 162
4.1.3.1. Tvorba obsahu 168
4.1.3.1.1. Číslování stránek 168
4.1.3.1.1. Úkol pro vás čtenáře 169
4.1.3.1.2. Vlastní obsah 170
4.1.4. Nastavení okrajů 171
4.1.5. Jak si šetřit práci s tvorbou odstavců? 172
4.1.6. Konce 175
4.1.7. Záhlaví a zápatí 175
4.1.8. Titulní strana 176
4.1.9. Obrázky 177
4.1.10. Citace 178
4.1.11. Tabulka 180
4.1.12. Grafy 181
4.1.13. Editor rovnic 184
4.1.13.1. Tabulátory, aneb jak očíslovat rovnici 184
4.1.14. Pozadí stránky 186
4.1.15. Uložení do jiných formátů 188
4.1.15.1. Formát docx 189
4.1.15.2. Formát pdf 190
4.1.15.3. Formát XPS 190
4.1.15.3.1. Ke schopnostem formátu 191
4.1.16. Poznámky pod čarou 192

4.2. PowerPoint 193
4.2.1. Pravidla pro tvorbu 193
4.2.2. Krátké představí 194
4.2.2.1. Karta Domů 195
4.2.2.2. Karta vložení 195
4.2.2.3. Karta návrh 195
4.2.2.4. Karta Přechody 196
4.2.2.5. Karta Animace 196
4.2.2.6. Karta prezentace 196
4.2.2.7. Karta zobrazení 196
4.2.3. Základní postup tvorby 196
4.2.4. Obrázky 203
4.2.5. Video 204
4.2.6. Audio 208
4.2.7. Komprese multimédia 208
4.2.8. Čas, datum, číslo snímku, zápatí 210
4.2.9. Přechody 212
4.2.10. Konfigurace prezentace 213
4.2.11. Strojový překlad prezentace 213
4.2.12. Ukládání 214
4.2.13 Tisk a podklady 215

4.3. Excel 217
4.3.1. Desatero Excelu 2013 218
4.3.1.1. Spuštění Excelu, pojmenování, uložení 218

4.3.1.2. Uložení sešitu 219
4.3.1.3. Vytvoření tabulky 220
4.3.1.3.1. Příklad tabulky 221
4.3.1.4. Formátování tabulky 222
4.3.1.4.1. Sloučení buněk 222
4.3.1.4.2. Úprava fontu 222
4.3.1.4.3. Ohraničení 223
4.3.1.4.4. Výpočty v tabulce 224
4.3.1.4.4.1. Součet 224
4.3.1.4.4.2. Výpočet procent 224
4.3.1.5. Rozkopírování vzorce 225
4.3.1.6. Využití vzorců 226
4.3.1.6.1. Vzorec - knihovna funkcí - SUMA 226
4.3.1.7. Vytvoření grafu 227
4.3.1.8. Vložení obrázku, klipartu 230
4.3.1.9.Vložení symbolů, vzorců 231
4.3.1.9.1. Vložení symbolu 232
4.3.1.9.2. Vložení rovnice 232
4.3.1.10. Tisk tabulky s grafem 232
4.3.1.11. Filtry 233
4.3.1.11.1. Vložení Filtru 233
4.3.2. Klávesové zkratky 235
4.3.2.1. Zkratky pro funkční klávesy 235
4.3.2.2. Zkratky s Ctrl 236
4.3.2.3. Zkratky s Alt 237
4.3.2.4. Zkratky se Shift 237
4.3.2.5. Zkratky ostatní 237
4.3.2.6. Navigace uvnitř listu 237
4.3.2.8. Výběr buněk v listu 238
4.3.2.9. Práce s aktivním (označeným) výběrem 238
4.3.2.10. Vkládání speciálních znaků 238
4.3.3. Data v buňkách – co obsahuje, co lze vkládat 239
4.3.3.1. Data v buňce 239
4.3.3.2. Co buňka obsahuje 239
4.3.3.2.1. Hodnota 239
4.3.4. Operátory 243
4.3.4.1. Vzorec 243
4.3.4.2. Aritmetické operátory 243
4.3.4.3. Relační operátory 244
4.3.4.4. Textové operátory 244
4.3.4.5. Odkazovací operátory 244
4.3.4.5.1. Dvojtečka - operátor oblasti 245
4.3.4.5.2. Středník - operátor sjednocení 245
4.3.4.5.3. Mezera – operátor průniku 245
4.3.4.6. Nadřazenost (priorita) operátorů 245
4.3.5. Relativní a absolutní odkazy 246
4.3.5.1. Druhy odkazů 247
4.3.5.2. A1 – klasický 247
4.3.5.3. R1C1 – programátorský 248
4.3.5.4. Změnit styl A1 -> R1C1 a zpět R1C1 -> A1 248
4.3.5.5. Absolutní relativní (smíšené) odkazy 249
4.3.5.5.1. Styl A1 249
4.3.5.5.2. Styl R1C1 249
4.3.5.6. Pokročilejší odkazy 250
4.3.6. Práce s listy 251
4.3.6.1. Výběr (volba) listu 251

4.3.6.2. Skrýt/zobrazit listy 251
4.3.6.3. Vložit list 253
4.3.6.3.1. Změna počtu vložených listu při otevření MS Excelu. 254
4.3.6.4. Odstranit / Smazat list 254
4.3.6.5. Kopírování listu 255
4.3.6.5.1. Klikem myší na záložku 255
4.3.6.6. Přejmenovat list 256
4.3.6.6.1. Klikem myší na záložku 256
4.3.6.6.2. Využít pás karet ke změně názvu listu 257
4.3.6.7. Záhlaví a zápatí 257
4.3.6.8. Zamykání buněk 258
4.3.6.8.1. Zamknout list 258
4.3.6.8.2. Změna zamčené buňky 259
4.3.6.9. Styl buňky 259
4.3.6.9.1. Vytvořit nový styl buňky 260
4.3.6.9.2. Změnit upravit styl 262
4.3.6.10. Formát buňky 262
4.3.6.10.1. Trocha teorie o vlastním formátu 263
4.3.6.10.2. Problém se zobrazováním římských číslic 264
4.3.6.10.3. Skloňování 264
4.3.7. Kontingenční tabulky a grafy 264
4.3.7.1. Kontingenční tabulka 264
4.3.7.1.1. Trochu úprav kontingenční tabulky 268
4.3.7.2. Vytvoření výpočtového pole 269
4.3.7.2.1. Změnit/odstranit počítané (výpočtové) pole 270
4.3.7.2.2. Seznam vzorců 270
4.3.7.2.3. Násobení dvou polí 270
4.3.7.3. Nastavení polí hodnot v kontingenční tabulce 271
4.3.7.4. Seskupení dat – měsíčně 273
4.3.7.5. Vložení průřezu (Slicers) do kontingenční tabulky 274
4.3.8. Kontingenční graf 275
4.3.9. Přehled funkcí 277
4.3.9.1. Funkce – A 278
4.3.9.2. Funkce – B 279
4.3.9.3. Funkce – C 280
4.3.9.4. Funkce – Č 281
4.3.9.5. Funkce – D 281
4.3.9.6. Funkce – E 283
4.3.9.7. Funkce – F 284
4.3.9.8. Funkce – G 284
4.3.9.9. Funkce – H 285
4.3.9.10. Funkce – CH 285
4.3.9.11. Funkce – I 285
4.3.9.12. Funkce – J 287
4.3.9.13. Funkce – K 287
4.3.9.14. Funkce – L 288
4.3.9.15. Funkce – M 288
4.3.9.16. Funkce – N 289
4.3.9.17. Funkce – O 290
4.3.9.18. Funkce – P 291
4.3.9.19. Funkce – Q 292
4.3.9.20. Funkce – R 292
4.3.9.21. Funkce – S 293
4.3.9.22. Funkce – T 294
4.3.9.23. Funkce – U 294
4.3.9.24. Funkce – V 295

4.3.9.25. Funkce – W 295
4.3.9.26. Funkce – X 295
4.3.9.27. Funkce – Y 296
4.3.9.28. Funkce – Z 296
4.3.9.30. Funkce nové v Excel 2013 a 2016 297
4.3.10. Funkce Když 299
4.3.10.1. Syntaxe v češtině 299
4.3.10.2. Popis parametru: 299
4.3.10.3. Parametr – Podmínka – může být: 300
4.3.10.4. KDYŽ – obrácena logika - jedna podmínka 300
4.3.10.4. KDYŽ – použití dvou podmínek – jak vnořovat funkce 300
4.3.10.5. KDYŽ – ještě více podmínek 300
4.3.11. Komplexní čísla a Excel 301
4.3.11.1. Zadání 301
4.3.11.2. Základní operace s komplexními čísly 301
4.3.11.3. Funkce pro práci s komplexními čísly 301
4.3.12. Matice 302
4.3.12.1. Zápis matice 302
4.3.12.2. Operace s Maticemi 302
4.3.12.2.1. Součet matic 303
4.3.12.2.2. Součin matic 303
4.3.12.2.3. Determinant 303
4.3.12.3. Proč matice? 303
4.3.12.3. Transpozice 303
4.3.12.4. Výpočet soustavy rovnic 304
4.3.13. Grafy 305
4.3.13.1. Sloupcový 305
4.3.13.1.1. Sloupcový graf - je vhodný 306
4.3.13.1.2. Sloupcový graf – není vhodný 306
4.3.13.1.3. Druhy sloupcových grafů 306
4.3.13.2. Výsečový graf 307
4.3.13.2.1. Výsečový graf – je vhodný 307
4.3.13.2.2. Výsečový graf – není vhodný 307
4.3.13.2.3. Vložení 308
4.3.13.3. Prstencový graf 308
4.3.13.3.1. Užití 309
4.3.13.3.2. Vložení 309
4.3.13.4. Bodový graf 310
4.3.13.5. Polární graf 313
4.3.13.6. Mini graf 314
4.4. Závěrem 315

5. PRÁCE S UŽIVATELSKÝM ULOŽIŠTĚM 316

5.1. Komprese souborů 316
5.1.1. Komprese jednotky 316
5.1.2. Kompresní formáty 317
5.1.3. Komprese multimédií 318

5.2. Zálohování 319
5.2.1. Migrace profilu uživatele 319
5.2.2. Záloha staré platformy 320

5.3. Vypalování 321
5.3.1. Datové disky 321
5.3.2. Multimediální disky 323

5.4. Defragmentace **325**

5.5. Ukládání souborů **326**

5.6. Závěrem **327**

6. KOMUNIKACE PO INTERNETU 328

6.1. Email **328**
6.1.1. Založení 328
6.1.2. Psaní 329

6.2. Sociální sítě **330**
6.2.1. Co do profilu nepatří? 330
6.2.2. Přezdívka 331
6.2.3. Heslo 331
6.2.4. Profilová fotografie 331
6.2.4. Statusy a sdílené stavy 332
6.2.5. Kontakty 333
6.2.6. Co dělat, když se přeci jen setkáte s úchylem, nebo vás někdo začne vydírat??? 333
6.2.7. Závěrem 333

6.3. Malware **333**
6.3.1. Viry 334
6.3.1.1. Druhy Virů 334
6.3.1.2. Trocha historie 335
6.3.1.3. Ochrana před viry 335
6.3.2. Spyware 335
6.3.2.1. Druhy 336
6.3.2.2. Možnost šíření 336

6.4. Ochrana před malware **336**
6.4.1. Anti-virový a antimalwarový software 336

6.5. Počítačová kriminalita **337**
6.5.1. Útok na počítač, program, údaje, komunikační zařízení 338
6.5.2. Neoprávněný přístup k údajům, (počítačová špionáž) 338
6.5.3. Šíření počítačových zpráv 338
6.5.4. Warez 338
6.5.5. Průnik do systému 339
6.5.6. Bankovní krádeže 340

7. POČÍTAČOVÉ SÍTĚ 341

7.1. Základní komponenty sítí **341**

7.2. Dělení počítačových sítí **341**
7.2.1. Přenosová rychlost 342
7.2.2. Rozsah 342
7.2.3. Typy uzlů 343
7.2.4. Vztahy mezi uzly 343
7.2.5. Topologie 343
7.2.5.1. Sběrnicová 343
7.2.5.2. Kruhová 344
7.2.5.3. Hvězdicová 344

7.2.5.4. Hvězdicová kruhová topologie 345
7.2.5.5. Strukturovaná kabeláž 345
7.2.6. Architektura 345
7.2.7. Možnosti přístupu 345
7.2.8. Pasivní prvky sítě 346
7.2.9. Aktivní prvky sítě 346

7.3. Referenční model OSI 346
7.3.1. Fyzická vrstva 347
7.3.2. Linková vrstva 347
7.3.3. Síťová vrstva 347
7.3.4. Transportní vrstva 347
7.3.4. Relační vrstva 348
7.3.5. Prezentační vrstva 348
7.3.6. Aplikační vrstva 348

7.4. Bezpečnost počítačové sítě 348
7.4.1. Uživatelská práva 348
7.4.2. Síťová a aplikační vrstva 348
7.4.3. Zabezpečení užtelských dat 348
7.4.4. Elektronický podpis 349
7.4.5. Bezpečnostní politika 349
7.4.6. Další vývoj 349

8. MODULACE 350

8.1. Amplitudová modulace 350

8.2. Frekvenční modulace 351

8.3. Fázová modulace 352

8.4. Nekódované diskrétní modulace 352

8.5. Kódované diskrétní modulace 353

8.6. Dvojstavové digitální modulace 353

8.7. Více stavové digitální modulace 354

8.9. Multiplex 355

9. MOBILNÍ SÍTĚ 356

9.1. Obecné principy 356
9.1.1. Metody vícenásobného přístupu 356
9.1.2. Způsoby přenosu 356
9.1.3. Problémy šíření rádiového signálu 357

9.2. Obecný popis bezšňůrových telefonních systémů 357
9.2.1. Analogové bezšňůrové telefonní přístroje 359
9.2.2. Digitální bezšňůrové telefonní přístroje standardu CT 360
9.2.2.1. Systém DECT 361
9.2.2.1.1. Vrstvový model DECT 364
9.2.2.2. Ostatní digitální bezšňůrové telefonní systémy 364

9.2.2.2.1. PHS (Personal Handyphone System) 364
9.2.2.2.2. PACS (Personal Access Communication Services) 365
9.2.2.3. Digitální bezšňůrové telefonní přístroje – přehled 365

9.3. Buňkové telefonní sítě 365
9.3.1. Princip buňkového systému 365
9.3.2. Handover 368
9.3.3. Analogové systémy ve světě 369
9.3.4. Digitální systémy ve světě 370
9.3.4.1. Přehled základních digitálních systémů celulární mobilní komunikace 371
9.3.4.1.1. GSM 371
9.3.4.1.2. ADC, IS-54, D-AMPS 371
9.3.4.1.3. ADC, IS-136, D-AMPS 372
9.3.4.1.4. JDC, PDC 372
9.3.4.1.5. ADC, IS-95, cdmaOne, CDMA 372
9.3.5. Historie systému GSM 373
9.3.6. Základní struktura sítě GSM 373
9.3.6.1. Mobilní uživatelské stanice – MS (Mobile Station) 373
9.3.6.2. Subsystém základnových stanic – BSS (Base Station Subsystem) 374
9.3.6.3. Síťový spojovací subsystém – NSS (Network Switching subsystem) 374
9.3.6.4. Operační a podpůrný subsystém – OSS (Operational and Support Subsystem) 374
9.3.6.5. Systém GSM dále spolupracuje s externími složkami. 374
9.3.6.5.1. MS 374
9.3.6.5.2. BSS 375
9.3.6.5.3. NSS 375
9.3.6.4. Operační a podpůrný subsystém – OSS 376
9.3.6.6. Rádiové rozhraní systému GSM 376
9.3.6.7. Zpracování signálu v systému GSM 378
9.3.6.7.1. Zdrojové a kanálové kódování 379
9.3.6.7.2. Prokládání a formátování burstů 380
9.3.6.7.3. Modulace a demodulace 381
9.3.6.8. Zabezpečení systému GSM proti zneužití 382
9.3.6.8.1. Ověření totožnosti 383
9.3.6.9.2. Šifrovací proces 383
9.3.6.9.3. Vrstvový model, logické kanály 384
9.3.6.9. Přenos dat v systému GSM 386
9.3.6.9.1. Klasický přenos dat 386
9.3.6.9.2. HSCSD 388
9.3.6.9.3. GPRS 389
9.3.6.9.4. EDGE 391
9.3.6.10. IMT-2000 392
9.3.6.11. Základní struktura UMTS 395
9.3.6.12. Rádiové rozhraní – alokace spektra 396
9.3.6.12.1. Alokace spektra UMTS 396
9.3.6.13. Handover v systému UMTS 397
9.3.6.14. Řízení výkonu v UMTS 398
9.3.6.14.1. Řízení výkonu – Uplink 398
9.3.6.14.2. Řízení výkonu – Downlink 399
9.3.6.15. FDD 399
9.3.6.15.1. DS-CDMA 399
9.3.6.15.2. Struktura přenášených dat na fyzické vrstvě 400
9.3.6.15.3. Přehled parametrů režimu FDD 402
9.3.6.16. TDD 402
9.3.6.16.1. TD-CDMA 402
9.3.6.16.2. Přehled parametrů režimu TDD 405

10. BEZDRÁTOVÉ TECHNOLOGIE 406

10.1. IrDA **406**
10.1.1. Fyzická vrstva standardu IrDA 406
10.1.1.1. IrDA 1.0 406
10.1.1.2. IrDA v.1.1 407
10.1.2. Protokoly technologie IrDA 408

10.2. Bluetooth **409**
10.2.1. Základní charakteristika technologie Bluetooth 409
10.2.2. Zpracování signálu 411

10.3. IEEE 802.11(Wi-Fi) **413**
10.3.1. Infrastruktura 415
10.3.1.1. Ad hoc 415
10.3.1.2. Fyzická a spojová vrstva standardu IEEE 802.11 415
10.3.1.3. Fyzická vrstva standardu 802.11 416
10.3.1.4. Spojová vrstva standardu 802.11 417

10.4. Paging **418**
10.4.1. Rádiové rozhraní 420
10.4.2. Pagingový systém ERMES 421
10.4.2.1. Služby sytému ERMES 422
10.4.3. Ostatní pagingové systémy 423
10.4.3.1. AUTORUF 423
10.4.3.2. EUROSIGNAL 423
10.4.3.3. POCSAG (Post Office Code Standardization Advisory Group) 423
10.4.3.4. FLEX (Flexible Speeded Wide Area Synchronous Protocol), ReFLEX, InFLEXion 424
10.4.3.5. pACT (personal Air Communications Technology) 424
10.4.3.6. RDS (Radio Data System) 425

11. DRUŽICOVÁ MOBILNÍ KOMUNIKACE 426

11.1. Oběžné dráhy družic **426**

11.2. INMARSAT **427**
11.2.1. INMARSAT-A a INMARSAT-B 427
11.2.2. INMARSAT-C 427
11.2.3. INMARSAT-D a D+ 428
11.2.4. INMARSAT-M 428
11.2.5. INMARSAT-miniM 428
11.2.6. INMARSAT-E 428

11.3. IRIDIUM **428**
11.3.1. Kosmický segment 429
11.3.2. Uživatelský segment 430
11.3.2.1. Regionální pozemské stanice 430
11.3.3. Řídicí segment 430
11.3.3.1. Služby systému IRIDIUM 430

11.4. GLOBALSTAR **432**
11.4.1. Kosmický segment 433
11.4.2. Uživatelský segment 433
11.4.3. Regionální pozemské stanice 434
11.4.4. Služby 434

11.5. Další systémy 2. Generace **435**
11.5.1. ICO (dříve označovaný Inmarsat P) 435
11.5.2. Odyssey TRW 435
11.5.3. TELEDISC 436

11.6. Mapa světa a používané technologie mobilní komunikce **436**

12. MOBILNÍ TELEFON 438

13. MATICE 440

13.1. Základní pojmy **440**

13.2. Speciální typy matic **440**

13.3. Základní operace s maticemi **441**
13.3.1. Násobení matic 442

13.4. Základní maticové úpravy **443**
13.4.1. Lineární závislost 444

13.5. Regulární a singulární matice **445**

13.6. Programy pro práci s maticemi **445**

13.7. Matice v PC grafice **446**

13.8. Závěrem **451**

14. ZÁKLAD POČÍTAČOVÉ GRAFIKY 452

14.1. Rasterová grafika **452**

14.2. Vektorová grafika **453**

14.3. Základní barevné systémy (modely) **454**
14.3.1. RGB 454
14.3.2. CMY (resp. CMYK) 455
14.3.3. HSV 455

14.4. Barevná hloubka **456**

14.5. Grafické formáty **456**
14.5.1. JPG 456
14.5.2. BMP 456
14.5.3. GIF 456
14.5.4. PNG 457
14.5.5. RAW 457
14.5.6. Vektorové formáty 457
14.6. Editory 457
14.6.1. Zoner Photostudio 458
14.6.1.1. Ořezávání 461
14.6.1.1.1. Specifický ořez 462
14.6.1.2. Otočení obrázku 462

16.6.1.3. Srovnání horizontu 464
14.6.1.4. Korekce červených očí 464
14.6.1.5. Skládání 3D fotografií 464
14.6.1.6. Skládání panoramatických snímků 468

15. POČÍTAČOVÁ SAZBA 471

16. WEBOVÉ STRÁNKY, ZÁKLADNÍ SEZNÁMENÍ 472

16.1. Základní rady a principy 472

16.2. Co je tedy html? 473

16.3. Syntaxe 473

16.4. Současné www, tedy standart xhtml 5 473

16.5. Kódování pro český jazyk 474

16.6. V čem psát? 474

16.7. Základní elementy hlavičky 476

16.8. Elementy 476
16.8.1. Řádkové elementy 476
16.8.2. Blokové Elementy 477

16.9. Rady pro usnadnění práce 480

17. PROGRAMOVÁNÍ 481

17.1. Definice problému 481

17.2. Algoritmus 482
17.2.1. Vlastnosti algoritmu 482
17.2.2. Zápis algoritmu 482
17.2.3. Vývojové diagramy 482

17.3. Výraz 483
17.3.1. Podmínka 483

17.4. Příkazové struktury 483
17.4.1. Cykly 484

17.5. Sestavení programu 485

17.6. Ověření správnosti 485

17.7. Příklad návrhu programu 485

17.8. Vývojová prostředí 486
17.8.1. MS Visual Studio 2013 486
17.8.1.1. Tvorba konsolové aplikace 487
17.8.1.2. Psaní kódu 489
17.8.1.3. Sestavení projektu 490

17.8.1.4. Tvorba setupu 491
17.8.2. BlueJ 495
17.8.3. Komponenty vývojového prostředí jazyka Java 505
17.8.3.1. Editor 505
17.8.3.2. Překladač 505
17.8.3.3. Interpret 505
17.8.3.4. Preprocsor 505
17.8.3.5. Linker 506

18. JAZYK JAVA 507

18.1. Provádění programu 507

18.2. Objekty 507

18.3. Třídy 507
18.3.1. Třída Math 507
18.3.2. Třída string 508

18.4. Konvence jazyka 509
18.4.1. Komentáře 509
18.4.2. Struktura programu 509

18.5. Základní pojmy 510
18.5.1. Identifikátor 510
18.5.2. Klíčové slovo 510
18.5.3. Oddělovač 510
18.5.4. Proměnná 510
18.5.4.1. Celočíselné datové typy 510
18.5.4.2. Datové typy reálných čísel 511
18.5.4.3. Datový typ znak 511
18.5.4.4. Logické proměnné 511
18.5.4.5. Deklarace a inicializace 511
18.5.4.6. Platnost proměnné 513
18.5.4.7. Přetypování proměnné 514

18.6. Operace 515
18.6.1. Výstup 515
18.6.1.1. Formátovaný výstup 515
18.6.2. Vstup 516
18.6.3. Aritmetické 517
18.6.4. Inkrementace a dekrementace 517
18.6.5. Modulo 517
18.6.6. Operace se znaky 518
18.6.7. Binární operace 518

18.7. Balíčky 518

18.8. Metody 519
18.8.1. Návratové hodnoty 519
18.8.2. Předávání parametrů 519
18.8.2.1. Příklad 520
18.2.1.2. Gettery a settery 521

18.9. Podmínky 521
18.9.1. Ternární výraz 522

18.9.2. Příkaz swith 523

18.10. Konstanty 524

18.11. Cykly 524
18.11.1. While 524
18.11.1.1. Příklad 524
18.11.2. Do while 524
18.11.2.1. Příklad 524
18.11.3. for 525
18.11.3.1. Příklad 525
18.11.4. Příkazy break a continue 526
18.11.4.1. Návěští 526

18.12. Pole 526
18.12.1. Vícerozměrná pole 528
18.12.2. Bubble sort 528

18.13. Dědičnost 529
18.13.1. Třída Object 529
18.13.2. Extends 529
18.13.2.1. Specifikátory přístupu 529
18.13.2.2. Vícenásobná dědičnost 529
18.13.2.3. Super 530
18.13.2.4. Příklad 530

18.14. Polymorfizmus 532

18.15. Abstraktní třídy 534
18.15.1. Klíčové slovo abstract 534

18.16. Rozhraní 537
18.16.1. Konvence 537
18.16.2. Příklad 537

18.17. Výjimky 538
18.17.1. Fail-fast 538
18.17.2. Typy výjimek 538
18.17.2.1. Error 539
18.17.2.2. Runtime exception 539
18.17.2.3. Exception 539
18.17.3. Vyvolávání výjimek 539
18.17.4. Try-catch-finally 540
18.17.5. Obvyklé zpracování výjimek 541
18.17.5.1. Neošetření 541
18.17.5.2. Přebalení na kontrolovanou výjimku 541
18.17.5.2.1. Příklad 541
18.17.5.3. Přebalení výjimky na RuntimeException 542
18.17.5.3.1. Příklad 542

18.18. Vnořené třídy 543
18.18.1. Příklad 543

18.19. Anonymní třídy 545
18.19.1. Konstrukce 545
18.19.1.1. Příklad 545

18.20. Spojový seznam **546**
18.20.1. Rozhraní (spojového) seznamu 546
18.20.2. Implementace 546
18.20.2.1. Třída Node 547
18.20.2.1.1. Metoda add 547
18.20.2.1.2. Metoda get 548
18.20.2.1.3. Metoda remove 548
18.20.2.1.4. Metoda size 549
18.20.2.1.5. Metoda toString 549
18.20.3. Příklad 549

18.21. Generika **552**
18.21.1. Parametrizování třídy/rozhraní 552
18.21.1.1. Pojmenování typového parametru 553
18.21.1.2. Parametr v rozhraní ListIface 553
18.21.1.3. Typový parametr třídy MyLinkedList 553
18.21.2. Generika umí více 556
18.21.3. Co generika neumí 556

18.22. Iterátor **557**
18.22.1. Spojový seznam s užitím generik a iterátoru 558

18.23. Proudy a serializace **562**
18.23.1. Typy proudů 562
18.23.2. Decorator 563
18.23.2.1. Příklad 563
18.23.3. Binární proudy 563
18.23.3.1. BufferedStream 563
18.23.3.2. Serializace 564
18.23.3.3. Metoda reset 566
18.23.4. Znakové proudy 567
18.23.4.1. Příklad 567
18.23.4.1.1. Zjištění kódování textového souboru 568

18.24. Výčtový typ **568**
18.24.1. Výhody výčtového typu 568
18.24.2. Deklarace 569
18.24.3. Omezení 569
18.24.4. Společný předek – třída Enum 569
18.24.4.1. Příklad 569

18.25. Vlákna **571**
18.25.1. Co je to vlákno? 571
18.25.2. Procesy 571
18.25.3. Vlákna vs. procesy 571
18.25.4. Vlákna a procesy v Javě 572
18.25.4.1. Jednoduchá aplikace 572
18.25.4.2. Vícevláknová aplikace 572
18.25.5. Synchronizace 575
18.25.6. Neatomická inkrementace 576
18.25.6.1. Volatile 576
18.25.6.2. Synchronized 576
18.25.7. Synchronizované metody 577
18.25.8. Deadlock 579
18.25.9. Komunikace vláken 580
18.25.9.1. Wait, notify, notifyAll 580

18.25.9.2. Spurious wakeup 580

18.26. Grafické rozhraní 582
18.26.1. Abstract Window Toolkit 582
18.26.1.1. Swing 582
18.26.1.2. Společné vlastnosti AWT a Swingu 582
18.26.1.3. Zpracování událostí 582
18.26.2. Volání 583
18.26.3. Tvorba rozhraní 584
18.26.4. Přiřazování akcí komponentám 584
18.26.5. Návrhový vzor observer 584
18.26.5.1. Příklad v praxi 584
18.26.6. Dědění vlastních komponent 586
18.25.7. JPanel 586
18.25.8. Správci rozmístění 586
18.25.8.1. FlowLayout 586
18.25.8.2. BorderLayout 587
18.25.8.3. BoxLayout 588
18.25.8.4. GridLayout 589
18.25.8.5. GridBagLayout 590

18.26. Závěrem 591
18.26.1. Architektura 591
18.26.2. Proč aplikaci členit zrovna tímto způsobem? 592
18.26.2.1. Single responsibility 592
18.26.2.2. High cohesion a low coupling 592
18.26.3. Implementace 592
18.26.3.1. Model 592
18.26.3.2. CalculatorActionIface 594
18.26.3.3. AbstractCalculatorAction 595
18.26.3.4. DecimalPointAction 595
18.26.3.5. NumberAction 596
18.26.3.6. ResetAction 596
18.26.3.7. ResultAction 597
18.26.3.8. OperatorActionIface 597
18.26.3.9. AbstractOperatorAction 598
18.26.3.10. PlusAction, MinusAction, MultiplicationAction, DivisionAction 598
18.26.4. View 600
18.26.5. Calculator 606

19. JAZYK C++ 607

19.1. Přenositelnost a standardy 607

19.2. Postup psaní programu v C++ 608

19.3. Jmenné prostory 609

19.4. Paměť 610
19.4.1. Celočíselné datové typy 610
19.4.2. Datový typ pro reálná čísla 610
19.4.3. Textové datové typy 610
19.4.4. Logické datové typy 611
19.4.5. Deklarace proměnných 611
19.4.6. Přiřazení hodnoty do proměnné 611
19.4.7. Zobrazení hodnoty proměnné na konzoli 612

19.4.8. Přetečení a podtečení 613

19.5. Aritmetické operátory 614
19.5.1. Operátor sčítání 614
19.5.2. Operátor odčítání 614
19.5.3. Priority operátorů 614
19.5.4. Operátor dělení 614
19.5.5. Operátor modulo 614
19.5.6. Mocnina 614
19.5.7. Zkrácená forma zápisu 615
19.5.8. Příklad na výpočet školného 615

19.6. Píšeme román, aneb větvení pomocí if 615

19.7. Vícepolohový přepínač swith 616

19.8. Vnořené podmínky 621

19.9. Logické operátory, aneb slučuje podmínky 621

19.10. Cykly 622
19.10.1. Cyklus for 622
19.10.2. Cyklus While 623
19.10.3. Cyklud do while 624

19.11. Funkce 624
19.11.1. Předávání parametrů 626
19.11.2. Návratová hodnota funkce 626

19.12. Pole 627
19.12.1. Deklarace 627
19.12.2. Konstanty 628
19.12.3. Inicializace 628
19.12.4. Načtení hodnot do pole a výpis pole na konzoli 628

19.13. Abychom se neztratili v paměti, aneb ukazatele 631
19.13.1. Deklarace 631
19.13.2. Dereferencování ukazatelů 631
19.13.3. Dynamické alokování paměti 632

19.14. Znaky a třída string, aneb naučme počítač číst 634
19.14.1. Čtení z klávesnice do stingu 635
19.14.1.1. Čištění Bufferu, aneb cin.ingnore 635
19.14.2. Porovnávání řetězců 635
19.14.3. Převod řetězce na číslo 635

19.15. Soubory, neboli zachraňme data 636
19.15.1. Souborové datové proudy 636
19.15.1.1. Životní cyklus práce se souborem 636
19.15.2. Otevření souboru pro zápis 636
19.15.2.1. Členská funkce open 636
19.15.2.2. Konstruktor 637
19.15.2.3. Zápis do souboru 637
19.15.3. Otevření souboru pro čtení 638
19.15.4. Otevření souboru pro zápis i čtení 638
19.15.4.1. Kontrola otevření souboru 639

19.15.5. Uzavření souboru 639

19.16. Struktury, neboli sjednocení dat 639
19.16.1. Deklarace 639
19.16.2. Instance struktury 640
19.16.3. Přístup k členským proměnným 640
19.16.4. Inicializacce 640
19.16.5. Lineární spojový seznam 640

19.17. Třídy 642

19.18. Seznam užívaných objektů standartní knihovny 643

19.19. Seznam užívaných operátorů 643

19.20. Seznam využívaných funkcí 644

19.21. Vybrané třídící alegoritmy 644
19.21.1. Bubblesort 644
19.21.2. Insertion sort 646
19.21.3. Mergesort 647
19.21.4. Quicksort 649
19.21.4.1. Nerekurzivní varianta 650
19.21.5. Selection sort 652

20. DATABÁZE 654

20.1. Základní pojmy 654

20.2. Systémy řízení báze dat 654
20.2.1. Micorosoft Access 2016 655
20.2.2. Microsoft Management studio 2016 657
20.2.3. Připojení se k SQL serveru pomocí Visual studia 658

20.3. Tvorba databází 661
20.3.1. Relační model 663
20.3.2. Klíče 664
20.3.3. Datové modelování 665
20.3.4. Entity-relationship model 665
20.3.5. Databázový návrh 667
20.3.5.1. Normální formy 668
20.3.5.2. Vztahy mezi silnými entitami 668
20.3.5.3. Ostatní vztahy mezi entitami 668
20.3.5.4. Vybrané datové typy 669
20.3.5.5. Příklad databázového návrhu 671
20.3.5.5.1. Tvorba v Access 672
20.3.5.5.2. Tvorba v MS Management Studiu – grafické nástroje 676

20.4. Jazyk SQL 682
20.4.1. Definování dat pomocí SQL 684
20.4.2. Přehled vybraných příkazů jazyka SQL 687
20.4.3. Dotazy v jazyce SQL 690
20.4.4. Agregační funkce MS SQL server 692
20.4.5. Transakce 693
20.4.6. Techniky T-SQL 694
20.4.6.1. Podmíněné zpracování 694

20.4.6.2. Řízení toku 695
20.4.6.3. Kurzory 696
20.4.6.4. Uložené procedury 697
20.4.6.5. Uživatelsky defniované funkce 699
20.4.6.6. Odchytávání výjimek 700

20.5. Administrace Databází 701
20.5.1. Uzamykání prostředků 702
20.5.2. Úroveň izolace transakcí 703
20.5.3. Kurzory 703
20.5.4. Zabezpečení databáze 704
20.5.4.1. Uživatelské účty 704
20.5.4.2. Zabezpečení na úrovni databázového systému 705
20.5.4.3. Záloha a obnovení 705
20.5.4.4. SQL server hacking 707
20.5.5. Execution plan 708
20.5.5.1. Statistiky MS SQL serveru 709
20.5.5.2. Trubleshooting SQL pomocí Extended Events 711
20.5.6. SQL Server Data Tools 711
20.5.7. Columnstore indexy 711
20.5.8. Pár typů pro administrátory SQL serveru 2016 712

20.6. MS Access 2016 pokračování 713
20.6.1. Šipky a jejich význam 713
20.6.2. Tvorba dotazu 715
20.6.2.1. SQL dotaz 716
20.6.3. Formuláře 717
20.6.3.1. Tvorba 717
20.6.3.2. Dropdown box ve formuláři 719
20.6.4. Vytvoření sestavy 723
20.6.5. Vytvoření makra 725

20.7. Big data 727
20.7.1. Klasický způsob využití dat z datového skladu 727
20.7.2. Změna podstaty big data 728
20.7.3. Nové přístupy k analýze a zpracování big data 728
20.7.4. Vybavení pro práci s Big data 729
20.7.5. Big data znamenají budoucí příležitost 729

21. PROSTŘEDKY IDENTIFIKACE UŽIVATELE 730

21.1. Identifikace 730

21.2. Autentizace 730
21.2.1. Autentizace znalostí 730
21.2.2. Autentizace vlastnictvím 731
21.2.3. Autentizace založená na subjektu 731
21.2.4. Vícefaktorová autentizace 731
21.2.5. Autentizace systému 731
21.2.6. Autentizace osob mimo informační systémy 731
21.2.7. Útoky v prostředí internetu 732
21.2.8. Adaptivní autentizace 734
21.2.9. Čipové karty 734
21.2.10. OTP generátory 738
21.2.11. FIDO aliance 739
21.2.12. Biometrie 739

21.2.12.1. Otisky prstů 742
21.2.12.2. Obraz obličeje 745
21.2.12.3. Biometrický podpis 746
21.2.12.4. Psychologické aspekty biometrie 748
21.2.12.5. Ochrana a podvežení biometrických údajů 749
21.2.13. Kerberos protokol 749
21.2.14. Federalizace identit 750

21.3. Autorizace **751**

22. OBSAH PŘILŽENÉHO DVD 754

23. ZDROJE 755

24. REJSTŘÍK 756

1. Počítač



1.1. Co je počítač?

Na první pohled jednoduchá otázka, která by mohla leckoho i urazit. Odpovědět na ni se nedá nijak jednoznačně a lehce, pro každého z nás je počítač něco jiného. Pro jednoho je to zdroj obživy, pro druhého komunikační nástroj, třetí to má jako hračku a takhle bych mohl pokračovat ještě hodně dlouho.

Jak napovídá název, tak se jedná o stroj, který slouží k výpočtům, to je primárním úkolem počítače, nicméně lze užít pro spoustu dalších činností, které usnadňují lidem práci. O počítači platí to samé jako o ohni, je to dobrý sluha, ale zlý pán. Pokud se na počítač podívám očima chemika, tak je to hromádka křemíku, zlata, platiny, bóru, germania a bůhví čeho ještě. Pokud se na počítač podívám očima fyzika, tak je to složité elektrické zařízení využívající elektrostatiku i elektrodynamiku a různá záření. Každý pohled na počítač je jiný. První počítače byly dokonce mechanismy z ozubených kol a určovali postavení planet.

Z pohledu informatika je počítač zařízení zpracovávající vstupní data pomocí určitého algoritmů (obecného řešení dané skupiny problémů) a vrátí zpracovaná data na svůj výstup.

1.2. Jak počítač funguje?

Počítač si převezme od člověka, nebo jiného stroje data. Člověk může vložit do počítače data pomocí klávesnice, skeneru, fotoaparátu, videokamery a podobně. Stroj může vložit data například pomocí výstupu z teploměru, tlakoměru, čtečky čárového kódu. Počítač přijatá data převede do strojové podoby (tzv. strojový neboli dual kód, což jsou jedničky a nuly) a následně s daty provede požadovanou činnost (výpočet, uložení, odeslání atd.). Informace o průběhu operace a výsledek nám počítač vrátí v podobě výstupních dat na monitor či vytisknutím na tiskárně, obrazu, zvuku či videa, tedy v podobě, které člověk rozumí.

Když jsou data od uživatele přijata, tak je počítač uloží dočasně do operační paměti (známé jako RAM), odkud si je přebírá procesor, který vykoná vlastní požadovanou činnost a vrátí je do operační paměti. Z operační paměti mohou putovat na pevný disk (HDD), USB jednotku, to vždy v podobě nějakého souboru, nebo do grafické či zvukové karty, které přepracují data do jiné podoby, kterou je schopno zpracovat výstupní zařízení, což je monitor či reproduktory. Pokud není využita ani jedna možnost, tak je na řadě již jen trojice řešení, data putují do tiskárny nebo do síťové karty, která data připraví na odeslání do jiného počítače. Poslední možností je, že data zůstanou v operační paměti a procesor se k nim vrátí, aby je znovu zpracoval. Co se s daty stane, záleží na tom, jaký algoritmus (program a úkon) uživatel zvolí pro jejich zpracování.

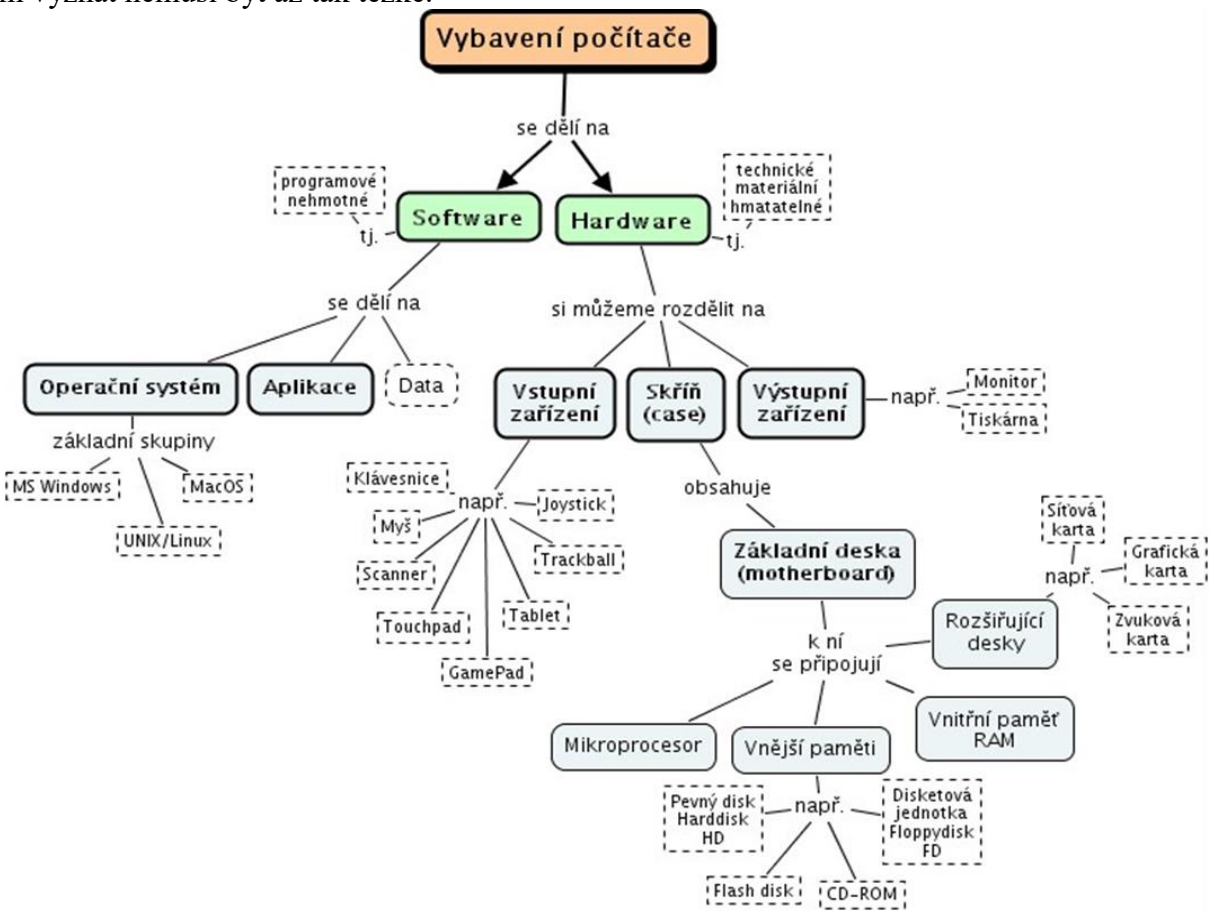
Zpracování dat řídí software (SW), operační systém a programy, ale provádí jej hardware (HW), což je ta část počítače, které se můžeme dotknout nebo je ukryta v „těle“ počítače.

1.3. Z čeho se počítač skládá?

Opět se nám zde dostává na první pohled jednoduchá otázka, která má v pravdě trošku složitější odpověď. V takovém tom prvním náhledu můžeme říct, že se počítač skládá se softwaru (logiky) a hardwaru (fyzických komponent). Pokud přijdeme ke stolu, kde je uložen počítač, tak vidíme periferní zařízení (monitor, klávesnici, myš, tiskárnu, reproduktory) a skříň, o jejímž obsahu mohou trošku napovědět štítky a známky na ni nalepené. Skříň ukrývá všechny nejdůležitější fyzické komponenty počítače.

Software můžeme rozdělit opět do několika skupin: operační systém, aplikace a speciální boot nástroje. Boot nástroje jsou samostatnou skupinou, protože tyto aplikace nepotřebují ke své činnosti operační systém, ale jedná o vysoce specializované a jednoúčelové nástroje sloužící třeba k přesunu dat mezi disky, odvirování počítače a podobně.

Pro lepší představivost jsem se rozhodl vytvořit jakéhosi pavouka, který ukazuje, z čeho se počítač skládá a kde v něm co najdeme. Pravda, počítač není jednoduché zařízení, ale i tak se v něm vyznat nemusí být až tak těžké.



1.4. Hygiena

Je mi jasné, že většina z vás se chytá za hlavu, co v knize o počítačích dělá toto slovo. Každý si pod ním představí mytí rukou, čištění zubů a další činnosti spojené s běžným životem. I programátoři, správci sítí a další pracující s počítači jsou lidé, takže musí dodržovat jistou hygienu, která se pravda liší od té, kterou při práci dodržují kuchaři či lékaři. Hygiena práce s PC se týká ne jen profesionálů, ale i vás, ať je to doma, ve škole, nebo kdekoli jinde a to bez

ohledu na to, zda máte notebook, nebo stolní počítač. Dodržováním hygieny předejdete zánětům šlach na rukou, na které nemálo trpí lidé, kteří s počítači pracují celý život, zhorčování zraku, problémům s lokty, kyčlemi, koleny, ale především páteří.

Abych jen nestrašil, tak musím prozradit, jak těmto problémům spojeným s prací na PC předcházet. Základem je správný posed u počítače. Nohy by měly v kyčlích svírat pravý úhel a v kolenou pravý nebo tupý úhel. Při položení rukou na klávesnici by lokty měly být rovněž v pravém úhlu. Monitor bychom měli mít tak, abychom hlavu měly rovně a vodorovně před očima měly 2/3 výšky obrazu. Správný posed je zobrazen na následujícím obrázku.



Problémům s očima se dá předcházet koupí vhodného monitoru. Monitor by měl mít vysokou obnovovací frekvenci, obraz by měl být tvořen pomocí LC krystalů a jako podsvícení by měly být užity LED. Monitor by neměl být ani malý ani velký, ideální jsou velikosti v rozmezí 14 až 19 palců. Dalším důležitým faktorem je správné nastavení jasu, kontrastu a písma na obrazovce. Pokud sedíme standardně u PC, tak bychom měly mít vždy rozsvícenou nějakou lampu či lustr, který zajistí, že monitor nebude mít výrazně vyšší jas, nežli jeho okolí. Měli bychom stejně dobře vidět jak na monitor, tak třeba na cedulku opřenou o kousek vedle. Jestliže u počítače trávíte většinu dne, pracujete s ním dlouhé hodiny v kuse, tak doporučuji nosit brýle bez dioptrií, pokud nemáte své dioptrické, které budou vybaveny antireflexem. Tyto brýle jsou vhodné i pro profesionální řidiče a to zejména v ranních hodinách a v podvečer. Další ochranný prvek znáte nejspíše z firem, pokud pracujete v kanceláři, jde o takzvanou pauzu za zobrazovací jednotkou, tedy o nic menšího, nežli o přerušení práce zhruba po dvou hodinách a změnu pozice, nechat oči odpočívat ostřením do dálky. Tato pauza by měla mít 5 až 10 minut, pokud se bude jedna to kratší úseky, dvě minuty, tak je nutné tyto úseky vkládat zhruba po vyučovací hodině (45 minut). Pokud zvažujeme koupi notebooku, tak nedoporučuji lesklou obrazovku, tyto obrazovky jsou pak problematické při užívání na terase nebo jinde venku. Sám mám notebook s lesklým displejem, a když jsem si vzal počítač do venkovního přístřešku, abych nemusel být při své práci stále zavřený doma, tak jsem neustále hledal kurzor myši, protože jsem přes odlesky neviděl pořádně na obrazovku.

Co se týká zánětů šlach, tak tam je to trošku komplikovanější. Pokud používáme myš, tak riziko tohoto onemocnění snížíme volbou vhodné myši. Myš by nám měla umožnit nechat zápěstí co nejpřirozeněji ležet na podložce, zápěstí by se nemělo nijak moc zvedat. Celkově jsou vhodné ergonomické myši nebo spíše plošší myšky. Pomoci může také kvalitní podložka

pod myš. Je dobré využít toho, že myši nejsou většinou uzpůsobeny konkrétní ruce, takže při práci ruce střídat. Pokud užíváme notebook, tak je vhodné střídat užívání myši a touchpadu.

1.5. Abychom si rozuměli…

Nějak jsem se rozepsal, to víte, když se jeden pro něco v životě zapálí, tak je to fajn, protože v krematoriu by již bylo pozdě, a ani mi nedošlo, že si nemusíme vlastně ani pořádně rozumět. Věřím tedy, že zatím jste mi všichni rozuměli, ale i tak bych měl uvést věci na pravou míru a vysvětlit alespoň některé pojmy, které jsem použil, či mám v úmyslu použít dříve, nežli se mi v tom začnete ztrácet a rozhodnete se svět počítačů opustit. Ve věcech je nutné mít jasno, aby se dalo pochopit souvislosti, takže pojďme na to.

1.5.1. Software

Jde o nehmatatelnou součást počítače, která jej celý řídí. Jinými slovy by šlo říct, že tvoří duši počítače. Běžně jej označujeme zkratkou SW a můžeme jej rozdělit do dvou velkých skupin: operační systém a aplikace. SW se stará o provedení uživatelem požadované operace a o komunikaci uživatele s počítačem.

Operační systém (značen OS) je sada nástrojů, aplikací a rozhraní, která umožňuje uživateli a programům komunikaci s fyzickou strukturou počítače. Operační systém též umožňuje zprávu aplikací, dat a datových uložišť. Stará se o celkové řízení počítače. Příkladem je Windows, Linux, MacOS, BeOS, DOS, UNIX a další.

Aplikace, neboli program, je takový software, který provádí přímo zpracování dat dle požadavků a povelů uživatele. Příkladem aplikace je webový prohlížeč, videopřehrávač, textový editor a podobně.

Ovladač je speciální software, který má za úkol řídit komunikaci konkrétní součást uvnitř počítače, například grafické karty, s operačním systémem.

Firmware je program, který je uživateli nepřístupný a je uložený přímo v konkrétní součásti počítače a stará se o její ovládání na základě pokynů přejatých od ovladače.

BIOS se stará o překlad komunikace mezi softwarem a hardwarem. Je to speciální program, který je umístěný na základové desce.

UGI je grafické rozhraní pro komunikaci s uživatelem

Malware je škodlivý SW, který má často samo kopírovací zdrojový kód

Zdrojový kód je program zaspaný v programovacím jazyce. Zdrojový kód je textová podoba programu, tak jak ji napíše programátor.

Spustitelný soubor je zdrojový kód, který je přeložený do strojového kódu, který již dokáže být vyhodnocen procesorem.

1.5.2 Hardware

Je fyzickou strukturou počítače. Označujeme jej jako HW a s trochou nadsázky je to přesně to, do čeho praštíme, když se nám něco nedaří či sekne. Za HW považujeme i pomocná zařízení připojená k PC. Jestliže SW tvoří duši, tak HW je tělem počítače.

Mechanická část, je taková, která je tvořena elektromotorem a převody. Komponenty obsahující mechanické části jsou pevné disky a optické mechaniky.

Polovodič je speciální kov, který vede proud jen jedním směrem, např,: Si či Ge.

Dioda je součást tvořená dvojicí polovodičů a jedním přechodem mezi nimi.

Tranzistor je součást tvořená dvojicí polovodičů a dvojicí přechodů mezi nimi.

Vzorkovací frekvence je frekvence měření amplitudy analogového signálu a určí se jako $f_{vz} > 2f_{max}$, kde f_{max} je největší možná frekvence analogového signálu.

1.5.2.1. Jednotky užívané v IT

- Hz – základní jednotka frekvence (taktování)
- B = byte – základní jednotka objemu dat
- b = bit – základní jednotka paměťového místa
- b/s = bit za sekundu = základní jednotka „rychlosti“ přenosu dat
- ot/min = otáčky za minutu (udávají se u disku a odvíjí se od nich rychlost)
- GB/s = jednotka propustnosti
- V = volt – napětí
- A = ampér – proud
- dpi = počet bodů na palec – rozlišení
- lux – světelní intezita
- W – watt – příkon
- nm = nanometr – jednotka délky
- “ = palec = 2,54 cm – jednotka délky užívaná v úhlopříčkách
- °C – základní jednotka teploty

Převody jednotek – řazeno sestupně					
Paměť				1B	8b
Objem dat	GB	1024 MB	1048576 kB	1073741824 B	8589934592 b
Frekvence	GHz	1000 MHz	1000000 kHz	1000000000 Hz	$1*10^{-9}$ s
				1 Hz	$1*10^{-1}$ s
Čas	1 den	24 h	1440 min	86400 s	
		1 h	60 min	3600 s	
			1 min	60 s	

Snad jsem na nějakou jednotku, která se užívá běžně v IT, nezapomněl. I zde dodržujeme fyzikální zákony, takže platí jednotky soustavy SI a základní převody pomocí předpon, tak je známe z hodin fyziky.

V informatice se pracuje s dvojkovou soustavou (tzv. binárem), a protože 2^{10} je 1024 což je nejpodobnější fyzice (užíváme klasickou desítkovou soustavu), tak diference mezi předponami není 1000 (10^3), ale 1024. Tedy na vysvětlenou toho šíleného souvětí, 1MB = 1024 kB = 1024*1024 B, zatím co ve fyzice máme 1MV = 1000 KV = 1000*1000 V.

1.5.3. Číselné soustavy

Protože počítačová HW pracuje s napětím, které buď je, nebo není na daném místě přítomno, tak je jasné, že by se asi velmi složitě dosahovalo práce v desítkové soustavě. Z důvodu jednoduchosti, rychlosti a stability se proto v informatice používá dvojková soustava, neboli binár. Pro SW se pro zrychlení a zpřehlednění logiky užívá hexadecimální soustava, která je následně BIOSem převedena do bináru a poskytnuta HW.

1.5.3.1. Binární soustava

Je číselná soustava která používá pouze dva symboly - nejčastěji 0 a 1. Dvojková soustava je poziční číselná soustava mocnin čísla 2.

Abychom číslo z desítkové soustavy převedli do dvojkové, tak máme dvě možnosti: dělíme číslo dvěma a sepisujeme zbytky, které potom sestavíme zdola nahoru (ukážeme si na příkladu), nebo pomocí vah, kdy hledáme nejvyšší mocninu čísla dvě, která se do čísla vejde,

poté hodnotu této mocniny odečteme do čísla a s výsledkem postup opakujeme, dokud nedosáhneme nuly. Pak pro lidi, kteří převádí často a větší číselné hodnoty je možné užít tabulku tzv. strojového kódu.

Příklad převodu pomocí dělení dvěma

$17_{10} = x_2$

Dolními indexy u čísel značíme soustavu, ve které je číslo zapsáno. Postup je jednoduchý, číslo 17 vydělíme 2 a napíšeme si i zbytek. Použijeme funkci modulo, neboli celočíselné dělení, tak, jak se jej učí děti již ve 2 třídě základní školy.

17 : 2 = 8 zb. 1

Výsledek opět dělíme dvojkou a sepisujeme zbytky a to i v případě, že zbytek je 0.

8 : 2 = 4 zb. 0
4 : 2 = 2 zb. 0
2 : 2 = 1 zb. 0
1 : 2 = 0 zb. 1

Nesmíme zapomenout, že číslo 1 je také dílčí výsledek a je tedy nutné jej dělit dvěma. Dělení opakujeme až do chvíle, kdy dosáhneme nuly.

Výsledek vznikne jak již bylo řečeno sepsáním zbytků zdola nahoru, takže tedy náš výsledek je: x = 10001

Druhý příklad:

$8_{10} = x_2$

8 : 2 = 4 zb. 0
4 : 2 = 2 zb. 0
2 : 2 = 1 zb. 0
1 : 2 = 0 zb. 1

Tedy výsledek: x = 1000

Příklad převodu pomocí vah

Každá cifra v čísle má nějakou váhu. Známe to z desítkové soustavy jako jednotky, desítky, stovky atd. Tyto váhy se dají vyjádřit ve tvaru exponentů, tedy v naší desítkové jsou jednotky $*10^0$, desítky $*10^1$, stovky $*10^2$ atd. Je mi jasné, že spoustu z vás jsem tímto lámaným souvětím zaskočil, ale ukážeme si to na příkladu. Číslo 172 jde zapsat jako: $1*10^2+7*10^1+2*10^0$.

Zmíněného rozkladu lze užít pro převod mezi číselnými soustavami. Základ soustavy je číslo, které je umocňováno. Pro dvojkovou soustavu uvádím v následující tabulce hodnoty několika základních mocnin. Převáděnou hodnotu odečteme od hodnoty nejvyšší mocniny, která se do převáděné vejde, a výsledek zapíšeme jako zbytek. Do zbytku hledáme opět nejvyšší možnou mocninu a postup opakujeme, dokud zbytek není nula. Ty pozice, kde chybí mocnina dvou (je nulová) ve výsledném kódu nahradíme nulou a nesmíme zapomenout na dvě na nultou. Pro kontrolu si za jedničky v kódu dosadíme hodnoty příslušných mocnin a sečteme. Náš součet se musí rovnat číslu, které jsme převedli.

Tabulka hodnot mocnin dvojky

Mocnina	Hodnota	Mocnina	Hodnota	Mocnina	Hodnota
2^0	1	2^6	64	2^{12}	4096
2^1	2	2^7	128	2^{13}	8192
2^2	4	2^8	256	2^{14}	16384
2^3	8	2^9	512	2^{15}	32768
2^4	16	2^{10}	1024	2^{16}	65536
2^5	32	2^{11}	2048	2^{17}	131072

Příklad váhového převodu

$24_{10} = x_2$
$24 = 2^4$ zb. 8
$8 = 2^3$ zb. 0
$1*2^4+1*2^3+0*2^2+0*2^1+0*2^0$
Výsledek: 11000

Příklad váhového převodu 2

$17_{10} = x_2$
$17 = 2^4$ zb.1
$1 = 2^0$ zb. 0
$1*2^4+0*2^3+0*2^2+0*2^1+1*2^0$
Výsledek: 10001

1.5.3.2. Hexadecimální soustava

Jak jsem již nakousl v této kapitole o číselných soustavách v IT, tak operační systém a některé aplikace užívají hexadecimální, neboli šestnáctkovou soustavu. Takto se zapisují například adresy v paměti nebo se kódují znaky. Hexadecimální kód je pak převeden BIOSem do binára.

Hexadecimální cifra obsahuje čísla 0 až 15. Jak tedy dvouciferné hodnoty zakódovat jednociferně? Využívá se naprosto jednoduchého principu: čísla 0 až 9 se píší stejně jako v desítkové soustavě, číslo 10 se značí jako A, 11 je B a takto to jde až do 15, což je F.

Nemusíte se bát, že by bylo potřeba nějak složitě dělit 16 a přepisovat výsledky, nebo snad počítat mocniny 16. Převod do hexadecimální soustavy je velmi elegantní a jednoduchý, ač by pomocí váhové metody byl realizovatelný. Základ soustavy, tedy 16, je roven 2^4, tedy 1111, ale protože kódujeme i nulu, tak binárnímu číslu 1111 ve strojovém kódu ve skutečnosti odpovídá číslo 15, což je v hexadecimálu F. Už vidíte tu spikleneckou jednoduchou fintu? Celé kouzlo převodu do hexadecimální soustavy spočívá v převodu do binární soustavy. Binární číslo rozdělíme na „bloky“ vždy po 4 cifrách. Dělení na bloky začneme odzadu, protože pokud nebude číslo vyplňovat všechny bloky zcela, tak poslední blok doplníme nulami před číslo. Z takto sestavených bloků již jednoduše sestavíme jednotlivé cifry hexadecimálního čísla. Cifry pak stavíme v číslo z hora dolů, pokud bloky vyhodnocujeme pod sebou. Pro určení cifry si buď obsah bloku převedeme do desítkové soustavy, nebo užijeme následující tabulku.

Tabulka převodů čísel

decimál	binár	hexadecimál
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Příklad převodu

$19_{10} = x_{16}$	10011 = **00010011**	$0001_2 = 1_{10} = 1_{16}$
19:2 = 9 zb. 1	Po rozdělení čísla na čtveřice	$0011_2 = 3_{10} = 3_{16}$
9:2 = 4 zb. 1	odzadu se zjistilo, že se musí	
4:2 = 2 zb. 0	doplnit dopředné nuly.	$1*16^1+3*16^0 = 16+3*1=19$
2:2 = 1 zb. 0	Blok 1 je zelený	
1:2 = 0 zb. 1	Blok 2 je žlutý	Výsledek: $19_{10} = 13_{16}$

1.5.4. Shrnutí kapitoly

Myslím, že jsem snad nezapomněl osvětlit nic, co by mezi námi tvořilo komunikační bariéru. Co se týká HW, tak většinu pojmů znáte z fyziky, nebo se s nimi ve fyzice setkáte, pokud jde o SW, tak je toho hodě odvoditelné z faktu, že počítače pracují pod taktovkou binární soustavy, tedy je napětí nebo není. Jako každé odvětví i informatika má svojí terminologii a bohužel tu malou trošku memorování vyžaduje také. Jistě jste se někde setkali i se spoustou dalších pojmů, které vysvětlím postupně v průběhu našeho poznávání světa počítačů. Jen snad ještě jedna malá poznámečka, pokud někde uvidíte například: kbps, tak je to to samé, jako kb/s, kdy písmeno p nahrazuje lomítko a je zkratkou anglického, respektive latinského, slova per, což znamená za.

1.6 Historie

Jako každé řemeslo, i IT má svojí historii. Ta se sice nepodobá nijak známému vtipu o nejstarším řemesle, kterým je zjevně elektrikář, protože když pánbůh řekl budiž světlo, tak kabely již museli být nataženy. Člověk je již od přírody tvor líný a jak má možnost si nějak ušetřit práci, ostatně i já to teď dělám, tak to udělá. Jedna z cest, která napadla spoustu významných lidí, bylo dát tu práci stroji. Asi si říkáte, kdo asi tak může být nejlínější? Odpověď asi mnohé překvapí, ale nikoho nezaskočí, jsou to matematici a hvězdáři, těm jako prvním sloužili počítače, ale především jejich mechaničtí předchůdci. Pojďme si tedy něco málo říct o těchto kouzelných skříňkách.

1.6.1. Předchůdci

První zařízení, která se později vyvinula v dnešní počítače, byla velmi jednoduchá a byla založena na mechanických principech. Vývoj probíhal až do poloviny 20. století ve dvou větvích: analogové počítače a číslicové počítače. Analogové počítače modelovaly problém převedením do jeho mechanické nebo elektrické analogie, číslicové pak jeho převedením na číselné hodnoty a jejich numerickým zpracováním. S vývojem číslicové techniky během 2. poloviny 20. století, ale analogové počítače prakticky zanikly.

1.6.1.1. Abakus

Abakus vznikl přibližně před 5000 lety. Je prvním známým nástrojem, který usnadňoval počítání s čísly. Původně šlo jen o zaprášený kámen (starohebrejské slovo abaq znamená „prach“), který se používal v Babylonii již od poloviny třetího tisíciletí př. n. l. Nejstarším dochovaným exemplářem je salamiská tabulka, která pochází zhruba z roku 300 př. n. l., avšak historik Hérodotos popsal příklady pro tabulku tohoto typu již o více než století dříve. Ve starověkém Řecku a Římě se používala dřevěná nebo hliněná destička, do které se vkládaly kamínky („calculli“) – odtud název kalkulačka.

V překladu do češtiny je abakus počítadlo takové jak jej znají děti v prvním ročníku



základní školy.

1.6.1.2. Mechanické kalkulátory

První mechanické zařízení, které patrně sloužilo k výpočtům, je Mechanismus z Antikythéry (podle řeckého ostrova Antikythera, kde byl objeven vrak lodi), vzniklý někdy mezi roky 150 a 100 př. n. l. Podle dnešních poznatků sloužil k výpočtu kalendáře resp. polohy Slunce, Měsíce a planet. Mechanizmus se skládal z více než třiceti ozubených koleček seřazených v rovnostranných trojúhelnících a na svou dobu byl skutečně miniaturní. Zajímavé je, že mechanizmus je nejspíš založen na heliocentrickém principu (tedy Země obíhá kolem Slunce), kterému se v Evropě začalo dařit až o osmnáct století později.

Další známý mechanický kalkulátor sestavil roku 1623 Wilhelm Schickard z ozubených koleček z hodinových strojků (proto bývá nazýván „počítací hodiny“). Uměl sčítat a odčítat šesticiferná čísla a měl být prakticky použit Johannem Keplerem při astronomických výpočtech.

Známý francouzský matematik, fyzik a teolog Blaise Pascal vyrobil ve svých 19 letech v roce 1642 počítací stroj, který uměl sčítat a odčítat (Pascaline). Gottfried Wilhelm Leibniz ho následoval v roce 1671 a kolem roku 1820 vytvořil Charles Xavier Thomas první úspěšný sériově vyráběný kalkulátor – Thomasův Arithmometr, schopný sčítat, odčítat, násobit a dělit. Ten byl převážně založen na Leibnizově přístroji. Technologie mechanických počítacích strojů se udržela až do 70. let 20. století.



Mechanické kalkulátory pracovaly v desítkové soustavě (včetně ENIACu), která je sice pro člověka obvyklá, je však implementačně složitější než v současnosti používaná dvojková soustava, kterou popsal Leibniz.

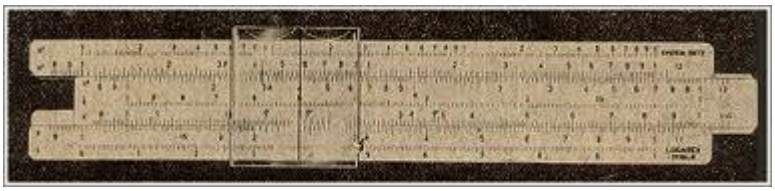
V roce 1725 použil Basile Bouchon děrovaný papír pro řízení tkalcovského stavu (první CNC stroj). O rok později v roce 1726 vylepšil Jean-Baptiste Falcon funkci spojením jednotlivých papírových karet, čím zjednodušil úpravy a změny programu. V roce 1801 použil francouzský vynálezce Joseph Marie Jacquard v tkalcovském stavu děrné štítky, které bylo možné vyměnit beze změny v mechanice samotného stavu. Tento okamžik je považován za milník v programovatelnosti strojů.

Nápad použít děrné štítky k programování mechanického kalkulátoru uplatnil v roce 1835 Charles Babbage. Děrný štítek obsahoval znaky ve formě kombinace dírek a umožňoval obsah opakovaně použít. K uchovávání dat a jejich pozdějšímu dalšímu využití použil poprvé děrné štítky Herman Hollerith, který se svou metodou vyhrál v roce 1890 v USA konkurz na sčítání lidu (to předchozí totiž trvalo plných 7 let). Jeho firma se později stala základem slavné počítačové společnosti IBM a tento charakter zpracování dat se udržel dalších 100 let. Pro analýzu a další zpracování dat na děrných štítcích byly vyvíjeny specializované stroje – děrovače, tabelátory a třídiče.

Technologie děrných štítků o něco později umožnila návrhy prvních programovatelných strojů. Tehdejší metoda programování spočívala v tom, že programátor předal své děrné štítky ke zpracování do výpočetního střediska a čekal, jestli získá výsledky nebo výpis chybových hlášení. Pokud došlo k chybě, musel zpětně zapracovat opravu do svého programu, který mezi tím již dále vylepšil. Poté znovu odeslal štítky do výpočetního střediska a celý cyklus se opakoval.

1.6.1.3. Logaritmické pravítko

Roku 1614 objevil John Napier novou matematickou metodu, umožňující realizovat násobení a dělení pomocí sčítání a odčítání s využitím logaritmů. Následně byly v Anglii sestaveny první logaritmické tabulky. Po nich následovalo logaritmické pravítko, kde byla reálná čísla reprezentována vzdáleností na ose. Logaritmické pravítko bylo prakticky beze změny používáno dalších 200 let, dokonce bylo používáno i k provádění výpočtů v rámci programu Apollo.



1.6.2. Nultá generace

Nultá počítačová generace v sobě zahrnuje počítače s reléovými obvody a její vznik spadá na pomezí 30. a 40. let minulého století. Jejími typickými představiteli byly Harvard Mark I, Harvard Mark II nebo stroje německého inženýra Konráda Zuseho Z2 a Z3. U nás patřil k nulté počítačové generaci první čs. počítač SAPO.

Počítače této etapy byly vybaveny reléovými registry a magnetickými bubnovými operačními pamětmi. Dosahovaly průměrných operačních rychlostí několika operací za sekundu a jako přídavných periferních zařízení používaly děrnoštítkové a děrnopáskové jednotky, elektrické psací stroje nebo dálnopisy. Uplatňovaly se zejména pro vědeckotechnické výpočty a bývaly většinou umístěny na vědeckých nebo univerzitních pracovištích.

1.6.2.1. Z1

První, komu se podařilo sestrojit fungující počítací stroj, byl německý inženýr Konrad Zuse. V roce 1934 začal pracovat na konstrukci mechanické výpočetní pomůcky a po řadě různých zdokonalení dokončil v roce 1936 základní návrh stroje pracujícího v dvojkové soustavě s aritmetikou v plovoucí čárce a programem na děrné pásce (jako nosič byl použit kinofilm). Neznalost prací Babbageho a jeho následovníků však měla za následek, že Zuse do svého projektu nezahrnul podmíněné skoky. Přes tento nedostatek však můžeme tvrdit, že roku 1938 spatřil světlo světa první počítač nazvaný Z1. Byl ještě elektromechanický s kolíčkovou pamětí na 16 čísel a byl velmi poruchový, pro praktické použití nevhodný.



1.6.2.2. Z2 a Z3

Počítač Z2 již obsahoval asi 200 relé a byl také dílem Zuse. Paměť však byla stále ještě mechanická, převzatá ze Z1. Potom se Konrád Zuse spojil s Helmutem Schreyrem a společně se pustili do vývoje ještě výkonnějšího počítače Z3, který dokončil v roce 1941. Tento první prakticky použitelný počítač na světě obsahoval 2600 elektromagnetických relé a byl užíván též k výpočtům charakteristik balistických raket V2. Pracoval ve dvojkové soustavě a prováděl až 50 aritmetických operací s čísly v pohyblivé řádové čárce za minutu (ani ne jedna za sekundu). Paměť byla na tehdejší dobu velká, 64 čísel po 22 bitech. Údaje se ručně zadávaly pomocí klávesnice. Počítač byl v roce 1944 zničen při náletu.

1.6.2.3. Mark I



V letech 1939–1944 pracoval ve Spojených státech na podobném projektu Howard Hathaway Aiken. Oficiálně se projekt jmenoval Automatic Sequence Contolled Calculator (ASCC, česky automatický sekvenčně řízený počítač), neformálně se nazýval Howard Mark I. Celý projekt financovala firma IBM (International Business Machines), která vznikla sloučením bývalé Holleritovy společnosti Tabulating Machine Company s několika dalšími a zabývala se do té doby zejména výrobou děrnoštítkových strojů.

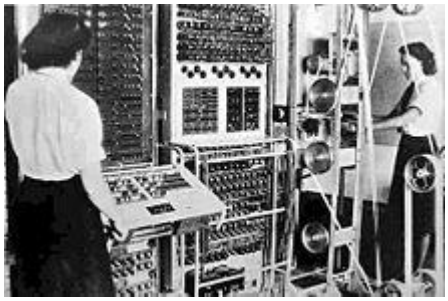
Aikenův projekt počítacího stroje chápala jako demonstraci svých technických možností. Byl to její první vstup do světa výpočetní techniky. Později byl ve světě znám spíše pod názvem Mark I. Počítač byl dodán v roce 1944 Harvardově univerzitě v Cambridge. Patnáct metrů dlouhé monstrum bylo poháněno elektromotorem o výkonu 3,7 kW, který byl napojen na dlouhou hřídel zprostředkovávající pohon jednotlivých částí počítače, který obsahoval 765

000 elektromechanických prvků. Program nesla děrná páska, jejíchž 24 stop bylo rozděleno do tří skupin po osmi (2 adresy + kód operace).

Počítač pracoval v desítkové soustavě s pevnou čárkou. Paměť měla dvě části - statickou, do které bylo možno před zahájením výpočtu vložit až 60 dvacetičtyřmístných čísel, a dynamickou (operační) paměť tvořenou elektromechanicky ovládanými kolečky. Do této paměti si mohl počítač zaznamenat a zpětně přečíst dalších 72 čísel (23 místných). Zároveň zde probíhaly aritmetické operace sčítání a odčítání.

Mark I dovedl sečíst dvě čísla za 0,3 s, vynásobit je za 6 s a vypočítat např. hodnotu sinus daného úhlu během jedné minuty. Americké námořnictvo ho využívalo k výpočtu balistických tabulek.

1.6.2.4. Mark II



Po úspěchu počítače Mark I začal Aiken pracovat na počítači Mark II. Toto zařízení bylo již čistě reléové. Aritmetika pracovala v plovoucí čárce s desítkovými číslicemi, které byly dvojkově kódovány pomocí čtyř relé. Operační paměť počítače mohla pojmout až 100 čísel s deseti platnými číslicemi. Sčítání již trvalo pouze 0,125 s a násobení průměrně 0,25 s. Celý počítač obsahoval přibližně 13 000 relé. Počítač začal pracovat v roce 1947 a byl předán americkému námořnictvu.

1.6.2.5. SAPO

Prvním počítačem vyrobeným v Československu byl SAPO (Samočinný Počítač), který byl uveden do provozu v roce 1957. Obsahoval 7000 relé a 400 elektronek. Měl magnetickou bubnovou paměť o kapacitě 1024 dvaatřicetibitových slov. Pracoval ve dvojkové soustavě s pohyblivou řádovou čárkou.

Tento počítač měl dvě zvláštnosti: první bylo, že byl pětiadresový neboli součástí každé instrukce bylo 5 adres (2 operandy, výsledek a adresy skoků v případě kladného a záporného výsledku) a druhou bylo, že se vlastně jednalo o tři shodné procesory, které pracovaly paralelně. Výsledek každé operace z jednotlivých procesorů se mezi sebou porovnal a o výsledku se rozhodovalo hlasováním. Pokud byl shodný alespoň ve dvou případech, byl považován za správný. Pokud se ve všech třech případech lišil, operace se opakovala.

Počítač SAPO byl zkonstruován prof. A. Svobodou, Dr. Oblonským a jejich spolupracovníky v Ústavu matematických strojů (pozdější Výzkumný ústav matematických strojů) a byl instalován v budově ústavu na Loretánském náměstí. Tři roky po svém spuštění, v roce 1960, počítač SAPO shořel. Z jiskřících releových kontaktů se vzňala loužička oleje, kterým se relé promazávala.

1.6.3. První generace

První počítačová generace zahrnuje počítače s **elektronkovými obvody**. Tyto počítače byly vyráběny **od poloviny 40. let až do konce 50. let** a měly registry s elektronkovými klopnými obvody. Jejími typickými představiteli byly **ENIAC**, **EDVAC**, **SSEC** nebo sovětský **URAL 1**. V Československu patřil k této generaci např. **EPOS 1**.

Počítače první generace disponovaly obrazovkovými nebo později také feritovými pamětmi a vstup a výstup byl kromě děrných štítků a děrné pásky realizován elektrickým psacím

strojem a řádkovou tiskárnou. Pro záznam většího množství dat se používaly magnetické páskové a magnetické bubnové paměti. Programování bylo realizováno pomocí strojového kódu nebo v jednoduchých strojově závislých jazycích. Dosahovaly operačních rychlostí několik set až tisíc operací za sekundu.

Elektronkové počítače bývaly většinou instalovány ve výpočetních střediscích a užívalo se jich pro vědeckotechnické výpočty nebo pro hromadné zpracování dat. Ve větším množství byly tyto počítače též instalovány v některých státních nebo soukromých podnicích v USA. Díky omezené životnosti elektronek trpěly však přístroje první generace značnou poruchovostí a nespolehlivostí, a proto byl jejich provoz často značně neekonomický.

1.6.3.1. ENIAC a MANIAC

V roce 1944 byl na univerzitě v Pensylvánii uveden do provozu elektronkový počítač ENIAC, který byl prvním počítačem, který pracoval podobně jako dnešní počítače. Na rozdíl od Z3 umožňoval vytvoření smyčky i podmíněné skoky a byl Turingovsky úplný. Prováděl až 5000 součtů za sekundu, ale byl energeticky velmi náročný, poruchový a jeho provoz byl drahý. Jeho provoz byl ukončen v roce 1955.

ENIAC byl přímou inspirací pro počítač MANIAC (Mathematical Analyser Numerical Integrator And Computer), který byl sestaven roku 1945 a uveden do provozu John von Neumannem. V laboratořích Los Alamos National Laboratory byl použit k matematickým výpočtům popisujícím fyzikální děje a byl využit i k vývoji jaderné bomby.

1.6.4. Druhá generace

Druhou počítačovou generaci tvořily počítače s **tranzistorovými obvody**. Tyto stroje vznikaly **od druhé poloviny 50. let až do poloviny 60. let** a byly vybaveny registry s tranzistorovými klopnými obvody a feritovými pamětmi. Vnější paměti a periferní jednotky měly již vlastní řídící jednotku pro připojení k jednotce základní a pro řízení přenosu dat. Operační rychlost počítačů s tranzistorovými obvody se pohybovala v několika tisících operacích za sekundu.

Typickými modely druhé generace byly počítače **IBM 1401**, **IBM 7090**, **ZUSE 23** nebo sovětský **MINSK 22**. V Československu byl prvním tranzistorovým počítačem **EPOS 2**, který se vyráběl pod označením **ZPA 601**.

Počítače druhé generace nalezly uplatnění při hromadném zpracování dat, vědeckotechnických výpočtech a dokonce při řízení technologických procesů. Během této generace vznikly první vyšší programovací jazyky FORTRAN, ALGOL, COBOL aj. Došlo také k zavádění systémů pro dálkový přenos dat a terminálů, které pracovaly v systémech rezervujících letenky, jízdenky a hotelová místa. Vznikly nové druhy periferních jednotek, a to včetně speciálních jednotek pro strojové čtení písma a také hlasové vstupní a výstupní jednotky. Objevily se rovněž první řídící počítače pro číslicové řízení a číslicové diferenciální analyzátory

1.6.5. Třetí generace

Třetí počítačová generace zahrnuje počítače s **monolitickými** a **hybridními integrovanými obvody**. Uspořádání polovodičových prvků na jediném čipu tvořily tři stupně integrace **SSI** (malá integrace obvodů), **MSI** (střední integrace obvodů) a **LSI** (velká integrace obvodů). Počítače třetí generace vznikaly **od první poloviny 60. let do konce 70. let**. Tato generace není definována již jen z hlediska součástkové základny, ale hodnotí se také vzájemná vazba hardwarové a softwarové stránky. Jejími typickými představiteli byly počítačové řady **IBM 360** nebo **Siemens 4004**. V Československu a ostatních zemích zastoupených v Jednotném systému elektronických počítačů to byla řada **JSEP-R1**.

Tato etapa počítačového vývoje se vyznačuje vytvářením řady počítačů, které jsou na různých úrovních vzájemně kompatibilní. Počítače třetí generace charakterizovalo stavebnicové řešením všech prvků počítačového systému, multiprogramování, využití interních a externích databank a zvětšení programového vybavení, které zahrnovalo operační systémy a knihovny uživatelských programů. Vnější paměti bývaly magneticko-diskové a magneticko-páskové. Jako operační vnitřní paměti se využívaly feritové paměti, paměti s tenkými magnetickými vrstvami a polovodičové paměti. Operační rychlost se pohybovala zpravidla v několika desítkách až stech tisících operací za sekundu. V období třetí generace se objevují též **mikropočítače** a **minipočítače** např. řady **Systému malých elektronických počítačů** (SMEP).

1.6.6. Třiapůltá generace

Jako přechod mezi třetí a čtvrtou počítačové generací se vyskytuje v některých publikacích termín **tzv. třiapůltá generace**. Jedná se o zdokonalenou řadu počítačů třetí generace, která se vyznačovala vyšší hustou prvků v integrovaných obvodech, zlepšenými provozními vlastnostmi a vyšší operační rychlostí, která se pohybovala řádově až v několikset tisících operací za sekundu.

Počítače třiapůlté generace mají polovodičové paměti zabezpečené proti chybám samoopravným kódem a virtuální dynamickou paměť. Programové vybavení se specializovalo pro dálkový přenos a zpracování dat. Představitelem třiapůlté generace byla řada počítačů **IBM 370**, dále pak druhá řada počítačů **JSEP-R2**, zastoupená např. **EC 1025/1026** nebo **minipočítač SM 52/11** druhé vývojové etapy SMEP.

1.6.7. Čtvrtá generace

Čtvrtá počítačová generace v sobě zahrnuje počítače vytvoření s využitím **VSLI** neboli **velmi velké integrace obvodů** a vyznačuje se pokročilou miniaturizací, zvyšováním výkonů a paměťových kapacit. Celý řídící procesor s jeho lokální pamětí je uložen v jediném malém integrovaném obvodě. Počítače čtvrté generace se objevují již **od konce 70. let** a vývoj této generace není ukončen ani v současnosti, neboť se prakticky nevyrábějí jiné počítače, které by tomuto označení nepodléhaly. K typickým představitelům počáteční fáze této generace náležely v Československu např. počítače **EC 1027** nebo **SM 52/12** a ve světě **IBM 308X**.

1.7. Ženy v Historii IT

I když většina z vás si řekne, „žena a IT? To má být asi nějaký vtip“. Nejsem toho názoru, že by něžné pohlaví nemělo nadání pro počítače, jen ženy jsou od přírody nastaveny jinak nežli muži, takže se jejich talentu musí pomoci. Z vlastní zkušenosti mohu říct, že ženy to s počítači umí, a když se v nich ten skrytý talent probudí, tak mají lepší výsledky nežli jejich mužští kolegové. Jedno staré přísloví říká, „za vším hledej ženu“, tedy i za počítači je nutno vidět ženu, nejen jako uživatele, programátorku, správce, ale také jako zdroj nápadů a morální oporu průkopníků oboru. Pojďme se tedy podívat, jak si v historii IT stály ženy samotné.

Za první historicky první programátorku (a programátora) je považována komtesa Ada Lovelace, dcera slavného anglického básníka Lorda Byrona, která se podílela po boku skotského matematika Charlese Babbage na vývoji předchůdce dnešních počítačů. Adina matka z obavy, že její dcera zdědila divokou povahu svého otce a ve snaze potlačit jakékoli básnické sklony, nechala Adu vzdělávat v matematice, která ji samotnou zajímala. Ada však měla štěstí, když v 18 letech navštívila přednášku anglického matematika Charlese Babbage, na které představil stroj nazvaný **'Difference Engine'**, který prováděl matematické výpočty. V roce 1844 se nabídla, že přeloží pro Babbage článek o 'Difference Engine' napsaný v italštině. Její práce, ve které naprosto dokonale popsala fungování tohoto stroje, udělala na Babbage dojem, a proto ji

požádal, aby k překladu připojila svůj vlastní komentář. Ada této nabídky využila a její poznámky, které byly třikrát delší než samotný článek, jí zajistily spolupráci s tímto slavným matematikem. Ada začala s Babbagem pracovat na vývoji pokročilejšího počítacího stroje nazvaného 'Analytical Engine', který je považován za předchůdce moderních počítačů. Tento stroj měl provádět výpočty za pomoci děrovacích štítků, které obsahovaly instrukce, paměťové jednotky a dalších komponentů, které obsahují dnešní počítač. V mnoha směrech si Ada uvědomovala potenciál "počítacího stroje" více než jeho duchovní otec. Její poznámky obsahují instrukce, podle kterých měl stroj řešit složitější matematické operace. Ada je autorkou programu, podle kterého měl stroj vypočítat tzv. Bernoulliho řadu čísel. Za tento významný počin bývá považována za prvního počítačového programátora vůbec.



V důsledku nedostatku pracovních sil v době druhé světové války ženy nahrazovaly muže, kteří odcházeli do armády, a výjimkou nebyl ani projekt prvního počítače ENIAC, na kterém se ženy významně podílely. V novinách a časopisech otiskla americká armáda inzerát s titulkem "Hledají se ženy s vysokoškolským vzděláním v matematice" ("Wanted: Women With Degrees in Mathematics"). V inzerátu bylo dále uvedeno, že armáda ženám nabízí vědecké a inženýrské pozice ve Filadelfii, které dříve zastávali muži. Přibližně stovka absolventek matematických oborů bylo najato, aby prováděly složité a monotónní výpočty balistických tabulek pomocí stolních kalkulaček. Šest z nich (Kay McNulty, Betty Jennings, Betty Snyder, Marlyn Wescoff, Fran Bilas a Ruth Lichterman) bylo později vybráno, aby vytvářely programy balistických výpočtů, neboť v této době bylo programování považováno za ženskou práci (na rozdíl od konstruování hardwaru), která nevyžadovala vlastní invenci - programátorky "pouze" převáděly výpočty do algoritmů, podle nichž byl výpočty schopen zpracovat ENIAC. Časem však programátorky (nazývané operátorky) zjistily, že bez znalosti hardwaru programovat nemohou, a proto se naučily i to, jak ENIAC funguje.

Nejslavnější z poválečných amerických programátorek byla Grace Hopper, která jako první žena získala doktorát z matematiky na Vassar College a v hodnosti admirála námořních reserv se podílela na vývoji počítače Harvard Mark I.

Do historie IT se výrazněji zapsala také Anita Borg, která byla ve Spojených státech jednou z mála žen s doktorským titulem v počítačových vědách (velká vzácnost i dnes a to nejen v USA), a která se s úspěchem prosadila ve vývojových týmech největších počítačových společností. Dvanáct let strávila ve výzkumné laboratoři společnosti Digital Equipment (dnešní Compaq), později pracovala jako inženýrka v laboratoři Network Systems v kalifornském Palo Altu na projektu MECCA Communications and Information Systems. V rámci tohoto projektu vyvinula metodu na generování celé adresní cesty, která se začala využívat pro analyzování a návrh vysokorychlostních paměťových systémů. Ve Spojených státech začala již v 80. letech 20. století usilovat o větší zapojení žen do IT. Na její počest Google uděluje od roku 2004 stipendium Anity Borg.

Další neméně významnou ženou v oboru byla Margaret Hamiltonová, která se podílela na vytvoření autopilota projektu Apollo. Její krédo bylo „Lidé jsou omylní a počítače s tím musí počítat“. Právě její zdrojový kód zachránil životy austronautům letícím na měsíc. Celý svůj život zasvětila propagaci a rozvoji „počítačové vědy“ a „očekávání neočekávatelného“. Její zásadní myšlenka, že lidé chybbují a je na to potřeba počítačové programy správně přippravit byla v její době novinkou, i když dnes to všichni professionání programátoři považují za naprostou samozřejmost. K programování se dostala jako skoro každý v její době přes matematiku, kterou vystudovala na University of Michigan, ale prvně se k programování dostala na MIT ve 24 letech. Práce na projektu SAGE ji vynesla prestiž a místo v NASA, kde byl smíšený tým programátorů a nikdo neřešil, jestli tu práci odvedl muž, či žena, důležité bylo v časové tísni vyřešit mnoho problémů naprosto precizně. Na obrázku níže je pak Margaret s vytištěnou verzí zdrojového kódu pro Apollo 11 v roce 1969.



Jednou z reprezentantek žen-inženýrek byla Doc. Ing. Zdena Rábová, CSc., která patřila k zakladatelům oboru v ČSR, Katedry samočinných počítačů na elektrotechnické fakultě VUT v Brně i samostatné Fakulty informačních technologií VUT. Byla zakladatelkou brněnské školy modelování a patřila k předním odborníkům v této oblasti v celé republice.

1.8. Souhrnem…

Je mi jasné, že většinu z vás tato kapitola moc nezaujala, ale berme to tak, na něčem se rozepsat musím. Vím, že většina z vás zde hledá hlavně odpovědi na praktické otázky používání počítačů, ať je to výměna komponentu počítače, editace textu ve Wordu, tvorba prezentací, zpracování obrazu, práce s webem a jeho tvorba či programování, ale i na tato témata dojde v dalších kapitolách.

Co jsme se dověděli v této kapitole? Nastínili jsme si odpověď na otázku, co je to počítač, vysvětlili jsme si základní hantýrku, abychom si v těch dalších kapitolách rozuměli, rozluštili jsme jednotky, jejich označení a převody. Řekli jsme si z čeho se počítač skládá a jak si chránit zdraví při práci s ním, protože počítač nám umí hodně dát, ale i hodně vzít. Rozluštili jsme základní matematickou podstatu fungování počítače a koukli se trošku do historie, kterou jsme zběžně proběhli od „dinosaurů“ až po dnešek. Samozřejmě jsme neopomněli vzdát čest něžné polovině populace, protože bez nich by tento obor nemohl vzniknout.

Jen jednu malou poznámečku, v některé počítačové literatuře, především ze začátku 90. let, se můžeme setkat i s pojmem pátá generace, což by měly být plně autonomní počítače pracující s umělou inteligencí a měli by být schopny samy sestavovat algoritmy řešení jednotlivých problémů. Tak daleko nejsme i přes práci spousty lidí, kteří se o to snaží, je to ještě hodně vzdálená budoucnost, ke které jak doufám, přispěje i někdo ze čtenářů.

2. Hardware

Řekněme pro jednoduchost začneme HW, přeci jen je to něco, co můžeme vzít do rukou, pořádně si to prohlédnout a zde platí ještě všechny zákony fyziky, tak jak je známe. Co budeme z fyziky potřebovat?, elektřinu a magnetismus, termiku, tlak, setrvačnost, vlnění a to je asi tak všechno. je mi jasné, že většina z vás si říká, jestli tuto knihu píše vůbec ITík, nebo fyzik, pravdu mají zastánci obou možností, kromě IT se zaobírám právě fyzikou. Ve většině knížek se komponenty řadí podle „důležitosti“, já to s vámi zkusím jinak, začneme tím, co je vidět a postupně budeme pronikat dovnitř, ono to nejde zcela ideálně, ale pokusíme se o to.

2.1. Skříň (Case)

Slečny by neměly jásat, sem žádné šaty ani boty neuloží. Jiný lidový název pro tuto součást počítače je bedna. Jde o tu část, kterou máte ve stole, na zemi, nebo v případě notebooku by se dalo říct, že je to ta část, na které je nahoře klávesnice. Notebooky ve skutečnosti tuto část neobsahují, ty mají tzv. šasí.

Jde o většinově kovový komponent, který chrání součásti počítače před vnějšími vlivy a urývá v sobě všechno kromě vyměnitelných jednotek a periferií (osvětlíme si dále, co to je). Tento prvek umožňuje fyzické rozložení komponent tak, aby zabraly málo místa a zároveň nebyl problém s jejich chlazením.

Skříň se vybírá podle několika požadavků. Hlavním požadavkem je velikost základové desky, dalším požadavkem je počet optických mechanik, přítomnost disketové mechaniky, umístění a počet čelních USB, instalovaný výkon a typ chlazení. U těch rozměrů a desingu skříně musíme brát ohled i na to, kde skříň stojí. S desingem záleží na vkusu každého člověka, ale pamatujme, že skříň musí být v první řadě funkční.

Skříně se dodávají v různých kategoriích: Big Tower, Midi Tower, Mini Tover, desktop (skříň určená na pracovní desku stolu, má jinou vnitřní architekturu a také jinak navrženy nosné prvky), mini ITX, Barebone (pro minipočítače určené umístěním na desku pracovního stolu), NUC a serverové skříně. Doma má většina z vás některou z verzí toweru. Velkou bolestí spousty nových skříní je umístění zdroje dole, což nedoporučuji s ohledem na snadné nasávání prachu. Skříně určené pro vodní či dusíkové chlazení (všeobecně tedy kapalinové chlazení) jsou specifické rozložením a úchytnými prvky pro chlazení.





A najdou se i tací kutilové, kteří si skříň ostaví sami, například z Lega, ale to příliš nedoporučuji. Málo lidí má k dispozici vhodné materiály a také technologii a na dostatečně přesné zpracování. Vymyslet skříň je velmi obtížné, protože každá deska je jiná, sice je ujednáno, kde mají být šrouby, ale i tak to dá dost práce, protože umístění šroubů se mění s typem desky.

2.2. Periferie

Toto slovo může pro větší část z vás být známo jako přídavné jméno ve spojení s viděním. V IT, ostatně jako ve spoustě případů, máme opět trošku jiný význam tohoto slova. Jako periferie označujeme zařízení, která jsou připojena k počítači, tedy přesněji kabelem nebo pomocí bezdrátové technologie jsou spojena se skříní. Tato zařízení jsou nám známější, nežli další komponenty, které budeme probírat později.

Jen pro připomenutí: Ovladač (driver) je program zajišťující komunikaci zařízení s OS a aplikacemi. Firmware je uložen uvnitř zařízení v paměti typu Flash. Tento software je nutno udržovat v aktuálním stavu (někteří výrobci nabízí službu Update, nebo pomocí webových stránek), aktualizace umožňují i zařízení, která nejsou učeny pro spojení s PC (přehrávače).

2.2.1. Vstupní zařízení

Je takové zařízení, které nám uživatelům umožňuje zadávat údaje ke zpracování do počítače. Bez těchto zařízení není možná činnost PC, protože nelze ani přihlásit uživatele OS při probuzení systému z režimu spánku nebo často i connected standby (Windows 8 a 8.1).

2.2.1.1. klávesnice

Jde o základní vstupní zařízení, které stačí k ovládání celého počítače na jakémkoli OS. Je složena z matice (soustavy) kláves, která je rozdělaná na tři podsestavy. Nejjednodušší představu o klávesnici si vytvoříme pomocí kostičkovaného papíru. V každá klávesa má své souřadnice (průsečík vodorovné a svislé čáry na čtverečkovaném papíru), když klávesu stiskneme, dojde k tomu, že se na dané souřadnici objeví napětí (pracovní napětí je 5 V), toto napětí registruje čip, který vytvoří binární kód a uloží jej do paměti, tzv. bufferu, odkud jsou tyto kódy odesílány (v pravidelných intervalech i při nečinnosti, pokud je buffer prázdný, tak odešle samé nuly) do počítače. Význam (tedy znak či funkci) přidělí dané klávese až ovladač, kterých u klávesnice, jako jediného zařízení, můžeme mí nainstalováno větší množství a přepínat mezi nimi. Ovladače jsou pro každý jazyk a některé, jako čeština, mají těch ovladačů více. Čeština má tři ovladače, české, české quarty a české programátorské.

Klávesnici můžeme mít virtuální (tvořenou programem a ovládanou pomocí myši či dotykové obrazovky), klasickou, nebo ergonomickou (dnes dost vzácné) a obě ještě mohou být standartní, nebo multimediální. Klávesnici můžeme k počítači připojit kabelem (nejčastěji USB (2.0, ale na standartu nezáleží), nebo PS2), nebo bezdrátově (technologie Bluetoot, kterou rozebereme později, nebo přijímač zapojený v USB).

Klávesnice je dělaná na části, Alfanumerickou (písmena abecedy), numerickou (číslice, tato část je vypínatelná), editační klávesy (šipky, delete, tab, end, back space), funkční klávesy (F1 až F12, kterým význam přidává aplikace (nepsaný zvyk je F1 = nápověda)), potvrzovací a stornovací klávesy (enter a esc), přepínací klávesy (NumLock, CapsLock, Insert, skollock), rozšiřující klávesy (Ctrl, Alt, Shift), které se kombinují v klávesové zkratky, a na konec klávesa pro rozbalení nabídky start (pro Windows 8 a 8.1 jsou dvě a slouží k přepnutí na úvodní obrazovku). Jednotlivé skupiny jsou barevně rozlišeny na následujícím obrázku.



Ještě by se slušelo, ukázat vám, jak vlastně jednotlivé typy klávesnice vypadají. Následující dvojice obrázků ukáže nejdříve standartní a následně ergonomickou klávesnici, která obsahuje i tauchpad.





Pro ty z vás, kteří by potřebovali nějaké další informace o klávesnici, jsou k dispozici stránky společnosti Microsoft na adrese: http://windows.microsoft.com/cs-cz/windows/using-keyboard#using-keyboard=windows-7

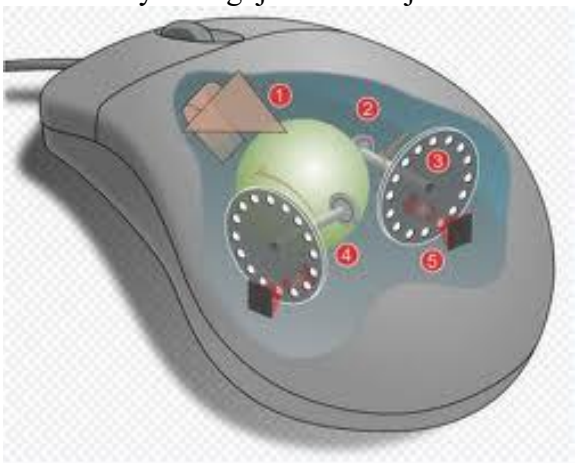
2.2.1.2. Myš

Každý asi někdy zažil, že se mu zasekla myš a počítač musel ovládat jen pomocí klávesnice. Asi se shodneme, že to není nemožné, ale v dnešní době, kdy jsme rozmazlení z grafických uživatelských rozhraní, to přeci jen moc pohodlné není. Ti zkušenější z vás mohou namítnout, že při využívání příkazového řádku se myš moc využít nedá (vlastně až od Win XP je možné myší provádět vkládání příkazů pomocí nabídky pravého tlačítka), ale ruku na srdce, kromě zprávy serverů instalovaných jako core se běžně počítač ovládá graficky.

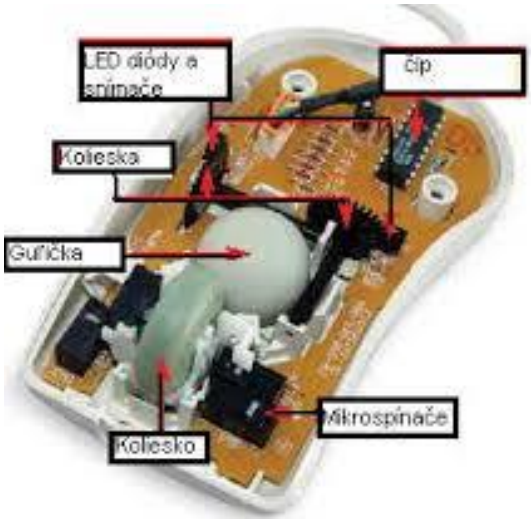
Všechnu tu senzaci jménem grafika odstartovala společnost Xerox, která to celé vymyslela a vymyslela myš dnešního typu (1963). Její vedení to tehdy zamítlo a celý výtvor předalo společnostem Microsoft a následně Apple. První počítače, které obsahovali grafické rozhraní a měly myš byla Mackintoshe společnosti Apple, u Microsoftu se myš na světlo světa dostala až s prvními verzemi Windows, v té době rozšířený MS DOS pracoval s příkazovým řádkem.

Myší máme několik druhů, kuličkové, moc je asi většina z vás nepamatuje, dnes se již více jak 10 let nevyrábějí a neprodávají, optické a laserové. Dále myši můžeme dělit dle účelu na standartní a herní a také dle připojení k počítači na bezdrátové (vlastní přijímače, nebo využití bluetoot) a s kabelem (USB 2.0 nebo PS2).

Kuličkové myši fungují na velice jednoduchém principu.



Kulička, která se otáčí v závislosti na pohybu myši po povrchu podložky, se dotýká uvnitř dvojice neprokluzujících „kartáčků“, které jsou uloženy kolmo (jeden snímá dopředný pohyb (je uložen příčně) a druhý snímá pohyb do stran (uložen podélně)). Tyto kartáčky pomocí hřídele otáčí clonou, která je děrovaná v pravidelných rozestupech. Z jedné strany na slonu svítí tenký paprsek světla z LED a druhé straně clony je světlo snímáno. Clona je neprůhledná, takže světlo projde jen dírami ve cloně. Snímané světlo je převedeno na napěťové impulsy (pokud světlo prochází, tak je přítomno napětí). Impulsy z obou clon jsou kódovány a předány počítači, která na jejich základě vyhodnotí pohyb myši, jeho směr a velikost.



Optická myš je trošku složitější, nicméně jednoduché zařízení. V myši je umístěna jakási malá kamera (CCD či CMOS prvek s maticí o velikosti několika desítek bodů), která snímá obraz v podobném rozlišení, jaké má například ikona programu. Rychlost jejího snímání je ovšem velmi vysoká, od jednoho tisíce snímků za sekundu se můžete dostat (u nejkvalitnějších myší) až k šesti tisícům vyhodnocených obrazů. Pokud myškou hýbneme, obraz se posune, čímž je možno zjistit, jakým směrem se pohybujeme, a jak rychle. K vyhodnocení pohybu se musí v myši nalézat relativně výkonný procesor. Aby tato kamerka něco zachytila, musí být plocha viditelná, tedy musí ji něco osvětlovat. K osvětlení plochy slouží jedna malá svítivá dioda (LED), jejíž světlo je namířeno pomocí hranolu/zrcátka na podložku.

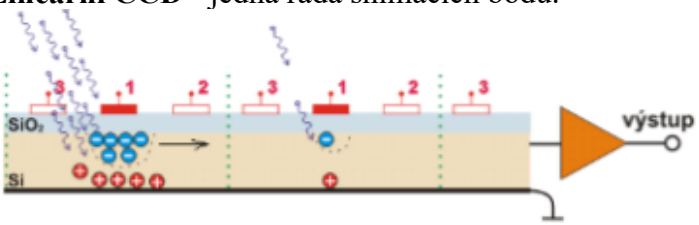
Laserová myš funguje na stejném principu jako optická, ale k osvícení plochy je užit laser a ke snímání speciální snímač. I vyhodnocovací procesor je dalece silnější, nežli u optické myši, protože dostává ke zpracování obraz několikanásobně kvalitnější.

2.2.1.2.1. Snímače CCD

Jsou to snímače obrazu ve fotoaparátech, kamerách, skenerech, čtečkách čárových kódů aj. Český název nemají. CCD -Charge-Coupled Device, zařízeni s vázanými náboji.

Princip: fotoelektrický jev.

Lineární CCD - jedna řada snímacích bodů.



Plošné CCD - matice bodů, snímáni po řadách.

Snímání barev:

- jednočipové - barevné filtry uspořádané do čtverce, zelená je použita dvakrát, neboť oko je na ni citlivější (Bayerův filtr); levnější zařízení;
- tříčipové - mají tři snímače, obraz je rozdělen zrcadly natřikrát a dopadá na snímače přes příslušné filtry; profesionální zařízení.

Velikost CCD: odvozuje se od dříve používaných snímacích elektronek; uvádí se v palcích, skutečná délka úhlopříčky čipu odpovídá zhruba 2/3 její délky v palcích, např. 1" - 16 mm. Množství pixelů je důležitý ukazatel kvality, ale zdaleka ne jediný - kvalita záleží i na dynamických vlastnostech čipu (rozsah odstínů), citlivosti, šumu aj.

Při dopadu světla na polovodič vzniká pár elektron - díra. Čím větší je intenzita světla, tím více je uvolněných nábojů. Záporné uvolněné náboje - elektrony - se posouvají pomocí kladných hodinových impulsů přiváděných postupně na elektrody 1-3 směrem k výstupnímu zesilovači a vyhodnocují. Kladné náboje se neutralizuji z uzemněné záporné elektrody.

2.2.1.3. Scanner

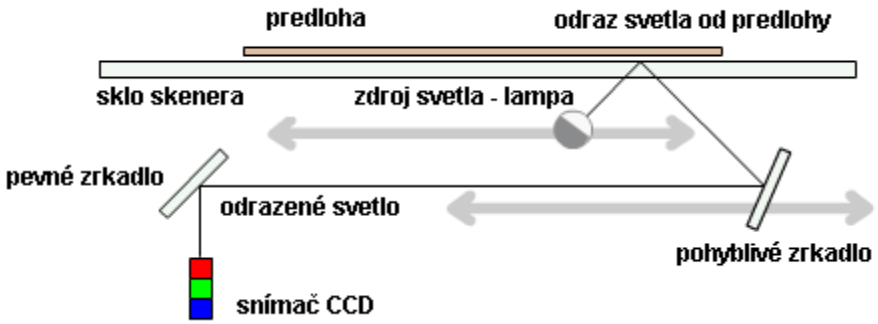
Všichni už jsme narazili na to, že jsme potřebovali do počítače dostat nějaký delší dokument a rozhodně se nám ho jako správným lenochům nechtělo opisovat a to nemluvím o dokumentech, které jsou orazítkované či ručně podepsané (třeba lékařské zprávy pro online žádost o pojistné plnění z úrazového pojištění). Všichni víme, že televize je dnes digitální, ale nějak se muselo povést dostat do digitální podoby staré filmy, digitalizují se třeba i sochy před jejich rekonstrukcí atd. toto jsou všechno případy, jistě byste sami přišli na spoustu dalších, které se řeší právě pomocí scanneru. Scannerů je nepřeberné množství, od těch, které si snadno spleteme s propisovací tužkou až po ty, které svojí konstrukcí připomínají spíše stany, ty umí pořizovat 3D záznamy objektu. Podle určení a výkonu se také mění připojení scanneru k počítači, je možné užít paralelní port, USB, Wi-Fi, SCSI.

2.2.1.3.1. Základní parametry

- Rychlost skenování (počet stran za minutu)
- Rozlišení (počet rozpoznaných bodů na délkovou jednotku (palec) – dpi (dots per inch), rozlišení skenerů bývá v násobcích (např. 600X1200 dpi))
- Barevná hloubka (kolik různých úrovní osvětlení ten který snímací prvek rozliší v oblasti červené, zelené a modré)
- Denzita – schopnost skeneru rozlišit různé jasy
- Typ propojení s PC (USB, SCSI, paralelní port, Wi-Fi)
- Typ Scanneru (bubnový, stolní, 3D atd.)

2.2.1.3.2. Princip

Tentokráte jsem se rozhodl pro jednoduchost vám nejprve ukázat nějaký principiální nákres.



Skenovaná předloha se osvětlí vhodným zdrojem světla, odražené (prošlé) světlo se zachytí světlo-citlivými prvky, které ho změní na elektrický signál. Tento signál je následně převeden na číselné hodnoty dále zpracovatelné v PC. Takto je postupně nasnímána celá předloha a vznikne tím matice čísel, kterou zpracuje daný program v počítači.

CCD snímání

Nejčastěji jsou snímače třířádkové, aby jedním průchodem snímací hlavy naskenovaly všechny barvy. Z toho vznikl pojem jednoprůchodové skenery. Předloha je osvětlována pomocí zářivky se studeným světlem, které se odráží od předlohy a přes zrcadla a filtry základních barev (červená, zelená, modrá) dopadá na vlastní snímač CCD. Filtry je nutno použít proto, že snímač měří intenzitu dopadajícího světla, nikoli jeho barvu.

Výhodou je možnost skenovat transparentní předlohy (negativy, diapozitivy), Některé skenery proto mají další zářivku přímo ve víku, nebo se k nim dodává adaptér.

Senzory rozlišují pouze hodnoty jasu, je tedy využito principů optiky.Tříprůchodové barevné skenery snímají předlohu jednou s osvětlením červeným, pak zelenám a modrým nebo používají barevné filtry, aby na senzory dopadla při každém průběhu jen potřebná část barevného spektra. Jednoprůchodové skenování při jednom průchodu jsou skenovány všechny tři barvy. Využívají jednu řadu CCD nebo tři řady CCD s rozšířením o zrcadla, hranoly a čočky projektující na různé řádky CCD. Využívá se také nahrazení hranolů a zrcadel fotodetektory.

CIS snímání (Contact Image Senzor)

Pro osvětlování používá miniaturní LED diody, opět v základních barvách - snímač si proto vystačí jen s jedním řádkem snímacích diod. Optická soustava je zde minimální - skenery postavené na této bázi mohou být mimořádně nízké. Detektory pro všechny tři barvy RGB jsou spojeny do vysoce integrovaného stavebního prvku, tím odpadají soustavy čoček a zrcadel pro separaci barev.

Výhody scannerů s CIS technologií

- levnější na výrobu,
- snímací hlava má menší rozměry,
- nízká spotřeba,
- napájení přímo z USB portu,
- jsou tenčí.

Nevýhody scannerů s CIS technologií

- nedosahuje kvalit CCD,
- nižší rozlišovací schopností na tmavých částech předloh
- neschopnost tzv. prostorového snímání.

2.2.1.3.3. Filmové scannery

Slouží ke snímání jednotlivých políček filmu, využívají tedy technologie CCD. Tyto scannery mají velmi vysoká rozlišení, mnohonásobě větší, nežli ty, které známe z multifunkčních tiskáren, můžeme se bavit o rozlišní kolem 30 000 dpi. Scannery pro digitalizaci filmových pásů bývají jednoprůchodové, pro digitalizaci záznamu z klasického fotoaparátu bývají i tříprůchodové (není nutná taková rychlost). Každý takovýto scanner je vybaven zařízením pro posuv filmu obdobně jako fotoaparát. Vhledem k účelu se užívají výhradně profesionálně a jejich cena se pohybuje v řádech desítek tisíc.



2.2.1.3.4. Bubnové

Předloha je na rotujícím válci a je snímána paprskem. Jejich nevýhodou je vysoká cena. Užívají se pro snímání velkých předloh např. plakátů, nebo tam, kde je potřeba vysoká kvalita výsledku (rozlišení může být až 30000 dpi). Jejich adekvátní náhradou jsou průtahové scannery, které je možné snadno zaměnit s plotrem.





2.2.1.3.5. Stolní

Toto je scanner, který známe z domova, bývá součástí multifunkčních tiskáren. Jeho rozlišení bývá nejčastěji 600 x 1200 dpi a rychlost kolem dvou stran za minutu. Nejčastější připojení je USB a Wi-Fi. Běžně se jedná o scannery na technologii CIS. Tradiční rozměr bývá o něco málo větší, nežli je formát papíru A4, ale vyskytují se jako součást kopírek i rozměry pro A3. většina těchto strojů disponuje technologií OCR (tedy, hlavně produktů společnosti hp, která využívá program I.R.S.I. OCR.). Doporučuji opravdu poctivě skenovat se zavřeným víkem, nebo pokud tam máte knihu a nelze to, tak alespoň zakrytými volnými místy, aby nešlo do scanneru okolní světlo.



2.2.1.3.6. Ruční

Využívají se hlavně k digitalizaci knih a listin v archivech. Jejich velkou výhodou je možnost skenovat velké plochy a atypické formáty, nicméně je s nimi relativně obtížné se naučit pracovat, chce to velký cit v rukou (jinak vám z textu vzniká čárový kód, mě se to osobně stalo několikrát, když jsem měl možnost s těmito zařízeními pracovat) a velkou trpělivost, protože tato zařízení jsou relativně pomalá. Co se týká kvality získaného výstupu, tak to hodně závisí na šikovnosti člověka a kvalitách konkrétního modelu, ale většinou jejich výstup má srovnatelnou kvalitu, nebo je kvalitnější nežli ze stolních scannerů.





Samostatnou kategorií ručních scannerů jsou tužkové scannery. Tyto scannery jsou schopny naskenovat jen jeden řádek textu a to většinou do velikosti písma 24. Jsou vybaveny technologií OCR (neumí dát výstup v podobě obrázku) a neumí rozlišit barvu textu. Většinou bývají součástí nějakého překladače či slovníku, která na displeji zobrazuje rovnou překlad skenovaného textu.



Poslední skupinu ručních scannerů známe asi všichni nejdůvěrněji, jde o čtečky čárových kódů, které jsou hojně využívány např. v obchodech.



2.2.1.3.7. 3D skenování

Tato technologie je vždy v popředí zájmu, je to taková Mekka digitalizace, ale má zatím spoustu zádrhelů, tím hlavním je cena, pořízení takovéhoto zařízení stojí stovky tisíc až miliony, dalším je časová náročnost práce i přípravy na práci. První 3D skenery byly sestaveny ze tří normálních skenerů, podle jejichž umístění bylo možné ze sejmutého snímku vytvořit 3D obraz. Dnes máme dvě základní možností snímání, buď pomocí jedné speciální hlavy, nebo pomocí více přesně rozmístěných snímačů.

Snímání více snímači, jako příklad si popíšeme snímání dvojicí snímačů a tato technologie dle popisu je určena pro malé předměty. Princip spočívá ve dvou snímačích, mezi kterými je otočná základna. Na tuto otočnou základnu postavíte objekt, který chcete snímat. Pomocí spousty obrázků ze všech stran vznikne za pomocí speciálního softwaru požadovaný 3D obraz.

Obecně jde o to, že více 2D obrazů, které získáme ze všech úhlů sestavíme v počítači na 3D objekt.



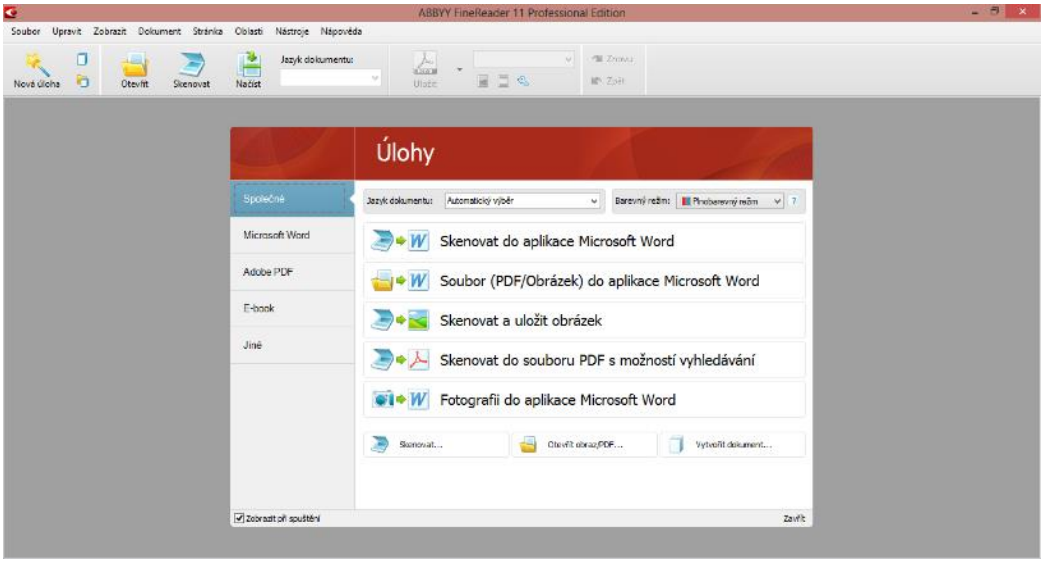
Snímání pomocí jednoho snímače: Jde o speciální typ laserového snímače, který je schopen pomocí laseru a odrazu světla zachytit 3D obraz. Tyto skenery / snímače jsou však určeny přímo pro konkrétní druh objektu, který jsou schopny 3D nasnímat.

Firma Cyberware vyrobila unikátní laserový skener lidské hlavy (Cyberscan), který je schopen nasnímat věrnou kopii lidské hlavy včetně povrchové textury. S podrobnými modely je možné dále pracovat jak ve formě polygonálních tak křivkových (splineových) objektů například v animačních SW typu Softimage 3D.



2.2.1.3.8. OCR

Určitě jste jako já narazili na to, že jste potřebovali nějakou citaci, nebo se vám líbil nějaký kousek knihy a chtěli jste si tento úryvek vložit do svého textu vytvářeného ve Wordu, nebo jste jej potřebovali upravit. Jako na potvoru vám scanner uložil obrázek nebo soubor pdf, ze kterého to nešlo zkopírovat. V takovéto situaci máte dvě možnosti, buď úryvek opíšete, což není tak zlé, pokud jsou to dva řádky, nebo použije OCR. OCR je optické rozpoznání znaku. Důležitou vlastností všech skenerů je možnost optického rozpoznávání textu, což je dáno použitým softwarovým vybavením pro daný skener. OCR umí v nasnímané obrázkové předloze rozpoznat text a převést jej do textového souboru (RTF, pdf, doc, docx). Pro OCR je potřeba mít zejména kvalitní předlohu. Ne všechny scannery mají ve svém SW vybavení kvalitní OCR, nebo OCR schopné pracovat s češtinou. Na trhu existuje řada OCR programů, z nichž mohu díky dobré osobní zkušenosti vyzdvihnout Abby Fine Reader 11, který umí češtinu (obsahuje i nástroje pro editaci textu a kontrolu pravopisu). Tento program umí převést soubory pdf na obrázky a následně na nich uplatnit technologii OCR. Umí zpracovat jak uložené obrázky, tak řídit scanner bez ohledu na typ a výrobce, jedinou podmínkou je, aby příslušný scanner měl správné ovladače.



2.2.1.4. Mikrofon

K počítači se mikrofon připojuje pomocí USB jako samostatná součást či jako součást webkamery nebo sluchátek. Pokud je mikrofon součástí sluchátek, tak často může být připojen pomocí audio jacku 3,5 mm.

Využívají pohybu membrány vyvolaného zvukovým vlněním k převodu na elektrické veličiny. Tlak zvukových vln působí:

- na jednu stranu membrány - tlakové mikrofony s všesměrovou charakteristikou
- na obě strany membrány - gradientní mikrofony se směrovou charakteristikou.

Hlavní parametry:

- citlivost C = U/p [mV. Pa] - poměr výstupního napětí a akustického tlaku
- frekvenční charakteristika - závislost výstupního napětí (úrovně) na frekvenci při konstantním akustickém tlaku
- vnitřní impedance Zj [Q]
- zatěžovací impedance Z_n [fi]
- směrová charakteristika - závislost citlivosti na úhlu v určité rovině při určité frekvenci - všesměrová - kulová, směrová - kardioidní čili ledvinová (potlačují zvuk zezadu), osmičková
- šum

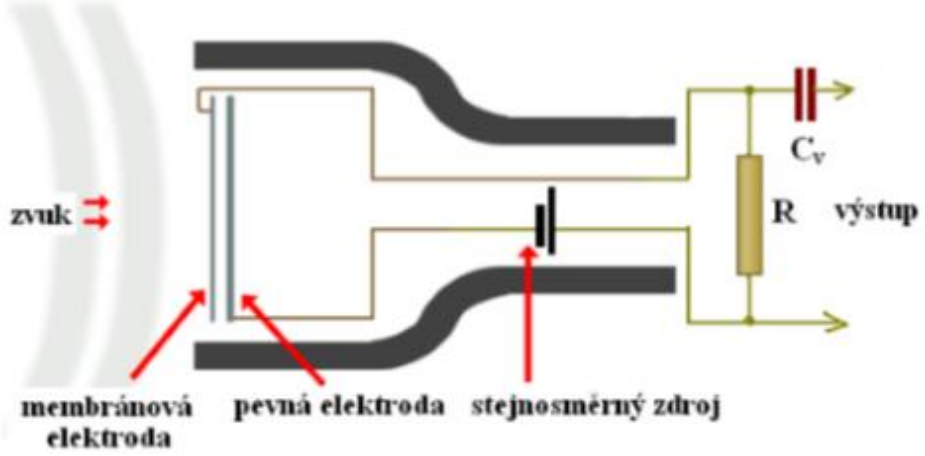
Krystalový (piezoelektrický) mikrofon: Princip: piezokrystal stlačovaný membránou akustickým vlněním generuje střídavé napětí. Vlastnosti: velká vnitřní impedance, vysoká citlivost, nízká kvalita; zastaralý.

2.2.1.4.1. Kondenzátorový mikrofon

Princip: základ tvoří kondenzátor, kde jedna elektroda je membránová; vlivem chvění membrány se mění kapacita kondenzátoru; stejnosměrný zdroj (připojený přes velký odpor R) udržuje konstantní náboj na kondenzátoru; podle vztahu Q = C.U se mění (střídavá složka) napětí a vyvádí se přes Cv. Alternativní snímáni: z LC rezonančního obvodu, jelikož rezonanční frekvence se mění s kapacitou.

Vlastnosti: vysoká kvalita a citlivost, vyrovnaná frekvenční charakteristika; použití pro profesionální snímání zvuku.

Tento typ je celkově v PC nejpoužívanější.



2.2.2. Výstupní zařízení

Jde o zařízení, s jejichž pomocí získáváme zpracovaná data z počítače. Nepatří sem síťové karty ani optické mechaniky, či falsh disky, sice k tomu může úvodní věta svádět, ale tentokrát máme na mysli zařízení, které umožňují předat data přímo uživateli, tedy monitor, dataprojektor, tiskárnu, reproduktory, eventuálně CNC stroj, které rozebírat nebudeme.

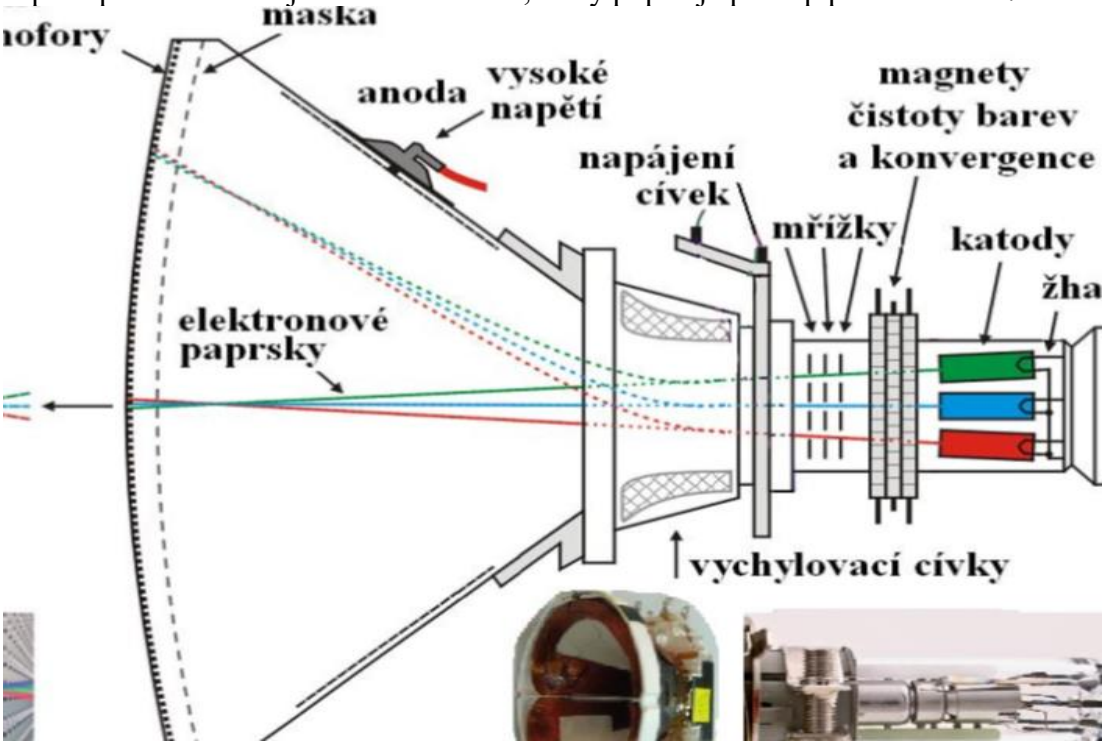
2.2.2.1. Monitor

Monitor je to, co asi zaujme každého na první pohled, když přijde k počítačové sestavě. Dnes si asi většina z vás nedokáže představit monitor, který je tlustší nežli má velikost obrazu, ale takové to v začátcích bylo, u prvních počítačů ani monitor nebyl. Je teda pravda, že monitor je jednou z nejdůležitějších součástí PC sestavy, protože kvalitní monitor chrání naše oči a ruku na srdce, každý si přeje vidět jako ostříž, tak proč si zrak ničit.

Nejprve si představíme jednotlivé základní typy monitorů a následně si řekneme, podle čeho vlastně takový monitor vybrat.

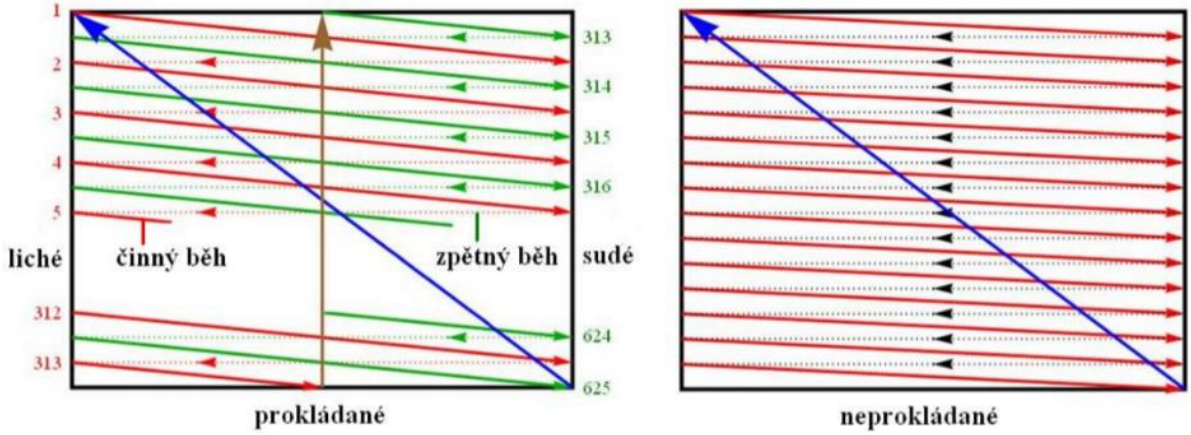
2.2.2.1.1. CRT

Popis a princi zde raději začnu obrázkem, který popisuje princip práce zařízení.



Vnitřek je vakuovaný a v hrdle jsou umístěny tři žhavené katody – trysky R, G, B, ze kterých vystupují elektrony do vakua. Elektronové paprsky jsou urychlovány kladným vysokým napětím anody, které se přivádí na grafitový povlak. Paprsky dopadají na stínítko pokryté luminofory, které převádějí kinetickou energii elektronů na světlo jedné ze tří barev, kdy výsledná barva se získá ze tří složek různé intenzity (aditivní míchání barev); bílá odpovídá maximu R, G a B. Kovová maska zajistí, aby každý paprsek dopadal jen na svůj luminofor (typ masky in line nebo trinitron) a vychylovací cívky odklánějí paprsek ve svislém a vodorovném směru, aby dopadal vždy na určené místo (u monitorů osciloskopů se dříve používalo elektrostatické vychylování pomocí napětí přiváděných na dvojice destiček). Mřížky upravují elektronový paprsek - zaostřují ho, regulují jas a zatemňují zpětné běhy. Magnety čistoty barev a konvergence upravují dráhu paprsku, aby dopadal na správný luminoforový proužek a procházel správnými otvory v masce.

Obraz je vykreslován, pomocí vychylování paprsku elektronů cívkami, po jednotlivých řádcích. Jsou dva druhy zobrazování: prokládané (i - interlaced scan) – vykreslí se nejprve liché, pak sudé řádky (vznikne lichý a pak sudý půlsnímek) a neprokládané (p - progressive scan) – vykreslují se postupně všechny řádky (vznikají celé snímky). Při rychlém návratu paprsku (zpětný běh) zprava doleva je ho třeba zatemnit, aby nevytvářel svítící stopu.



Prokládané řádkování – používá ho analogová televize nebo monitory se slabším grafickým procesorem. Obnovovací frekvence je frekvence, se kterou se vykreslují půlsnímky nebo celé snímky (např. analogová televize 50 Hz – půlsnímky, 25 Hz – snímky) u PC monitoru má být alespoň 75 Hz, ideálně 100 Hz.

CRT monitory mají výborný jas a kontrast (poměr maximálního a minimálního jasu – bílá a černá), velký pozorovací úhel, rychlou odezvu, ale slabší geometrii a ostrost obrazu. Dalším neduhem vysoká spotřeba a velké rozměry. Tyto zobrazovací jednotky jsou nevhodné pro lidi s epilepsií (mohou vyvolat záchvat) a lidi trpící závažnými vadami zraku. Doporučují se užívat brýle s antireflexními skly a širším provedením skel, mohou také způsobovat migrény.

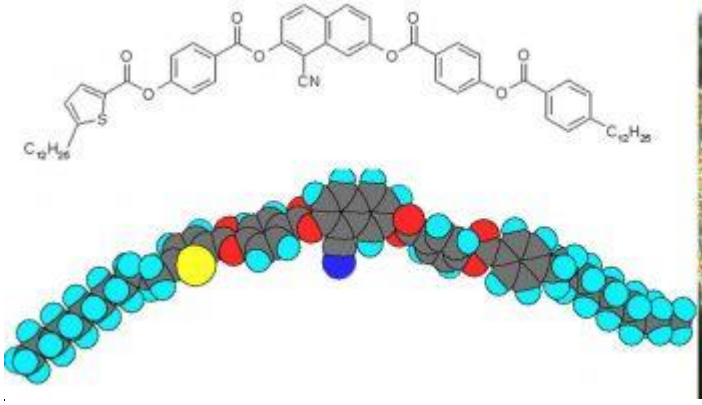


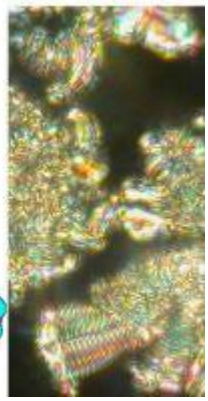
2.2.2.1.2. LCD – Liquid Crystal Display, displej s tekutými krystaly

Využívají vlastností zvláštních látek – tekutých či kapalných krystalů – při průchodu světla. Tekuté krystaly jsou látky, které jsou nejčastěji organického původu a jde o kapaliny s některými vlastnostmi pevných látek.



Jsou tvořeny podlouhlými velkými molekulami,

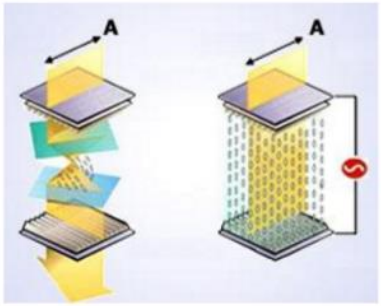


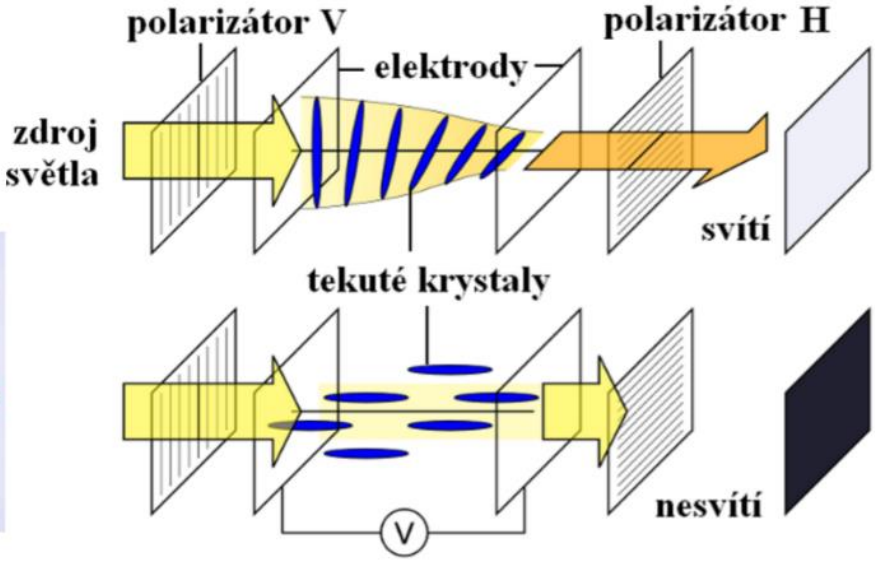


které se chovají různě v elektrickém poli a bez něj. V elektrickém poli se natáčejí do směru pole. Jsou různé typy LCD, které se liší od sebe tvarem a uspořádáním elektrod. Každý bod (subpixel) obsahuje polarizátory, které propouštějí světlo polarizované jen v jedné rovině, pokud polarizované světlo narazí na filtr opačně polarizující, neprojde jím. Když světlo prochází různě mezi natočenými krystaly, odráží se od nich a mění polarizaci.

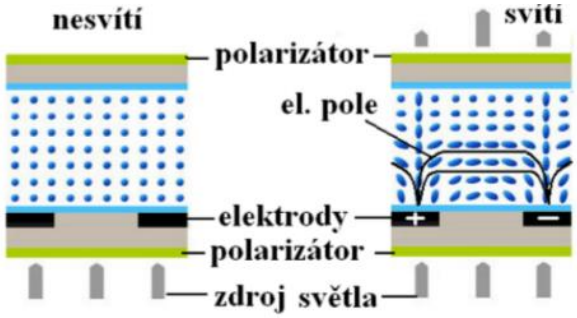
- **technologie TN (twisted nematic)**
  - bez elektrického pole jsou krystaly neuspořádané a světlo prochází
  - v el. poli se krystaly seřadí do jeho směru, světlo nemění polarizaci, neodráží se a neprochází
  - vadný subpixel svítí
  - nejstarší technologie
  - malý pozorovací úhel
  - pomalejší odezva

natáčení polarizační roviny

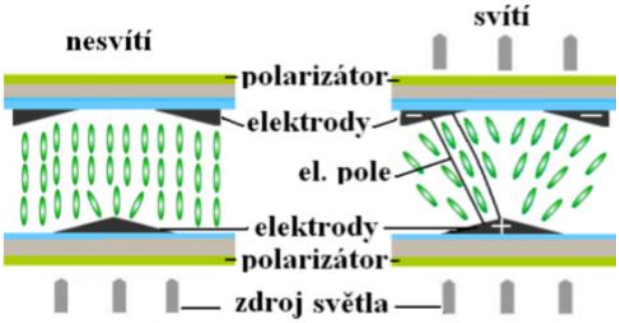




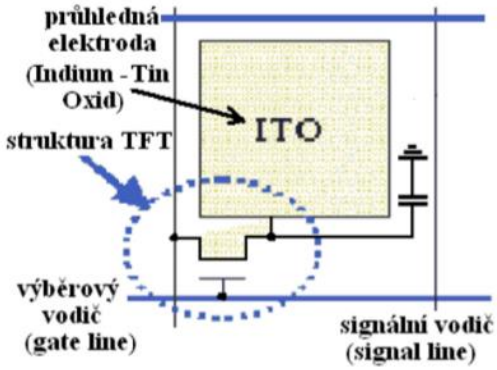
- **technologie IPS (in plane switching)**
  - elektrody jsou uspořádány v rovině
  - bez el. pole světlo neprochází
  - v el. poli se krystaly natočí do jeho směru
  - světlo se odráží a prochází
  - vadný subpixel nesvítí
  - několik verzí (S-IPS atd.)
  - výborné podání barev
  - relativně pomalá odezva
  - velký pozorovací úhel

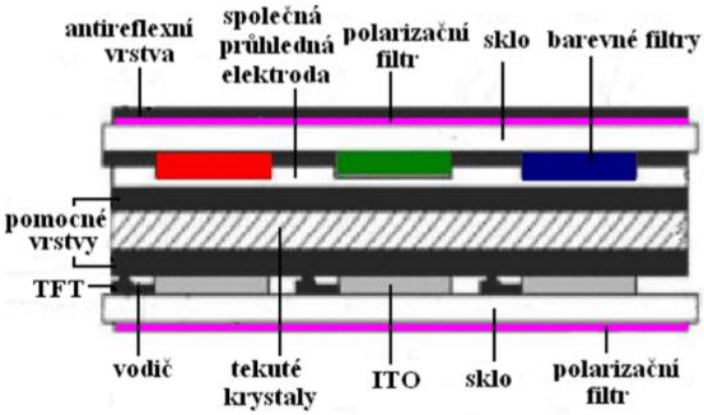


- **technologie VA (vertical alignment)**
  - varianty MVA (multi-domain), PVA aj.
  - pixel je složen z několika (4) oblastí - domén
  - molekuly se natáčejí stromečkovitě
  - vadný subpixel nesvítí
  - krystaly při natáčení vykonají kratší cestu
  - odezva je rychlejší
  - velký pozorovací úhel
  - vynikající kontrast



- **řízení průchodu světla**
  - aktivní matice - technologie TFT *(thin film transistor)*
  - elektrické pole v krystalech vzniká mezi společnou elektrodou a ITO elektrodou samostatnou pro každý subpixel
  - elektrické pole každého subpixelu se řídí tranzistorem zapojeným mezi signální vodič (signal line) a zem
  - tranzistor se řídí napětím přiváděným na výběrový vodič (gate line)
  - spínání: napětí na gate line (výběr řady), pak napětí na signal line (výběr subpixelu)
  - regulace jasu a barev – změnou napětí na ITO
  - tvorba barev – barevnými filtry R, G, B; tři subpixely různých barev tvoří jeden pixel – obrazový bod





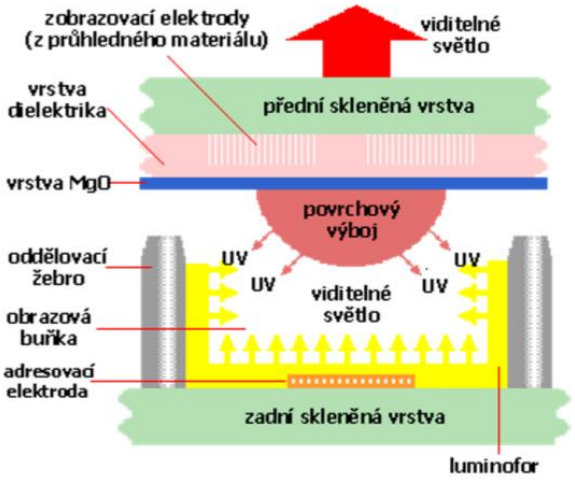
- **zdroje světla**
  - pasivní – odraz vnějšího světla (kalkulačky apod.)
  - podsvícení – požadavek rovnoměrně rozptýleného světla
    - ➢ výbojky – svítící trubice– dvě a více; vyšší počet - rovnoměrnější podsvícení *(CCFL – cold cathode fluorescent lamp)*
    - ➢ bílé světelné diody rozmístěné po okraji panelu, rozvod na plochu světlovody *(LED Edge)*
    - ➢ světelné diody po celé ploše panelu, rovnoměrnější světlo *(LED Direct)*
- **dotykové displeje**
  - dotyková vrstva nesouvisí se samotným LCD; jsou 2 základní technologie
    - ➢ kapacitní – změna kapacity vrstvy po přiložení prstu; tablety, telefony
    - ➢ odporová – změna odporu vodivé vrstvy při stlačení; méně přesné, robustnější; bankomaty, CNC stroje aj.

Vlastnosti

- jas – světelně technická veličina, kvalitní LCD má mít slespoň 300 cd/m^2
- kontrast – bývá až 1000:1, technologie VA má i více; dynamický kontrast – navíc změna intenzity podsvícení
- pozorovací úhel – maximální úhel pohledu na monitor ze strany vertikálně a horizontálně (při zachování určitého kontrastu)
- odezva – doba, za kterou se změní černá na bílou a zpět, dříve problém, nyní bývá jednotky *ms*
- geometrie obrazu, ostrost, spotřeba, rozměry, cena – velmi dobré parametry

2.2.2.1.3. PDP – plazmové monitory

PDP – Plasma Display Panel
plazma – zvláštní skupenství hmoty, směs molekul, iontů, elektronů

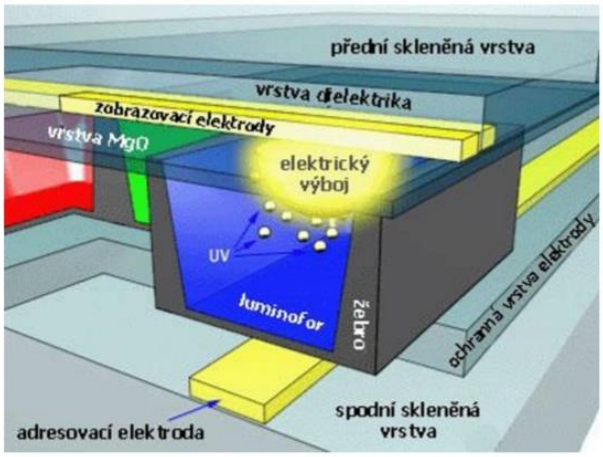


Popis a princip

- využívají elektrického výboje v plazmě
- subpixely - malé komůrky plněnými plynem při nízkém tlaku
- v el. poli vzniká plazmový výboj, který vytváří UV záření
- luminofory na vnitřní straně komůrek mění UV záření na světlo
- světlo je monochromatické, barva je dána druhem luminoforu
- dvě elektrody v komůrce nestačí, protože výboj by nastal po přivedení napětí se zpožděním
- elektrody jsou tři
  - horizontální - zobrazovací
    - scan
    - sustain
  - vertikální - datová čili adresovací
- vhodným sledem napěťových impulsů mezi elektrodami se dosáhne nejprve předionizace, poté výboje
  - inicializační fáze – přivede se napětí mezi adresovací a scan elektrodu, vzniká ionizace a komůrka je těsně pod prahem výboje
  - zobrazovací fáze - přivede se nižší střídavé napětí mezi elektrody scan a sustain, vznikne výboj
  - zhášecí fáze - napětí mezi scan a sustain se sníží, náboj se odčerpá, komůrka je připravena na další inicializaci
- jas se řídí dobou trvání výboje (rozdělení snímku na podsnímky s různou dobou trvání výboje); nelze regulovat napětím

Vlastnosti

- jas a kontrast – menší než LCD
- vynikající pozorovací úhel
- rychlá odezva
- sytější podání barev, vynikající podání černé barvy
- dovoluje velkoplošnou konstrukci – vhodné pro prezentační účely
- naopak menší rozměry nejsou možné s ohledem na minimální velikost komůrek
- velká spotřeba energie
- provedení pro domácnosti jsou již na ústupu

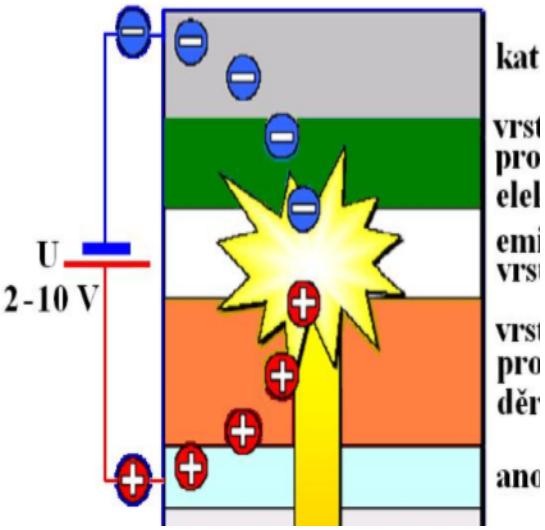


2.2.2.1.4. OLED display

katodou a anodu jsou tři tenké vrstvy organické hmoty
stvy přenášejí nosiče – elektrony a díry
ní vrstvě dochází k rekombinacím elektronů a děr
ná energie se vyzáří ve formě světla
světla záleží na použitém materiálu
ením různých tvarů vrstev lze vytvořit znaky nebo matici
logie AMOLED *(Active Matrix OLED)* - jednotlivé body
e jsou řízeny tranzistory (TFT)

i

ný jas, kontrast, pozorovací úhel
a v řádu mikrosekund
potřeba, nízké napájecí napětí
ně krátký život, zejména u modré barvy



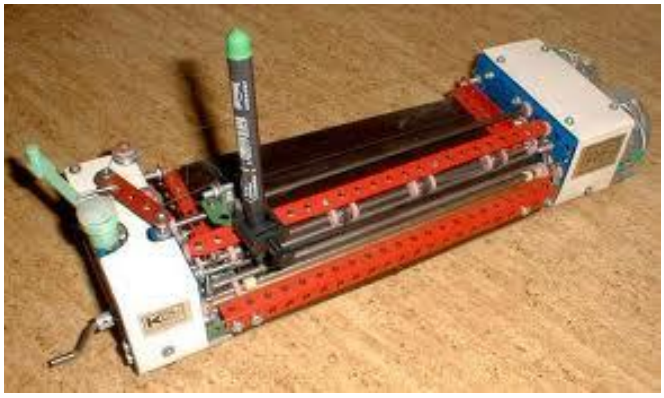
2.2.2.1.5. Jak vybrat monitor

Na trhu je nepřeberné množství značek, typů a druhů monitorů. Na typu zobrazovací jednotky se shodneme, vybereme LCD, respektive LED (LCD prosvětlovaný LED). Důležité je, říct si, co na tom monitoru budu nejvíce dělat, jestli budu dělat vývojáře webových stránek koukat na filmy, nebo zda budu psát texty ve Wordu. Podle hlavního využití je nutné uzpůsobit parametry monitoru. Ideální velikost se pohybuje mezi 17“ a 21“, u notebooků je to 15,6“. další věc je poměr stran, dnes je běžné mít monitor 16:9, ale na některé stoly na místo starého CRT se nemusí taková obrazovka vejít, takže je vhodnější sehnat obrazovku s poměrem 4:3, sice to s sebou nese řadu nevýhod, ale tak je nutné přemýšlet i nad tím umístěním. Dalším důležitým parametrem je kontrast, který by měl dnes být 1000:1 a vyšší. V souvislosti s kontrastem máme jas, který by se měl pohybovat kolem 300 cd/m^2. Zpracování signálu digitálního vstupu by mělo být 10 více bit. Dalším důležitým parametrem jsou v době digitálních monitorů barvy, jejichž počet by měl být 16,7 milionu a více. Obnovovací frekvence by měla být mezi 70 a 100 Hz, doporučuji spíše blíže k té stovce, hlavně pokud se budou sledovat filmy a hrát hry. Dle účelu je nutné zvolit i pozorovací úhly, pro sledování filmů a televize se bude hodit monitor s větším pozorovacím úhlem, naproti tomu na kancelářskou práci stačí menší pozorovací úhly. Z vlastní zkušenosti doporučuji matné provedení obrazovky, hlavně pro notebooky, protože jinak je z toho dobré zrcadlo a venku na terase nevidíte v tom notebooku skoro nic.

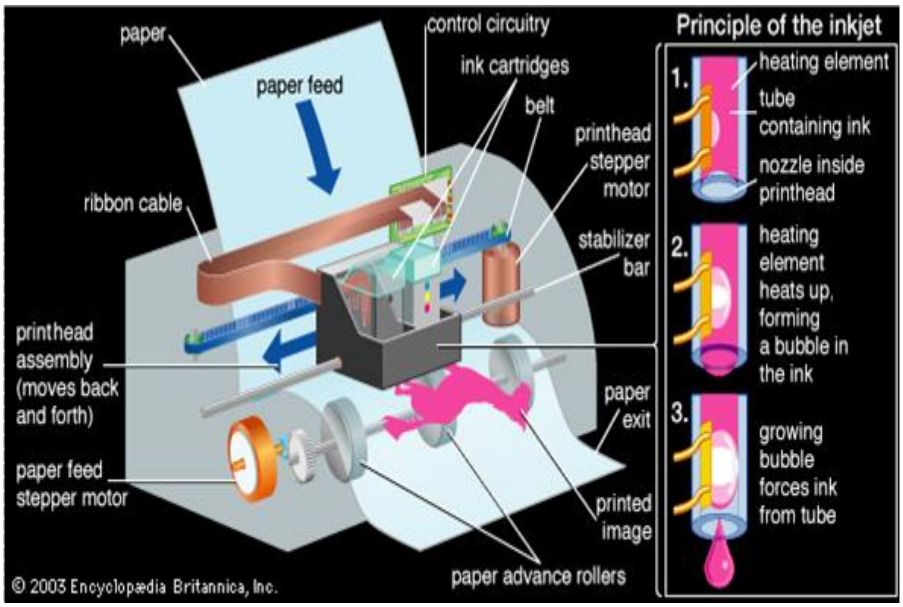
Důležitým parametrem jsou také podporovaná rozhraní, alespoň VGA a DVI, HDMI se dnes stává pomalu standardem. Také jako uživatelé oceníme, když má monitor malou energetickou spotřebu, či USB hub.

2.2.2.2. Tiskárny a plotry

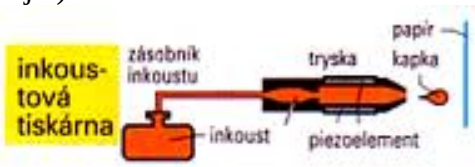
Kromě děrovače děrných štítků byly tiskárny prvním výstupním zařízením připojeným k počítači vůbec. Ke stavbě jednoduché tiskárny stačí propiska, stavebnice merkur, provázek, dvojice elektromotorků a mikročip. Podobně jako tyto kutilské kousky vypadaly první tiskárny, které spatřily světlo světa. Většinou to umělo kreslit grafy a rýsovat, ale lze na nich i provádět tisk.



2.2.2.2.1. inkoustové



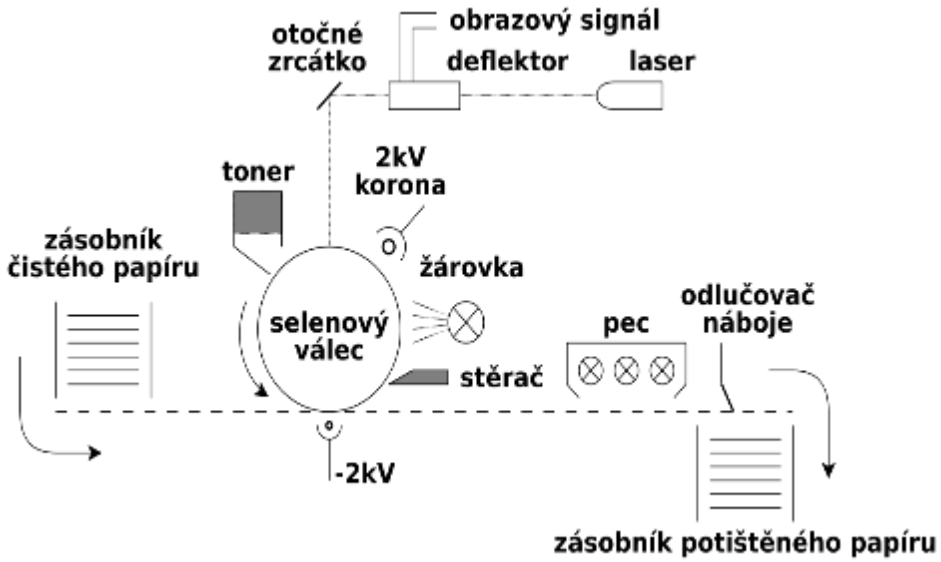
Princip: zahřívání inkoustu z patrony kartrige (v dávce kapičky) a tím je vystřelen na papír (bublejte), nebo vymáčknutím kapičky tryskou (piezoelektrický způsob) pomocí krystalku (inktjet)



Tyto tiskárny poskytují kvalitní tisk, především fotografií. Tisk nemá neduh kopírek, že by se papíry lepili po letech k sobě, na druhou stranu je nutné nechat tisk řádně zaschnout (hlavně velkoformátové fotografie) a také je provoz takovéto tiskárny drahý.

2.2.2.2.2. Laserové

Základem je válec s vrstvou polovodiče (selen), který je nabit stejným nábojem jako toner, laserem se osvítí místa na válci, která se mají tisknout, čímž místo ztratí náboj a na ta místa se přichytí toner, který se potom přitiskne na papír a zapeče se.



Výhody: trvanlivý tisk, rychlost, nižší provozní náklady nežli inkoustová
Nevýhody: Vyšší cena, vyšší energetická náročnost, nižší kvalita barevného tisku

2.2.2.2.3. Jehličkové

Princip psacího stroje (znak se tiskne úderem jehliček přes barvící pásku). Jehličky tvoří matici bodů, ze které je znak tvořen.

Výhody: Levný provoz, možnost tisku kopií (přes kopírák), vysoká rychlost
Pořizovací cena je nevýhodou.





2.2.2.2.4. Plotry

Jsou speciální velkoformátové tiskárny (A3 a větší), které umí bezokrajově tisknout i na atypické formáty. Klasický plotter kreslí obraz pomocí tužky nebo pera. Existují ale i varianty s inkoustovou tiskovou hlavou podobnou klasické tiskárně, případně řezací plottery, kde místo pera je nástroj na řezání (reklamní folie na auta). Medium (papír) může být pohyblivé v jedné ose nebo je pevně umístěno a pohybuje se pouze pero. Použití je převážně na technické výkresy, které kvůli rozměrům nelze na běžné tiskárně vytisknout. Plottery (CNC stroje) jsou také často využívány při gravírování.

Deskové, nebo též stolní plottery, u nichž se papír umísťuje celý na kreslicí plochu. Nad touto plochou je umístěna jakási obdoba portálového jeřábu s kreslicí hlavou, která se pohybuje jednak po kolejničce uvnitř ramene napříč kreslicí plochou a jednak s celým kreslicím ramenem po kolejnici podél kreslicí plochy. Upevnění papíru je zabezpečované téměř výhradně elektrostaticky, u některých starších plotterů se proto vyskytovaly problémy s nedostatečným vybitím po ukončení kreslení. Od deskových plotterů se v poslední době upouští (především u plotterů větších rozměrů), neboť jimi zastavěná plocha bývá neúnosně velká. Také je u nich limitován rozměr obou směrů obrazu velikostí kreslicí plochy.

Stojanový plotter posunuje kreslicí hlavu pouze v jednom směru - napříč papírem. Celá kreslicí plocha také není umístěna na desce, ale papír volně visí po obou stranách plotteru. Pohyb papíru bývá zajišťován přítlačnými válečky podobně jako u mandlu. Vypnutí papíru je zabezpečováno podtlakem pod kreslicí plochou, který je vyvolán pomocí otvorů mimo prostoru, v němž se pohybuje kreslicí hlava nebo u modernějších plotterů též elektrostaticky. Stojanový plotter odstraňuje obě zásadní nevýhody deskových plotterů, velkou zastavěnou plochu a omezení délky obrazu, proto je v současné době více rozšířen. Hlavním záporem je poměrně složitý mechanismus pro posun papíru, který musí zabezpečit, že i při několikerém posuvu se papír nepohne a bod s určitými souřadnicemi bude ležet na stále stejném místě papíru. Tyto

stroje bývají opatřeny zařezávacím nožem, který umožňuje papír zkrátit na potřebnou délku a užít tak k tisku role papíru formátu A0.

V každodenním životě se velice často setkáváme s výstupem zvláštní třídy plotterů označovaných jako vyřezávací plottery. Tyto plottery jsou v zásadě pouze variací na téma perový plotter, od kterých se odlišují pouze tím, že místo pera jsou vybaveny speciálním vyřezávacím nožem. Jako médium se ve spojení s těmito plottery využívá dvojvrstvá fólie, kdy spodní vrstva je pevný nosný materiál, který není vyřezávacím nožem proříznut, na který je přilepena prořezávaná samolepicí vrchní vrstva (může mít různou barvu), jde tedy vlastně o obdobu samolepicí tapety. Některé vyřezávací plottery otáčejí vyřezávací nůž aktivně, podle směru vyřezávání, jiné pouze pasívně odvozeně od pohybu kreslicí (vyřezávací) hlavy a fólie. Plottery s aktivním otáčením vyřezávacího nože umožňují dosáhnout jemnějších přechodů při prudkých změnách směru vyřezávání, řízení otáčení vyřezávacího nože elektronikou však zvyšuje významně jejich cenu a nepříznivě působí i na rychlost vyřezávání.

Inkoustové plottery (někdy též označované jako tryskové) jsou zvětšenou variací na téma inkoustové tiskárny. Jejich technologie tisku je téměř totožná s technologií tisku inkoustových tiskáren. Inkoustové plottery jsou často také schopny emulovat jazyk PCL nebo PostScript a pracovat jako tiskárna. Naopak některé kvalitnější inkoustové tiskárny jsou schopné emulovat i jazyk HPGL/2 a pracovat jako plottery. Je možno říci, že mezi inkoustovými plottery a tiskárnami se vytvořilo určité rozhraní odpovídající formátu A3. Velkou výhodou inkoustových plotterů je možnost kreslit nejen vektorovou grafiku, ale i grafiku bitovou, popř. smíšenou, neboť jsou technicky řešeny obdobně jako tiskárna. Barevné spektrum je omezeno na pouhých osm barev (smísením tří základních barev RGB), popř. 16 barev v případě použití čtyřbarevné kreslicí hlavy (CMYK). Některé dokonalejší inkoustové plottery jsou navíc vybaveny možností několikastupňového řízení dávkování inkoustu (zpravidla čtyři úrovně), čímž je možno dosáhnout širšího spektra barev (64 barev při inkoustech CMY, popř. 256 barev při použití čtveřice inkoustů CMYK). Podobně jako u inkoustových tiskáren je možno optimální kvality dosáhnout při použití speciálního papíru, vyhoví však jakýkoli papír, který neobsahuje papírový prach a na němž se inkoust nerozpíjí. Inkoustové plottery sice nedosahují kvality vektorové kresby perových plotterů, přesto je jejich kvalita pro většinu aplikací plně postačující. Jejich další nevýhodou jsou vyšší provozní náklady způsobené vyšší cenou inkoustu. Výhodami je především možnost kresby bitové grafiky a vyšší rychlost kreslení než u klasických perových plotterů. Ideální oblastí nasazení inkoustových plotterů jsou rozměrově rozsáhlé výstupy v nichž se kombinuje bitová a vektorová grafika (plakáty, billboardy, apod.).



2.2.2.3. Reproduktory

Přeměňují elektrický signál na mechanický pohyb membrány, při kterém vzniká zvukové vlnění.

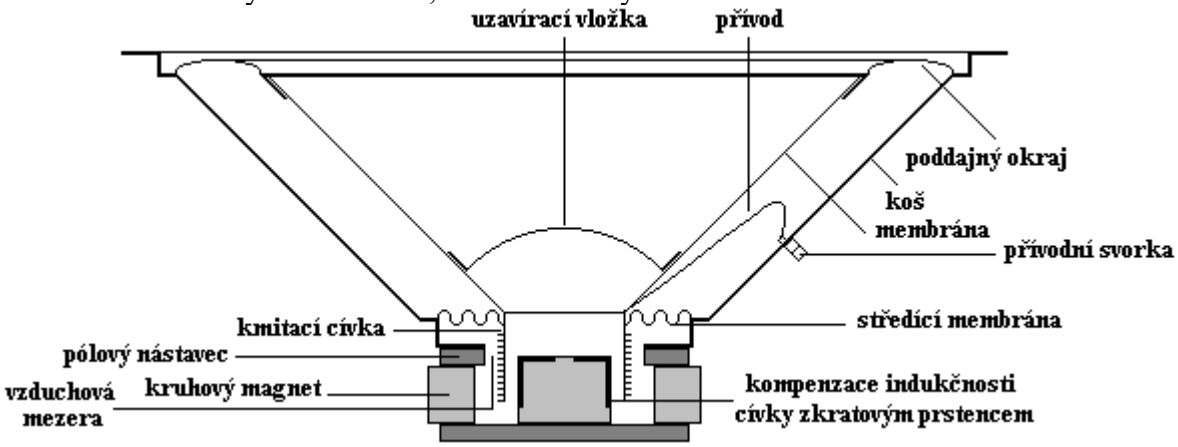
Vlastnosti:

- charakteristická citlivost C = p/S [Pa/VA] - ve vzdálenosti 1 m, závisí na frekvenci
- frekvenční charakteristika - závislost akustického tlaku na frekvenci (v určitém bodě při konst. napětí na reproduktoru)
- jmenovitá impedance Zbývá většinou 4 nebo 8 Q
- maximální příkon P = Uřf"/Zn (předpokládá se činný charakter impedance); špičkový (impulsní) je vyšší než trvalý (výrobci často klamou uváděním tzv. hudebního výkonu, tj. špičkového - ten je několikanásobný oproti trvalému sinusovému RMS)
- směrová charakteristika - závislost akustického tlaku na úhlu v určité rovině (při konst. napětí a určité frekvenci)
- rezonanční frekvence - nejnižší frekvence, kdy má impedance maximální hodnotu; určuje dolní mezní frekvenci reproduktoru
- zkreslení - lineární, nelineární (harmonické, intermodulační....)

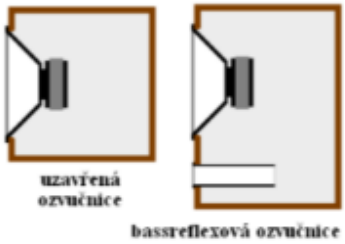
Existuje více typů reproduktorů založených na stejných principech jako mikrofony. V praxi se však vyskytují převážně elektrodynamické reproduktory.

2.2.2.3.1. Elektrodynamický reproduktor přímo vyzařující

Princip: tenká cívka navinutá na papírové podložce je umístěna ve vzduchové mezeře silného permanentního magnetu. Prochází-li cívkou střídavý proud, působí na ni síla podle Ampérova zákona a cívka se rozkmitá. Protože je spojena s membránou (obvykle papírovou) upevněnou v koši. kmitá i membrána a rozechvívá vzduch. Optimální rozměry reproduktoru závisejí na frekvencích - vyšší frekvence, menší rozměry.



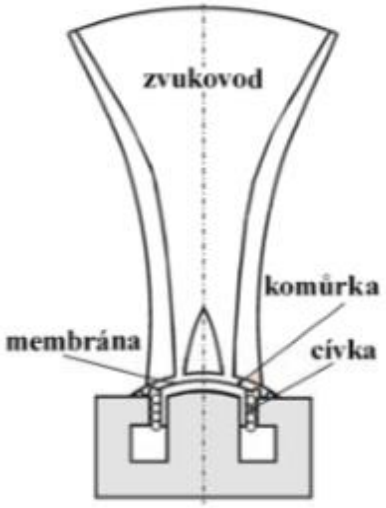
Ozvučnice: Skříň, ve které musí být reproduktor umístěn, aby nedocházelo k akustickému zkratu (akustické tlaky před a za membránou jsou v protifázi a ruší se zejména při nižších frekvencích). Nejlepší jsou dřevěné, vyplněné tlumicí hmotou (vata). Jsou různé druhy. např. uzavřená (nejběžnější), bassreflexová (pro hloubky - nátrubek využívá zpětný tlak vzduchu). Obecně platí, že rozměry ozvučnic jsou větší pro nižší frekvence. Výpočet optimální velikosti je složitý, provádí se poloempiricky.



2.2.2.3.2. Elektrodynamický reproduktor nepřímo vyzařující

Označován též jako tlakový.

Princip: Membrána je umístěna v tlakové komoře. S okolním prostředím je spojena zvukovodem. Plocha membrány je větší než plocha vstupu do zvukovodu, proto rychlost pohybu zvuku na vstupu zvukovodu je větší než rychlost pohybu membrány. Tím se dosáhne vyšší účinnosti (až 20% oproti jednotkám procent u přímo vyzařujících). To je vhodné pro ozvučení velkých prostor, zvláště venkovních. Lépe se reprodukují vyšší frekvence.



2.2.2.3.3. Vícepásmové reproduktorové soustavy

Obsahují několik reproduktorů pro nižná frekvenční pásma. Nejčastěji jsou třípásmové (výšky, středy, hloubky). Pro filtraci frekvencí se používají na výstupu zesilovače pasivní LC filtry, aktivní výhybky, nebo jsou pásmové již výkonové stupně zesilovačů.

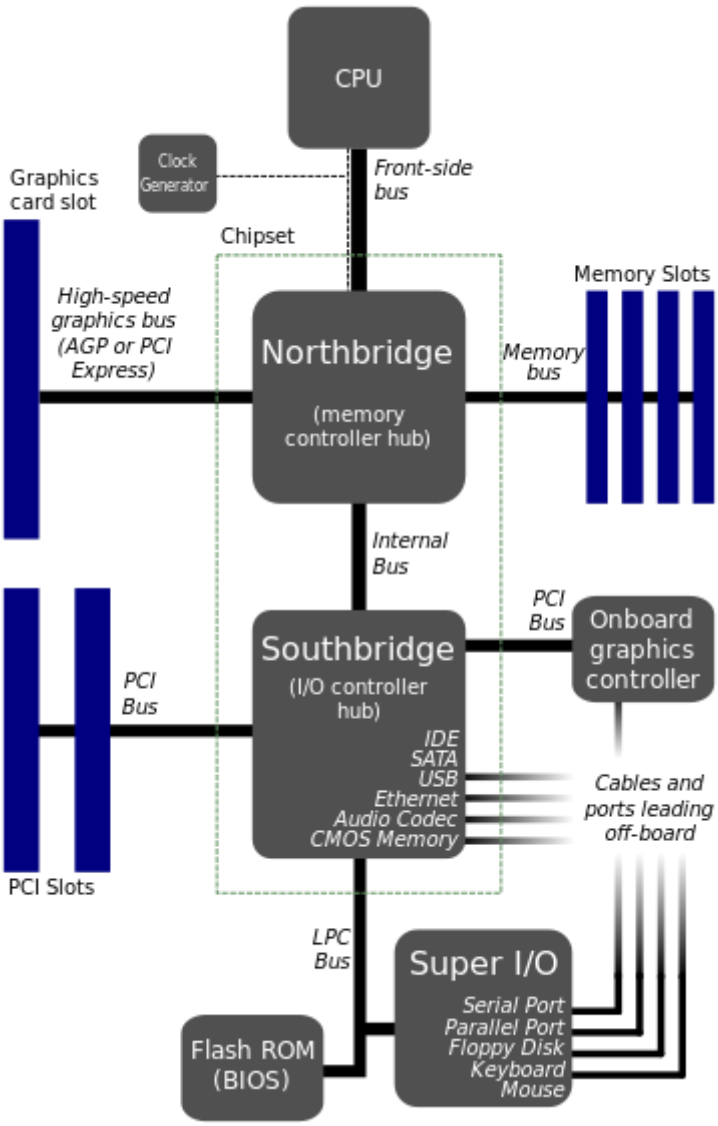
Pro navození dojmu prostorového zvuku se používá dvoukanálový stereofonní systém, v posledních letech i vícekanálový zvuk (5.1, 7.1). Jednička představuje subvvoofer. basový reproduktor od 20 Hz.

Tyto soustavy jsou aktivní, tedy zesilují každé pásmo samostatně, nebo pasivní, kdy je zesílený signál pouze rozdělen filtrem na pásma.

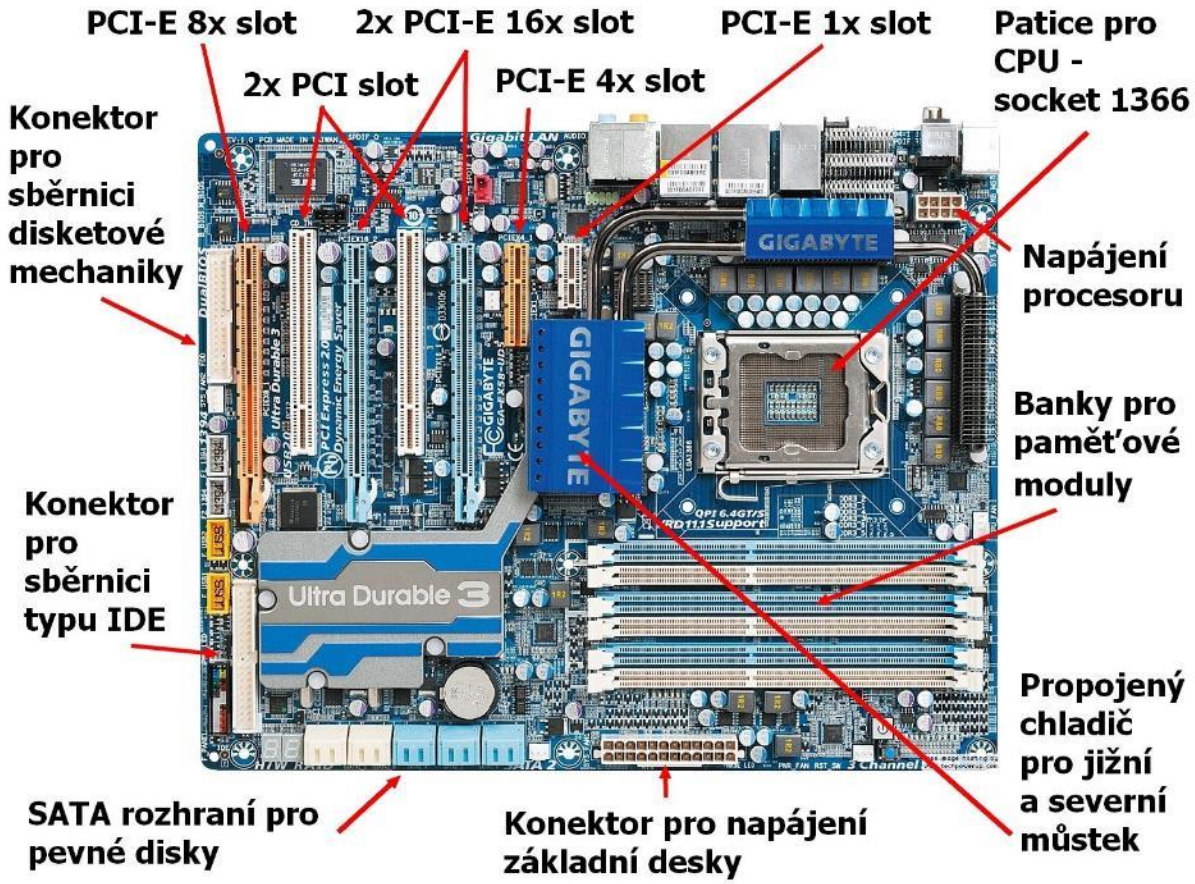
2.3. Obsah skříně

2.3.1. Základová deska

Základní deska (anglicky mainboard či motherboard) představuje základní hardware většiny počítačů. Hlavním účelem základní desky je propojit jednotlivé součástky počítače do fungujícího celku a poskytnout jim elektrické napájení. Postupem času se funkce základní desky rozšiřovala v tom, že sama začínala obsahovat některé součástky počítače, které se do ní dříve musely zapojovat zvlášť. Typická základní deska umožňuje zapojení procesoru, operační paměti. Další komponenty (např. grafické karty, zvukové karty, pevné disky, mechaniky) se připojují pomocí rozšiřujících slotů nebo kabelů, které se zastrkávají do příslušných konektorů. Na základní desce je dále umístěna energeticky nezávislá paměť ROM, ve které je uložen systém BIOS, který slouží k oživení počítače hned po spuštění. Nejdůležitější integrované obvody jsou zabudovány v čipové sadě (anglicky chipset). Fyzicky může jít buď jenom o jeden čip, nebo dva (v tom případě se označují jako northbridge a southbridge). Čipová sada rozhoduje, jaký procesor a operační paměť je možné k základní desce připojit. K základové desce se rovněž připojují periferie.



Základní desky jsou obvykle chlazené vzduchem pomocí chladičů, často namontovaných na větší čipy, jako je Northbridge, v moderních základních deskách. Nedostatečným nebo nesprávným chlazením (K nesprávnému chlazení může často dojít pokud je na základní desce naneseno velké množství prachu, které se tam dostane pomocí ventilace) může dojít k poškození vnitřních součástí počítače, nebo způsobit jeho pád, krajně i požár. Pasivní chlazení nebo ventilátor, který je namontován na napájení je dostačující pro mnohé stolní počítače do roku 1990. Novější desky mají integrovaná teplotní čidla pro detekci teploty základní desky a CPU a také kontrolovatelné konektory ventilátoru, které BIOS nebo operační systém může používat pro regulaci otáček ventilátoru. Některé počítače (které mají velmi výkonné mikroprocesory, velké množství paměti RAM a vysoce výkonné grafické karty) používají systém chlazení vodou (někdy lze jako médium použít dusík, ale celkově je toto chlazení problémové). Nedostatečné chlazení a zvýšení teploty zhoršuje funkčnost základní desky.



2.3.1.1. Sběrnice

Komponenty počítače musí být propojeny, aby se mohly podílet na jeho chodu. Sběrnice slouží pro komunikaci mezi procesorem a dalšími komponenty. Je to soustava vodičů umožňující přenos signálů mezi komponenty, kdy se přenáší: data, adresy, instrukce => 3 typy vodičů: datové, adresové a řídící. Obvykle je sběrnice vyrobena pro konkrétní typ CPU.

2.3.1.1.1. Základní parametry

- **Šířka slova** = počet bitů, které lze po sběrnici naráz přenést. Je tím určen i typ procesoru, který muže se sběrnicí pracovat
- **Frekvence =** Taktovací rychlost
- **Propustnost (rychlost) =** Počet přenesených bajtů za časovou jednotku (obvykle za sekundu)

2.3.1.1.2. Režimy práce

- **Synchronní režim**
  Pracuje synchronně s CPU = jak sběrnice tak CPU mají taktování stejným generátorem
- **Pseudosynchronní režim**
  Dovoluje zpozdit přenos bajtu o určitý počet hodinových period
- **Multimaster režim**
  Sběrnice může být řízena několika zařízeními, nejen procesorem

2.3.1.1.3. Části

- **Systémová sběrnice**
  - Spojuje lokální sběrnici (a CPU) s okolním světem
  - Je zakončena normovanými konektory (sloty)
- **Lokální sběrnice (FSB – front Side Bus)**
  - Je připojena přímo na CPU, kterým je řízena
  - Propojuje obvody na základové desce

2.3.1.1.4. Sloty a porty

- Typy slotů jsou dány typem sběrnice – slouží k připojení přídavných karet
- Sběrnice jsou: systémové a pomocné
- Porty = rozhraní (interfaces) přes která se připojují vstupní a výstupní zařízení
- Konektory dříve byly propojeny pomocí karty, společné pro ID HDD a FFD, která byla zasunuta v jednom ze slotů
- Dnes jsou integrovány konektory na základové desce

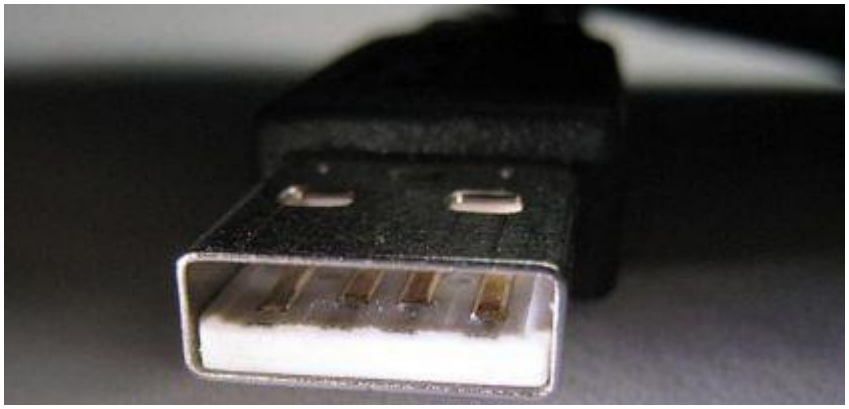
2.3.1.1.4.1. Sériové rozhraní

Bity „proudí“ za sebou jeden po druhém (jako korálky při navlékání na nit). Původně určeno jen pro připojení tiskáren, přenos dat jedním směrem. V roce 1994 došlo k inovaci v podobě normy IEEE 1284, která zavedla obousměrný přenos (tzv. modemové porty LPT), je souborem více dalších norem a není příliš jednoznačná – skutečný režim se musí nastavit v setupu. Norma RS 232C – je nejstarším rozhraním, označuje se COM nebo PRN, je pomalé ale uni-



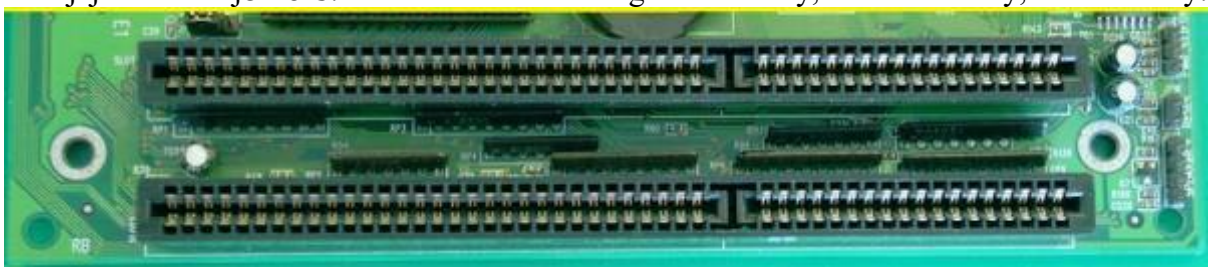
verzální.

Dalším sériovým rozhraním, které bylo vyvinuto je USB (Universal Serial Bus). USB je univerzální sériová sběrnice, moderní způsob připojení periferií k počítači. Nahrazuje dříve používané způsoby připojení (sériový a paralelní port, PS/2, Gameport apod.) pro běžné druhy periférií – tiskárny, myši, klávesnice, joysticky, fotoaparáty, modemy atd., ale i pro přenos dat z videokamer, čteček paměťových karet, MP3 přehrávačů, externích pevných disků a externích optických mechanik. Rychlost přenosu verze 2.0 je až 12 MB/s a u verze 3.0 je to až 5 Gb/s. K jednomu PC lze připojit až 127 zařízení, která většinou využívají technologie plug and play (v překladu „připoj a používej“), což je počítačová technologie umožňující jednodušší rozpoznávání a konfigurace hardware. Umožňuje operačním systémům správně zjistit, jaký hardware je v počítači, na základě čehož může automaticky (s větší či menší úspěšností) k tomuto hardware nainstalovat či spustit příslušné ovladače a automaticky nastavit číslo přerušení, I/O adresu, rozsahy sdílené paměti, případně další skutečnosti (např. detaily o typu zařízení). Pro USB zařízení není nutný restart po instalaci zařízení technologií plug and play. V současné době se rozvinulo USB-C, které je definované pouze na fyzické verstvě, protokolově v sobě ukrývá většinou USB 3.0 a 3.1, ale může ukrývat i USB 2.0 či HDMI, do budoucna má tento konektor nahradit Micro a Mini USB kabely. Příkladem zařízení, která disponují tímto konektorem je k roku 2017 Nicrosoft Lumia 950, Samsung Galaxi S8 či notebook MSI GL62 6QD. USB 3.1 (dle nového značení zcela správně USB 3.1 Gen 2) dosahuje maximální propustnosti 7,2 Gb/s, čistě teoretickým výpočtem byl cíl stanoven na 10 Gb/s.



2.3.1.1.5. ISA

Vyrobena roku 1984, pro počítače 286 a PC-AT. Předchůdcem byla 8 bitová XT-BUS. Šíře jejího slova je 16 b. Používá se na starší grafické karty, zvukové karty, síťové karty.



2.3.1.1.6. MCA

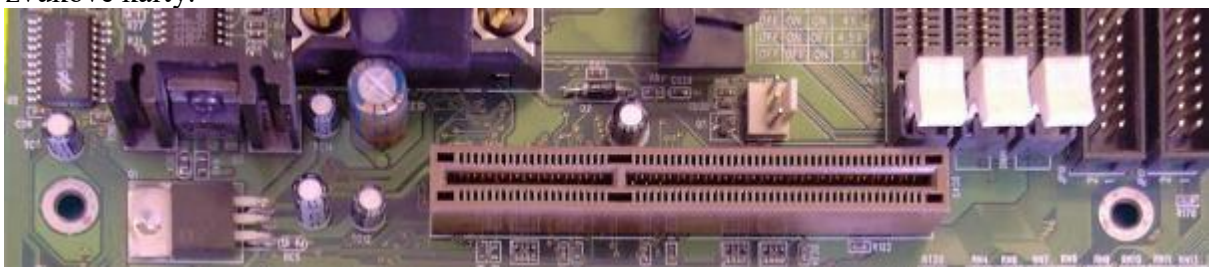
Je kompatibilní s ISA. Má dvě patra kontaktů, horní pro ISA a dolní rozšiřující.Kvůli kompatibilitě ponechána nízká frekvence. Šíře slova se zvýšila na 32 bit. Tato sběrnice vyvinuta IBM pro počítače řady PS/2 se moc nerozšířila, ale byla první 32 bitovou sběrnicí. Využití měla stejné jako ISA.

2.3.1.1.7. EISA

Záměrem poskytnout sběrnici s vysokým výkonem, ale kompatibilní s ISA a tak se roku 1989 objevila na trhu nová sběrnice konkurující MCA. Šířka slova byla obdobně jako u MCA zvolena na 32 bitů. Byla vyvinuta pro procesory Intel 386 a 486. z důvodu kompatibility musela i zde být ponechána nízká taktovací frekvence.

2.3.1.1.8. VESA VL-BUS

Vyrobena roku 1992 pro procesory Intel 486. Je konstruována na rychlost 25-50 MHz, ale má omezenou zatížitelnost. Šíře slova je 32 bit a slouží pro síťové karty, starší grafické karty a zvukové karty.



2.3.1.1.9. PCI

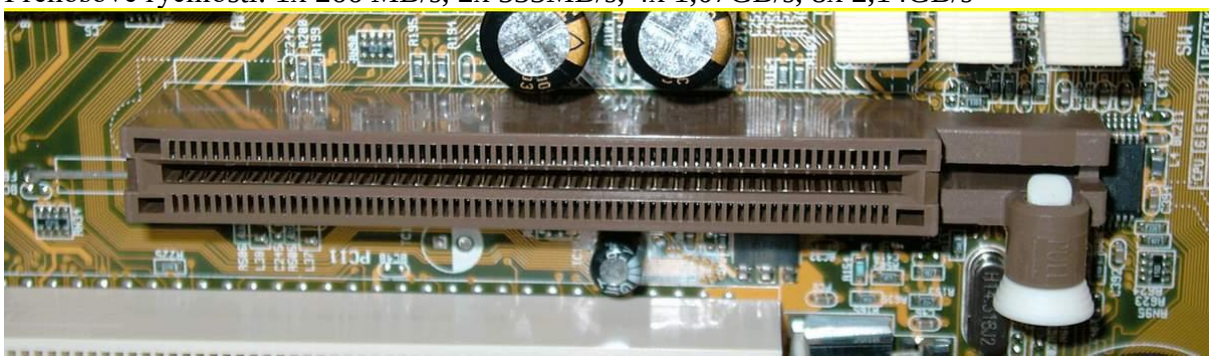
Navržena pro procesory Intel Pentium. Jde o první 64 bitovou sběrnici, ale existuje i 32 bitová verze. Používá se v současnosti pro síťovékarty, bluetoot moduly, zvukové karty a čtečky. Její konfigurace je automatická (plug and Play). Sloty jsou bílé!!. Pracuje s napětím 3,3 V (předešlé s 5 V).



2.3.1.1.10. AGP (accelerated graphics port)

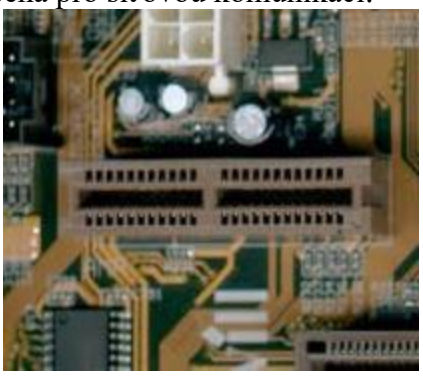
Je to speciální sběrnice propojující grafický adaptér přímo s operační pamětí. Je určena pro přenos obrazu. Vyžaduje minimálně PII (nebo odpovídající AMD) a pracuje s 66 MHz.

Přenosové rychlosti: 1x 266 MB/s, 2x 533MB/s, 4x 1,07GB/s, 8x 2,14GB/s



2.3.1.1.11. CNR (comunication network riser)

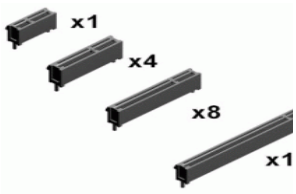
Poměrně nová sběrnice určena pro síťovou komunikaci.



2.3.1.1.12. PCI-E

Nová sběrnice určená pro grafické karty, která je rychlejší nežli AGP. Pracuje na frekvenci 2,5 GHz a s napětím 0,8 V. V 32 bitovém režimu dosáhne 4 GB/s oběma směry.Existuje v různých rychlostních variantách, grafická karta se zasouvá do nejrychlejší (PCI –E 16x). Od rychlosti je odvozena velikost slotu. Číselné označení znamená rychlost (propustnost).

x1 - 250 MB/s
x4 - 1000 MB/s
x8 - 2000 MB/s
x16 - 4000 MB/s
x32 - 8000 MB/s



2.3.1.2. Další konektory

Interní – nachází se na ploše základní desky a připojovaná zařízení obvykle uvnitř počítačové skříně

- IDE
- SATA
- FLOPPY
- napájecí konektory
- konektory pro připojení ventilátorů
- konektory zvukové karty
- rozšiřující konektory USB
- konektory k připojení kabelů od předního krytu
  - indikační LED diody (zapnuto, HDD)
  - USB konektor
  - eSATA konektor
  - IEEE 1394

externí – se nachází na zadním panelu základní desky

- USB, PS/2, eSATA, COM, LPT, VGA, DVI, HDMI, konektory zvukové karty, LAN

2.3.2 CPU (Procesor)

Výroba mikroprocesorů začala roku 1971 na základě podkladů M. E. Hoffa z roku 1969. První firmou vyrábějící procesory byla Texas Instruments. kód je řídící jednotkou počítače – centrální jednotka (Central procesor Unit). Jde o integrovaný obvod, jehož úkolem je provádět instrukce programu ležícího v operační paměti. Základem každého procesoru je logický obvod, který dokáže zpracovat sadu jednoduchých mikroinstrukcí. Každý procesor je též vybaven druhou sadou instrukcí – strojovým kódem. Procesory vyrábí společnosti Intel, AMD, NVIDIA (ARM) a Motorola.



2.3.2.1. Strojový kód

Jde o instrukční sadu, kterou je vybaven každý procesor. Obsahuje instrukce pro manipulaci s pamětí, řízení toku a zpracování instrukcí. Jednotlivé instrukce mají definovaný číselný kód. Procesor umí tyto instrukce detekovat a vykonat. Každou instrukci si procesor interně rozloží na mikroinstukce a ty provede.

2.3.2.1.1. Strojová instrukce

Strojová instrukce se přečte z operační paměti, dekóduje, rozloží na mikroinstukce a provede. Skladba instrukce: kód instrukce a adresy operandů (hodnot), se kterými Instrukce pracuje. Instrukce obsahuje adresy míst, kde jsou operandy uloženy. Programování ve strojovém kódu se pro svou složitost používá spíše výjimečně. Programátoři pracují ve vyšších programovacích jazycích, nebo v jazyce symbolických adres (assembler), které jsou specializovanými programy překládány do strojového kódu příslušného procesoru.

2.3.2.2. Dělení procesorů

Některé odlišnosti mohou vést k nekompatibilitě procesorů, tedy k jejich slučitelnosti. Strojovým kódem - procesory mohou mít pro stejnou instrukci jiný kód (nejsou spolu kompatibilní), výrobci: Intel a Motorola. Dle množství a druhu instrukcí (vnitřní stavbou) – existují 2 koncepce – CISC (kompletní sada instrukcí) a RISC (regulovaná (omezená) sada instrukcí, často účelově zaměřená (grafika, servery). Další parametry:

- Šíře slova – schopnost zpracovat najednou určité množství bitů (8, 16, 32, 64, 128)
- Frekvence – každá základová deska je vybavena generátorem taktů (generuje taktovací impulsy) (jakési pracovní tempo procesoru) – procesor je od výrobce vybaven násobičem (poměr mezi interní frekvencí a frekvencí sběrnice)
- Zapouzdření
- Typ patice
- Velikost cache

2.3.2.3. Součásti

Řadič nebo řídicí jednotka, která zajišťuje řízení součinnosti jednotlivých částí procesoru dle prováděných strojových instrukcí (jejich dekódování, načítání operandů instrukcí z operační paměti a ukládání výsledků zpracování instrukcí).

Sada registrů pro uchování operandů a mezivýsledků. Přístup k registrům je mnohem rychlejší než přístup do operační paměti připojené k procesoru pomocí sběrnice. Registry dělíme na obecné (pracovní, universální) a řídící (např. čítač instrukcí, stavové registry, registr vrcholu zásobníku, indexregistry). Bitová šířka pracovních registrů je jednou ze základních charakteristik procesoru.

Jedna nebo více **aritmeticko-logických jednotek** (ALU, anglicky Arithmetic-Logic Unit), které provádí nad daty aritmetické a logické operace. Některé procesory obsahují jeden (nebo více) matematických koprocesorů (FPU, anglicky Floating Point Unit), které provádějí operace v plovoucí řádové čárce.

Procesory zpravidla obsahují mnoho dalších rozsáhlých funkčních bloků jako například **paměť cache** a různých periferií, které z ortodoxního hlediska nejsou součástí procesoru. Proto vznikl pojem „jádro procesoru“, aby bylo možné rozlišit mezi vlastním procesorem a integrovanými periferními obvody. Vzhledem k současné vysoké integraci tak mnohde dochází k rozmazávání hranice mezi pojmem mikroprocesor a mikropočítač. Některé současné procesory obsahují více jader, takže v jednom pouzdru je obsaženo více samostatných procesorů.

2.3.2.4. Mechanismus stránkování

Nese problém dvou přístupů do tabulek uložených v operační paměti, což může procesor zdržovat. Za účelem zrychlení má procesor vestavěnou vyrovnávací cache paměť zvanou TLB, ve které jsou uloženy posledně použité lineární adresy a k nim odpovídající fyzické adresy.

2.3.2.5. Tabulka základních parametrů

Parametr	Popis	Jednotka	běžný rozsah
Rychlost jádra	Počet operací provedených za jednu sekundu	MIPS (milionů instrukcí za sekundu)	0 – 10000 MIPS
Řada	Oficiální označení velikosti tranzistorů	nm	10000 nm (roku 1971) - 14 nm (2014)
Výrobní proces	Skutečná velikost hradla tranzistorů	nm	45 nm (2004), 25 nm (2014)
Šířka slova	Maximální bitová šířka operandů instrukcí	bit	4 – 64
Počet jader	Počet a typ jader integrovaných v procesoru	číslo	běžně 1 – 8 jader
Šířka externí datové sběrnice	Maximální počet bitů, které je možné během jedné operace přenést z/do čipu	bit	8 – 128
Frekvence externí datové sběrnice (FSB)	Maximální frekvence přístupu do externí paměti RAM	Hz	stovky MHz
Interní paměť cache	Kapacita rychlé interní vyrovnávací paměti integrované přímo na čipu procesoru	Byte	4 – 12 MiB
Taktovací frekvence	Pracovní frekvence procesoru	GHz	0,3 až 4,15
Tepelný příkon	Teplo produkované při plné zátěži po 24 hodinách	watt	40 – 250 W

2.3.2.6. Rychlost procesoru

Nejčastěji používaným vyjádřením rychlosti procesoru je takt procesoru (taktovací frekvence). V jednom taktu jsou uvnitř procesoru provedeny přesně definované operace. Kvůli různým technikám zvýšení výkonu však již dnes není frekvence rozhodujícím faktorem a nelze ji využít jako srovnání ani mezi kompatibilními procesory. Skutečnou rychlost procesu je kombinace taktovací frekvence a dalších jeho vlastností. V procesorech typu RISC trvala každá strojová instrukce právě jeden takt, zatímco v procesorech typu CISC trvala každá strojová instrukce různý počet taktů (jeden až řádově desítky taktů), a proto nelze tyto dvě různé rodiny procesorů přímo porovnávat na základě taktovací frekvence. Dnes se však rozdíly mezi oběma architekturami stírají, takže i CISC procesor může za takt vykonat několik instrukcí. RISC procesory v současné době vykonávají až kolem dvanácti jednoduchých instrukcí za takt. Takové procesory se nazývají superskalární.

Pokud je v procesoru implementován pipelining (pro zvýšení taktu) a/nebo superskalární provádění (pro zvýšení počtu vykonaných instrukcí za takt), je v procesoru více strojových instrukcí v různém stupni rozpracování. V takovém případě má smysl tzv.: Out-of-order execution neboli provádění mimo pořadí kdy pořadí strojových instrukcí je změněno (při zachování správnosti výsledků) pro omezení závislostí mezi zpracovávanými instrukcemi a tedy pro zvýšení výkonu. Pipelining a superskalární provádění se v současnosti často kombinují,

takže například procesory stolních počítačů již běžně mají implementován superskalární pipelining.

Rychlost procesoru ovlivňuje též přítomnost cache, která urychluje přístupy do operační paměti typu RAM. Pokud cache není přítomna, musí procesor při čtení nebo zápisu do paměti čekat na dokončení této operace, což je typicky několik taktů sběrnice, která paměť spojuje s procesorem. Rychlost této sběrnice je typicky nižší, než takt procesoru (ve většině současných procesorů je dokonce interní takt procesoru násobkem této frekvence), a proto je toto zdržení velmi významné. Vřazení rychlé cache mezi procesor a paměť může tyto čekací stavy omezit nebo dokonce úplně eliminovat, což závisí na úspěšnosti cache v predikci následujících operací s pamětí a také na celkové velikosti cache. Cache může být několikastupňová (L1, L2, L3 – čím nižší číslo, tím blíže k procesoru) a může být umístěna i na základní desce počítače. Na výkonnost cache má též vliv optimalizace programu.

Rychlost procesoru je jen jedním z faktorů, které ovlivňují celkovou rychlost počítače. Mezi další faktory patří velikost dostupné operační paměti RAM, rychlost pevného disku, propustnost propojujících sběrnic, ale i rychlost dalších vstupně/výstupních součástí počítače (pro hraní her například grafická karta).

2.3.2.7. Patice

Patice neboli Socket či Slot je konektor na základní desce určený pro připojení procesoru. Původně osobní počítače IBM PC obsahovaly stejné patice pro procesory (tzv. sockety). V roce 1997 společnost Intel začala vyvíjet novou patici nazvanou Slot 1 a společnost AMD na základě koupě licencí k procesoru DEC Alpha vytvořila Slot A. Proto dnes není možné zaměňovat procesory zcela volně. Poslední socket, který používaly všechny společnosti společně, byl Socket 7 do něhož šlo upevnit bezpočet procesorů, firmy jako Cyrix (MII), Intel (Pentium MMX), AMD (K6) nebo IBM vyráběly pro tento socket procesory s frekvencemi v rozmezí 75 - 266MHz, tou nejsilnější verzí pak bylo AMD K6 s frekvencí 300MHz. Socket 7 byl poté vylepšen a výsledná verze s názvem Super Socket 7 byla obohacena o vyšší frekvenci sběrnice tzv. FSB, ta byla zvýšena na 100MHz a dále podporovala slot AGP do kterého se dal vložit grafický adaptér. Poté začala společnost Intel vyrábět procesory pro Slot 1 (Pentium 2) a cesty se rozešly.

S příchodem Slotu 1 (Slotu A) se na krátkou dobu změnila podoba procesorů, tak jak je známe dnes. Až s nástupcem Slotu A, Socketem A též nazývaným 462 (AMD) se procesory vrátily zpět přímo na základní desku a zde též započala myšlenka více jader. Ta byla realizována na Socketu 939. Na přelomu let 2012/2013 jsou nejrozšířenější patice AMD AM3+, G34(pro server) a Intel 1155 pro střední třídu a 2011 pro nejdražší desktopové i serverové procesory.

2.3.3. Zdroj

Asi většina z vás již slyšela vtip o tom, jak se hádá Google a wikipedie, kdo je důležitější. Google říká, najdu všechno a všechny. Wikipedie na to, nevýt mě, tak nemáš kde hledat, vím takřka vše. Přijde elektřina a říká, být vámi bych pomlčela…

Napájecí zdroj počítače (PSU - Power Supply Unit) je zařízení, sloužící ke zpracování střídavého napětí dodávaného ze sítě (u nás 230V/50Hz) na nízké napětí, potřebné k napájení komponent počítače. Nejčastěji dodávané počítačové zdroje spadají do standardu ATX. Povolení nebo zakázání napájení je přes základní desku, která poskytuje podporu pro různé další funkce, jako např. pohotovostní režim.

Nejvíce energie se soustřeďuje v kovové obdélníkové "krabici", ze které vychází několik svazků drátů. Na opačné straně jsou větrací otvory a zásuvky IEC C14 pro připojení do sítě. Štítek připevněný na boku adaptéru obsahuje údaje o maximálním výkonu a certifikaci.

Rozměry ATX zdroje jsou: šířka 150 mm, výška 86 mm a hloubka obvykle 140 mm i když se mohou lišit v závislosti na výrobci.

Napájení používá nucené chlazení, obvykle pomocí ventilátoru s průměrem 120 mm. Jeho prostřednictvím proudí horký vzduch z vnitřku počítačové skříně ven, proto se používá ventilátor s vyšší účinnosti, než které je třeba pro udržení stále teploty ve zdroji. Ventilátor je hlavním zdrojem nepříjemného hluku.

Napájecí zdroje jsou hodnoceny na základě maximálního výkonu. Typické výkonové rozsahy pro domácí a kancelářské aplikace jsou v rozmezí od 300W do 500W (pro miniaturní a obzvláště staré počítače i pod 300W). Napájecí zdroje používané v počítačích pro hráče mají výkon poněkud vyšší - 500W až 800W a zdroje pro servery od 800W až do 1400W. Silnější zdroje jsou určeny především pro velké servery a v menší míře pro počítače s více procesory, více pevnými disky a několika grafickými kartami.

Někteří výrobci přesahují hodnotu výkonu svých zdrojů pro marketingové účely. Je to jeden z důsledků absence norem pro měřící podmínky. Nejčastěji se tak vyskytují:

- hodnota špičkového výkonu místo výkonu trvalého
- definice trvalého výkonu za nerealistických nízkých teplotách (např. při pokojové teplotě, ač uvnitř zařízení se teplota pohybuje okolo 40 °C)

Tyto zdroje vynikají velmi kvalitním vyhlazeným a přesným výstupním napětím, které je potřebné k napájení jednotlivých komponent. Na výstupech je možné najít i záporná napětí určená speciálně pro konkrétní součásti. Zdroje dodávají velký proud, protože zjednodušeně můžeme říct, že platí P = U*I, pokud je P řádově stovky a U je 3.3, 5, a 12V, tak je jasné, že proudy musí být velké. Na standartním zdroji najdeme 24 pinový výstup pro napájení základové desky, několikrát zastoupený pin pro napájení diskových jednotek, 12 konektor pro napájení procesoru, konektory pro grafickou kartu a další.

Špatně naddimenzovaný zdroj způsobuje nerozběhnutí počítače, či jeho nestabilitu. Do vnitřku zdroje nezasahujte, ani se o to nepokoušejte, protože hrozí nebezpečí úrazu elektrickým proudem i několik minut po vytažení napájecího kabelu. Důvodem tohoto nebezpečí jsou dva až čtyři velkokapacitní kondenzátory, které bývají i hlavní příčinou většiny poruch zdrojů. Pokud je zdroj v pořádku, tak všude na jeho obalu je zem, neboli pro neelektrikáře mínus. Zdroj slouží jako zemnící prvek celého počítače a zemní jej pomocí tzv. virtuální země i ve stavu nečinnosti. Pokud by někoho zajímalo, který kabel je zodpovědný za spouštění PC, tak je to zelený kabel z hlavního konektoru pro napájení základové desky, ten sepne zdroj, pokud dojede k jeho uzemnění, tedy pokud by byla porušená izolace a tento kabel by se dotýkal krytu zdroje, tak by počítač byl neustále zapnutý. Zdroj nemá žádný SW, žádné ovladače, žádný firmware, takže nejde jako jediný prvek vůbec ovládat pomocí SW, při vypínání PC z Windows základová deska rozpojí zelený a černý kabel (speciálním polovodičovým prvkem). Proto na starých počítačích (záleží na základové desce) je možné vidět oznámení: Nyní můžete počítač bezpečně vypnout.

Na obrázku vlevo je vnitřek starého zdroje (z dob počítačového středověku Windows 95) a vpravo je moderní 400W zdroj užívaný dnes zcela běžně.





2.3.4. Grafické karty

Grafická karta se stará o grafický výstup na zobrazovací jednotku (Monitor). V případě, že grafická karta obsahuje tzv. VIVO (video - in a video-out), umožňuje naopak i analogový vstup videosignálu např. při ukládání videosouborů z videokamery, videopřehrávačů apod. Se základovou deskou jsou spojeny pomocí sběrnice PCIE. Dříve se pro výpočty používaly odděleně jednotky vertex a pixel, dnes se používají částečně programovatelné jednotky - "unifikované shadery". Pokud je grafická karta integrovaná na základové desce (dnes již vzácně), tak je nejčastěji součástí CPU, které se pak takové CPU označuje jako APU.



2.3.4.1. Základní parametry

- obnovovací frekvence (65 – 85 Hz)
- rozlišení (počet vodorovných a svislých bodů)
- barevná hloubka – 24 bitová je označení jako high-color, 32 bitová jako true-color
- velikost paměti
- parametry grafického procesoru
- výstupní sloty (VGA, HDMI)
- integrace: integrovaná na základové desce, nebo externí
- největší výrobci: ATI (AMD), NVIDIA, Intel

2.3.4.2. Základní součásti

- Firmware (=BIOS)
    - základní programové vybavení grafické karty, které je na vlastním paměťovém čipu. Jsou v něm uloženy informace o modelu grafické karty, GPU, taktovací frekvenci GPU a grafické paměti, napětí GPU a další informace.
- Řadič pamětí - stará se o komunikaci mezi grafickou pamětí a GPU. NVIDIA i AMD podporují až GDDR5

- Paměť RAM (ukládání zobrazovaného obrazu – dnes od 128 MB do 2 GB)
    - zde jsou ukládány informace nutné pro grafické výpočty. Pokud je grafická karta integrovaná na základní desce, používá operační paměť celého počítače nebo má paměť vlastní, nejčastěji některý typ GDDR (GDDR 2, 3, 4, 5), dříve i DDR (1, 2, 3) nebo SDR kvůli nižší ceně.

Typ	DDR	DDR2	DDR3	GDDR2	GDDR3	GDDR4	GDDR5
Efektivní frekvence paměti (MHz)	166 - 950	533 - 1000	700 - 2200	700 - 1000	700 - 2400	2000 - 3600	3400 - 5600
Propust-nost (GB/s)	0,3 - 30,4	12,8 - 32	22,4 - 70,4	22,4 - 32	22,4 - 76,8	63 - 115,2	108,8 - 179,2

- GPU (grafický procesor – šíře slova 128 bit)
    - Slouží k vykreslení dat uložených v RAM. Stavba je závislá od požadavků kladených na grafické zařízení a ve specifických výpočtech (např. matice) je výkonnější nežli CPU. Má velký tepelný příkon (nutné výkonné chlazení) a velký odběr energie, což dělá z grafické karty jednu z nejnáročnějších součástí počítače. Výpočty jsou velice rychlé a efektivní (aby lidské oko bylo schopno zaznamenat pohyb, musí být vykresleno více jak 25 obrazovek za sekundu). Výpočty probíhají paralelně (počítá více rovnic najednou) což je klíček k vysokému výkonu. Specifickou skupinou výpočtů na kterou je zaměřen jsou výpočty s plovoucí desetinnou čárkou. Je možné jej integrovat do CPU, čímž vzniká APU. Unifikované shadery – moderní náhrada za pixelové a vertexové jednotky, každá firma má svoji vlastní architekturu shaderů. Jsou programovatelné a díky tomu nemusí počítat pouze zobrazitelná data, ale i výpočty pro vědu (např. BOINC) a další. Součástí je jednotka TMU (Texture mapping unit), která nanáší textury na objekty a jednotka ROP (Render Output unit) starající se o konečný výstup dat z grafické karty.
- Grafický akcelerátor
    - Kromě klasických matematických strojových instrukcí v něm mohou být implementovány i operace, které umožňují přímo pracovat se základními grafickými prvky a tvary, což umožňuje vykreslovat grafické objekty rychleji, než by bylo možné za použití hlavního procesoru počítače (CPU). Mezi nejběžnější operace z počátku 2D (dvourozměrné) počítačové grafiky patří operace BitBLT (kombinuje několik bitmapových vzorců pomocí RasterOp), obvykle realizované speciálním zařízením nazývaným „blitter“, a operace pro kreslení obdélníků, trojúhelníků, kruhů a oblouků. Moderní GPU kromě toho mají podporu pro 3D (trojrozměrnou) počítačovou grafiku, obsahují funkce pro renderování, podporu digitálního videa a další. Na počátku 90. let se objevilo OpenGL jako profesionální grafické API, které se stalo dominantou na PC a bylo jednou z hlavních hnacích sil vývoje hardwaru. Na konci 90. let se mezi herními vývojáři stalo populární rozhraní DirectX. Na rozdíl od OpenGL trval Microsoft na striktní, jednotné podpoře hardwaru.
- Sběrnice
- RAMDAC - převodník digitálního signálu, se kterým pracuje grafická karta, na analogový, kterému rozumí zobrazovací zařízení (monitor).
- Konektory

- VGA - analogový grafický výstup (používán starými monitory) Možno převést redukcí z digitálního výstupu DVI.
- DVI - digitální grafický výstup (používaný většinou LCD panelů, projektory a novějšími zobrazovacími zařízeními).
- S-Video - norma analogového kompozitního videosignálu pro přenos obrazu v rozlišení SD, nejčastěji režimech s prokládanými řádky (576 pro PAL a 480 pro NTSC).
- Component video - analogový výstup, používá 3 RCA konektory (Y, CB, CR), konektory jsou na některých projektorech, TV, DVD přehrávačích a dalších.
- Composite Video - analogový výstup s malým rozlišením, používá RCA konektor
- HDMI - výstup na zobrazovací zařízení (nejčastěji televizor) s vysokým rozlišením.
- DisplayPort - digitální grafický výstup ve vysokém nekomprimovaném rozlišení. S konektory DVI ani HDMI není kompatibilní.
- DB13W3 - analogový výstup používaný v systémech Sun Microsystems, SGI a IBM.

2.3.4.3. Užívané sběrnice

ISA

Byla vyvinuta roku 1984. délka slova je 16 bit a užívá se pro starší grafické a zvukové karty.

AGP (accelerated graphics port)

- Je to speciální sběrnice propojující grafický adaptér přímo s operační pamětí
- Je určena pro přenos obrazu
- Vyžaduje minimálně Pentium II (nebo odpovídající AMD)
- Pracuje na 66 MHz
- Přenosové rychlosti: 1x 266 MB/s, 2x 533MB/s, 4x 1,07GB/s, 8x 2,14GB/s

PCI-E - (Expres)

- Dnes běžně užívaná sběrnice pro grafické karty
- Rychlejší nežli AGP
- Pracuje na frekvenci 2,5 GHz a s napětím 0,8 V
- V 32 bitovém režimu dosáhne 4 GB/s oběma směry
- Existuje v různých rychlostních variantách, grafická karta se zasouvá do nejrychlejší varianty (PCI –E 16x). Od rychlosti je odvozena velikost slotu.
- číselné označení znamená rychlost (propustnost)
- • x1 - 250 MB/s
- • x2 - 500 MB/s
- • x4 - 1000 MB/s
- • x8 - 2000 MB/s
- • x16 - 4000 MB/s
- • x32 - 8000 MB/s

2.3.4.4. Rozlišení pro 16:9

V bodovém přehledu uvádím označení výstupu, které je užíváno pro daná rozlišení.

- WVGA: 854×480
- HD 720: 1280×720
- WXGA: 1366×768
- WXGA++: 1600×900
- HD 1080: 1920×1080 (full HD)
- QFHD: 3840 x 2160 (ultra HD)

2.3.4.5. Chlazení

- Vzduchové chlazení
    - Na vzduchové chlazení grafické karty se používá buď pasivní kovový chladič, nebo se přidává ventilátor. Případně se používá v kombinaci s heatpipes ke zvýšení chladicí účinnosti.
- Vodní chlazení
    - Většinou se používá pouze u nejvýkonnějších grafických karet, které produkují největší odpadní teplo, a to jenom jako TOP edice. Slouží k snížení teploty a hlučnosti a používá se zejména, pokud je celý počítač chlazen vodním chlazením.

2.3.4.6. Omezení grafických karet

- **Návrh grafické karty** - Grafické karty jsou roztříděny do řad (nižší, střední, vyšší, HIGH-END) a na tom závisí jejich výkon.
    - **PCB** - Stabilitu ovlivňují použité součástky a chladič.
- **Výkonem CPU** - Podle výkonu grafiky je potřeba výkonné CPU, aby byla schopna dodat potřebná data. Výkonné grafické karty postavené na 2 a více GPU potřebují pro plný výkon, buďto vysoko taktovaný dvou jádrový CPU (2,8 GHz a výš), nebo vysoko taktovaný čtyř jádrový CPU (2,5 GHz a výš)
- **Velikostí a frekvenci operační paměti** - Načíst data z paměti trvá podstatně kratší dobu než z HDD.
- **Rychlostí slotu** – standart je PCI-E
- **Ovladače** - starají se o to, aby grafická karta pracovala na plný výkon. Bývají často nedoladěné, problémy řeší až novější verze.
- **Zdroj** - Pokud máte slabý zdroj, nemusí dodat dostatek elektrické energie počítači a díky tomu se PC chová nestabilně a nepodává maximální možný výkon.
- **Nedostatečné chlazení** - Pokud máte na grafické kartě nedostatečné chlazení, může dojít k přehřátí pamětí, napájecích obvodů nebo dalších součástí. Moderní grafický čip má ochranu proti přehřátí, jako CPU, která při dosažení určité teploty přepne kartu do 2D režimu.

2.3.4.7. Výrobci

Pokud řeknu NVIDIA, ATI (respektive AMD) či Intel, tak až na posledně jmenovaného je každému jasné, že jde o formy, které se zaobírají výrobou grafických karet. Z osobní zkušenosti mohu srovnávat všechny uvedené výrobce (pro zajímavost, tato kniha vzniká na počítači s grafickou kartou NVIDIA GEFORCE 650M, což je výkonná herní grafická karta, která je zde užívána k renderování modelů v programu Autodesk Inventor). Při běžné práci průměrné slečínky nepoznáme rozdíl mezi Intel graphics a špičkovým herním modelem NVIDIA, tedy až na spotřebu energie. Z pohledu ovladačů je dalece lepší ATI, které je i o něco úspornější, drobná cena za to je, že ATI Raedon je určena především na rýsování, renderování, vědecké výpočty a kancelářskou práci. ATI zvládne složitější výpočty, ale ve složitých výpočtech má

trošku horší odezvu, za cenu větších možností a nižší spotřeby. Ani nemluvím o tom, že pořizovací cena ATI je dalece nižší, nežli u NVIDIE. Na druhou stranu vývojáři a vášnivý hráči náročnějších PC her ocení spíše herní modely NVIDIE, které jsou pro tyto účely lépe optimalizovány. Pokud jde o Intel, tak si se dvojicí rivalů nezadá v běžné práci, sledování HD filmů a zpracování fotografií i videa, bohužel pro rýsování a hraní náročných her jsou základní APU nevhodná.

Na trhu jsou i další výrobci, ale jejich podíl, např. MSI, není tak velký.

2.3.5. Zvuková karta

Zvuková karta učí počítač „mluvit a poslouchat“. Úkolem zvukové karty je převést digitální záznam zvuku do analogové podoby, kterou jsou schopné převést reproduktory do podoby zvuku a opačně, analogový signál z mikrofonu převést na digitální záznam. Dnes se již jen vzácně připojuje do slotu, v 99 % jde o integrovanou součást na základové desce. Pokud je potřeba kvalitní zvuková karta (např. v nahrávacím studiu) tak je připojena většinou pomocí USB, jen málo pomocí slotu přímo na základovou desku. Externí zvukové karty mají různé další funkce pro úpravu zvuku a mají konektory pro připojení hudebních nástrojů.



Typická zvuková karta obsahuje zvukový čip, který provádí digitálně-analogový převod nahraného nebo vygenerovaného digitálního záznamu. Tento signál je přiveden na výstup zvukové karty (většinou 3,5mm jack-sluchátkový). Zvuková karta také má „line in“ konektor, do kterého je možné připojit kazetový přehrávač nebo podobný zdroj zvukového signálu, který se digitalizuje a uloží (pomocí příslušného počítačového programu) na úložiště dat. Digitalizace se provádí pomocí vzorkování. V každém časovém intervalu se zjistí a zaznamená aktuální stav signálu neboli vzorek. Čím kratší je interval mezi vzorkováním, tím vyšší je vzorkovací frekvence, tím více vzorků bude pořízeno a tím bude výsledný záznam kvalitnější. Druhý faktor, který určuje kvalitu digitálního signálu, je počet použitých úrovní v každém ze vzorků. Nejčastěji používané vzorkovací frekvence: 11025 Hz (telefonní kvalita), 22050 Hz (rádio kvalita), 44100 Hz (CD kvalita), 48000 Hz, 96000 Hz. Počet bitů na jeden vzorek je většinou 8, 16 nebo 24. Třetí konektor, který většina zvukových karet má, se používá k přímému připojení mikrofonu. Signál z něj je možné také nahrávat na úložiště dat nebo ho jinak zpracovat (např. software na rozpoznávání hlasu nebo VoIP). Většina zvukových karet má také MIDI sloužící k připojení např. elektronického klávesového nástroje nebo jiného zdroje digitálního MIDI signálu. Pokud nahráváme z MIDI, tak nemusíme provádět vzorkování. Do počítače se uloží informace o tom, který nástroj hraje, výška tónu, délka tónu, dynamika úhozu na klávesu, atd.

Barevné označení vstupů a výstupů zvukové karty u většiny zvukových karet vyrobených po roce 1999 odpovídá standardu PC 99 firmy Microsoft pro barevné označení externích konektorů.

Barva	Funkce
Pink	**Analogový** mikrofonní **vstup**.
Light blue	Analogový vstup.
Lime green	Analogový **výstup** pro hlavní stereo signál (přední reproduktory nebo sluchátka).
Black	Analogový výstup pro zadní reproduktory.
Silver	Analogový výstup pro boční reproduktory.
Orange	S/PDIF **digitální** výstup.

2.3.6. Optické mechaniky

Počítače využívají optické mechaniky (ODD - optical disc drive) pracující na principu laserového světla, nebo elektromagnetických vln blízkých světelnému spektru, jako část procesu čtení a zápisu dat. Je to periferní zařízení na ukládání dat na optické disky. Některé mechaniky mohou jen číst z disku, ale většina mechanik umí čtení i zápis. Rekordéry se někdy nazývají vypalovací mechanika nebo zapisovací mechanika. Obecné prostředky a technologie zahrnuje rodiny obsahující CD, DVD, Blu-ray, HD DVD (tato technologie se moc nerozšířila). Optické mechaniky jsou většinou využívány k archivaci nebo výměně dat, ale hlavním účelem je instalace OS. Spolu s flash pamětí vystřídala disketovou mechaniku a magnetofonové přehrávače především kvůli ceně, velikosti a technickému vybavení.





2.3.7. Síťová karta

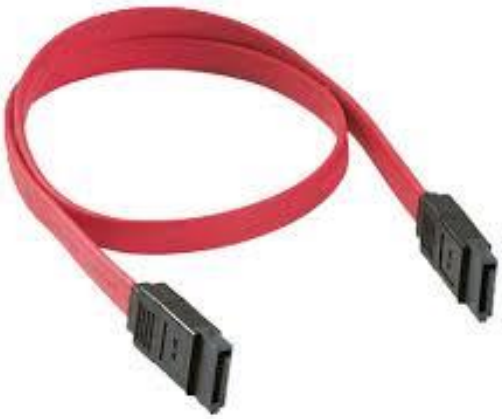
Síťová karta (Network Interface Controller, zkratka NIC) slouží ke vzájemné komunikaci počítačů v počítačové síti. Ve stolních počítačích má podobu karty, která se zasune do slotu (ISA, PCI, PCI-e) základní desky nebo (což je dnes daleko častější varianta) je na základní desce integrovaná. U notebooků je situace podobná, integrace převládá a pro externí připojení se používá rozhraní PCMCIA. Podporují řadu síťových standardů (Wi-Fi, Ethernett, rozhraní pro optické kabely). Často obsahují i Bluetoot modul.



2.3.8. Kabely

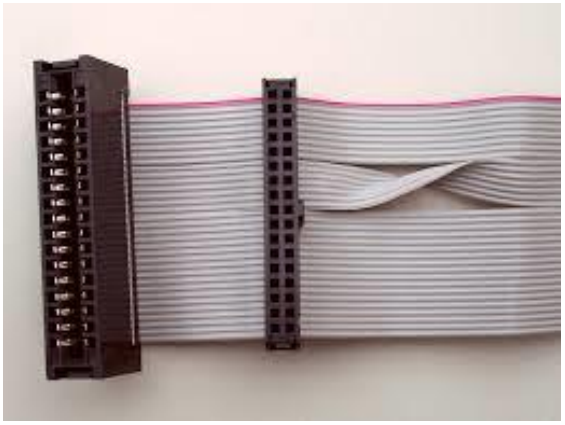
Nedílnou součástí každého počítače je propojovací kabeláž. Kabely neslouží jen k připojení zdroje ke komponentům a připojení periferních zařízení. Kabely též slouží k propojení komponent jako jsou pevné disky, optické mechaniky, disketové mechaniky, porty USB a další se základovou deskou. Kabely jsou specifické pro každé rozhání, jen kabely SATA jsou stejné pro věchy verze, liší se jen označení portů a logika. Nejčastěji se pro připojení HDD a ODD používá SATA kabel, ale dříve se běžně užívaly IDE (ploché široké kabely). Pro napájení větráčků slouží dvoj a trojpinové kabely.

Na obrázku vlevo nahoře je kabel SATA, vpravo nahoře je IDE kabel, vlevo dole je VGA kabel, vpravo dole je flopy kabel.









2.3.9. Chlazení

Každému z nás se někdy stalo, že mu z tepla byl špatně a musím se přiznat, že u mě to je poslední dobou velmi časté, jak je venků více jak 23°C tak je mi zle a opravdu každý ví, že když je vám moc velké teplo, tak se špatně pracuje a vůbec to člověku nemyslí. Počítače jsou na tom stejně, proto potřebují chlazení. Chlazení má za úkol odvést z vnitřku počítače ztrátové teplo, vznikající činností aktivních elektrotechnických součástek, které v počítači zajišťují jeho funkčnost. Odvod tepla je způsoben díky vhodnému použití materiálu (dnes je neoblíbenější bud měď nebo s kombinací hliníku, kvůli pevnosti, lehkosti a hlavně ceně), pouze kovový chladič (pasivní) nebo přidání ventilátory, dále také za použití kapaliny (nejčastěji destilovaná voda nebo případně voda upravená tak, aby se nestala elektricky vodivá) v oběhu nebo kapaliny

pod bodem mrazu (tekutým dusíkem nebo oxidem uhličitým). Každá firma se snaží vyvinout co nejlepší tvar, jak u chladiče pasivního/aktivního, tak u chladiče vodního, kde jde jen o to co nejlépe tvarovat kanálky v chladicím bloku.

Chlazení je lépe předimenzovat nežli podcenit. V balení procesoru, pokud není uvedeno jinak, nalezneme chladič, který je relativně přesně dimenzován na běžnou zátěž, ale nemá dost velký přebytek chladícího výkonu pro dlouhodobé špičkové výkony. Lepší je koupit si chladič externí, ale tento chladič se musí dimenzovat, doporučuji brát vždy pro výkon tak o 30 až 50 W vyšší, nežli udává výrobce procesoru.

Aktivní chlazení je prováděno proudícím vzduchem. Proud vzduchu je obvykle vytvářen ventilátorem. Nejvíce používané ventilátory v počítačích mívají rozměry o hraně 80, 90/92 nebo 120 mm a 550-4000 otáčkách za minutu, dnes jsou preferovány nízko otáčkové ventilátory 700-1500 RPM a větší velikosti. Aktivní chlazení je použito pro chlazení procesoru, grafické karty a zdroje, případně některé součástky mohou být chlazeny proudem vzduchu vytvořeným poblíž některého dalšího větráčku. Pomocí aktivních chladičů se vytváří tzv. „tunely“, v principu jde o dosažení lepšího proudění vzduchu skříní (na přední části je jeden aktivní chladič, který nasaje vzduch do skříně, ten se zde ohřeje a zdrojem nebo dalším aktivním chladičem pod zdrojem je vysáván mimo skříň). Díky zvýšenému průtoku vzduchu způsobeným ventilátorem je potřeba podstatně menší plochy (velikosti) chladiče a díky tomu jsou nižší náklady na výrobu chladiče, i když je součástí ventilátor. S rozměry při stejných otáčkách roste průtok vzduchu.

rozměr	**50 mm**	**80 mm:**	**90/92 mm:**	**120 mm:**	**140 mm:**
Otáčky [RPM]	2500 ~ 4500	600 ~ 4800	600 ~ 4500	400 ~ 3500	500 ~ 1900
hlučnost	20 ~ 27dB	16 ~ 49 dB	13 ~ 47 dB	8 ~ 45 dB	10 ~ 42 dB
průtok vzduchu		14 ~ 73 CFM	25 ~ 93 CFM	24 ~ 133 CFM	30 ~ 105 CFM
využití	čipsety a nízko odběrové GPU	zdroje, CPU, větrání skříně	CPU, GPU, větrání skříně	zdroje, CPU, GPU, větrání skříně	zdroje, CPU, větrání skříně

Ještě existují velikosti 25mm, 40mm, 45mm, 60mm, 70mm, 100mm, 130mm, 172x150mm, 175x175mm, 180x180mm, 220x220mm, 250x250mm a další...



2.3.10. Paměti

Máme zde další prvek společný počítači a člověku, tím je paměť. My si potřebujeme zapamatovat jména, místa, telefonní čísla, to je to samé jako data pro počítač. Také si potřebujeme zapamatovat jak se co dělá, to je v řeči počítače program. Jak je vidět, tak paměť nás lidí slouží ke stejnému účelu jako u počítače. Člověk má jen jednu paměť, počítač nemá tak dokonalou paměť, proto jich potřebuje více různých typů. Paměti počítače můžeme rozdělit do trojice základních skupin: vnitřní, vnější a registry. Základní parametry jsou: kapacita, přístupová doba v nanosekundách, přenosová rychlost (prostupnost), máme také blíže charakterizující parametry: statičnost/dynamičnost, energetická závislost a přístup k paměti.

Vnitřní paměti jsou realizovány z polovodičových součástí, jde o obvody (čipy) na základové desce (např.: operační paměť, cache). Je to matice paměťových buněk, z niž každá má 1 bit.

Vnější paměti jsou vyměnitelná paměťová média pro dlouhodobé uchování a zálohování dat a programů. Záznam je na magnetickém, nebo optickém principu, jde např.: Harddisky, diskety, CD, DVD.

Registry jsou malá paměť umístěná v procesoru sloužící k uložení hodnoty, adresy a instrukce. Používají se ke krátkodobému uložení právě zpracovávaných instrukcí.

2.3.10.1. Kapacita

Udává množství informací, které lze do paměti uložit, tudíž je tím faktorem, který uživatele zajímá nejvíce. Jednotky kapacity máme dvě: bit (signál (je nebo není, 1 nebo 0)) b a Bajt (8b) B. Běžně užíváme násobky, které jsou vidět níže.

kB = 2^{10}B = 1024B

MB = 2^{20}B = 1024kB

GB = 2^{30}B = 1024 MB

TB = 2^{40}B = 1024 GB

Velmi důležitým parametrem, který souvisí s velikostí paměti (velké paměti jsou realizovány jinak, nežli malé registry) je přenosová rychlost (prostupnost), který udává, kolik dat je možné do paměti zapsat za časovou jednotku (1,6 GB/s až 4,0 GB/s u DDR). S přenosovou rychlostí úzce souvisí přístupová doba což je doba, která uplyne od zadání požadavku do jeho zpřístupnění.

2.3.10.2. Dynamičnost

Dynamické paměti postupně ztrácejí uložené informace, i když jsou připojeny ke zdroji energie a informace je nutno cyklicky obnovovat. Paměti jsou realizovány kondenzátorem, kdy se vyhodnocuje přítomnost náboje. Typickým příkladem dynamické paměti je Operační paměť, neboli lidově RAM.

Statické paměti uchovávají informaci po celou dobu připojení ke zdroji, informaci není nutné obnovovat. Paměť je realizována klopnými obvody, nejčastěji JK. Typickým příkladem může být SRAM nebo právě velmi rychlá Cache procesoru.

2.3.10.3. Energetická závislost

Je schopnost paměti uchovat informace i po odpojení energetického zdroje.

2.3.10.4. Přístup k paměti

Sekvenční – před přístupem k informaci je nutné přečíst všechny předcházející záznamy. Sekvenční paměti si člověk může představit jako pásku v audiokazetě, nebo lépe jako šňůru korálků, ze které chce vyvléknout konkrétní korálek.

Přímý přístup (Random Acces Memory) – RAM - je možné zpřístupnit požadovanou informaci přímo. Jako představu si můžeme říct, že si máme vybrat jednu ze sklenic, které stojí na baru, tak si pro ni šáhneme přímo (když víme jakou chceme, nepočítám nějaké vybírání si) a nemusíme kvůli tomu brát všechny předcházející.

2.3.10.5. Vnitřní paměti energeticky nezávislé

Označujeme je jako paměti typu ROM. Jedná se o čipy (paměťové obvody) jež uchovávají informaci i po přerušení napájení a jsou to statické paměti.

ROM (Read Only Memory) je paměť pouze pro čtení (obsah paměti je zadán již při výrobě).



PROM (Programable ROM) lze pomocí programátoru jednou natavit (naprogramovat), po výrobě neosahuje žádné informace.



EPROM (Eraseable PROM) je paměť, do níž lze informaci zapsat obdobně jako do PROM a lze ji rovněž smazat pomocí ultrafialového záření. Tyto pamětí nesmí být vystaveny dlouhodobému působení slunce, jinak dojde k ztrátě dat v nich uložených. Kapacita dosahuje až jednotek MB. Tyto paměti jsou realizovány pomocí speciálních unipolárních tranzistoru, které jsou schopny na svém přechodu udržet elektrický náboj po dobu až několika let. Paměti EPROM jsou charakteristické malým okénkem v pouzdře integrovaného obvodu obsahujícího tuto paměť.



EEPROM (Elecrically EPROM) má obdobné vlastnosti jako EPROM, ale lze jí mazat elektrickými impulsy přímo v zařízení, kde je instalována. Elektrickými impulsy je i mazatelná.



FLASH lze nastavovat i mazat pomocí softwaru přímo v zařízení, kde je instalována. Tato paměť je zapisována i mazána po blcích.



2.3.10.6. Vnitřní paměti energeticky závislé

Energeticky závislé vnitřní paměti jsou určeny jak pro zápis, tak pro čtení. Jde o paměti RAM a RWM.



Dynamické (DRAM) má paměťovou buňku tvořenou mikroskopickým kondenzátorem. Informace je uložena jako elektrický náboj na kondenzátoru. Díky vybíjení kondenzátorů se provádí periodicky refresh = dobíjení (tuto funkci provádí některý z obvodů Chip setu). Užívají se na místech s potřebou velké kapacity a spokojí se s nižší rychlostí (OP). Existuje několik typů lišících se rychlostí a činností (EDO, SDRAM, DDR atd.)

Statické (SRAM) má paměťovou buňku tvořenou bistabilním klopným obvodem. Po dobu napájení je v buňce uložena informace bez ztráty úrovně signálu, protože je reprezentována jedním ze stavů klopného obvodu. Toto paměti jsou rychlé, ale drahé na pořízení a mají vyšší spotřebu energie. Užívají se jako cache.

2.3.10.6.1. SIMM

- Single inline memory module
- 30 pinů
- Kapacita 256 kB, 1MB a 4MB
- Přístupová doba: 70 – 80 ns
- Dnes se již neužívají
- (72 pinů
- Kapacita: 4, 8, 16, 32, 64 MB
- Přístupová doba 60 – 70 ns)



2.3.10.6.2. DIMM

- Dual in-line memory module
- 168 vývodů
- Kapacita 512 MB
- Přístupová doba 7 – 10 ns



2.3.10.6.3. Cache

- Slouží k vyrovnání rychlosti mezi procesorem a pomalejší OP
- Paměť energeticky závislá (typ RAM) – statická
- Paměť určená pro čtení i zápis (RWM)
- Paměť s přímým přístupem
- Může být interní, nebo externí (L1 až L3)
- Většinou kapacita v řádek kB až jednotek MB



2.3.10.6.4. CMOS

Je to výrobní technologie, kterou se vyrábí RAM a Cache i samostatný typ paměti. Po vypnutí PC je napájena vlastním akumulátorem (knoflíková baterie podobná té do hodinek, která je většinou umístěna v těsné blízkosti této paměti).V této paměti je uložena konfigurace a hodiny (reálný čas). V CMOS paměti bývají většinou uloženy:



- informace o typech a kapacitách jednotek pružných disku
- informace o typech, kapacitách a parametrech pevných disku
- typ používané video karty
- kapacita operační paměti
- nastavení parametru Cache paměti
- poradí jednotek pro zavádění operačního systému
- povolení / zakázání různých funkcí základní desky:
- využívání interní a externí Cache paměti
- antivirová ochrana systémových oblastí disku
- prohození poradí jednotek pružných disku
- stínování určitých částí paměti (zavádění programového vybavení z pomalejší ROM paměti do rychlejší paměti RAM)
- činnosti rozhraní pružných disku, pevných disku
- činnosti vstup / výstupních portu
- nastavení rychlosti repetice klávesnice
- nastavení parametru přenosu informací z pevných disku
- nastavení parametru pro režim s úsporou elektrické energie
- nastavení přirazení IRQ úrovní
- nastavení hesla k programu SETUP, popř. k celému počítací

2.3.10.6.5. BIOS

Je paměť typu ROM (konkrétně flash). Její název je odvozen od obsahu (programu BIOS). Basic input output systém je základní program pro přímou komunikaci s hardwarem, který přebírá signály od hardwaru a v určeném formátu je podává OS, čímž zajišťuje nezávislost OS na konkrétní sestavě. Typ BIOS je dán výrobcem základové desky (ABM, Compaq, MSI). Program BIOS je uživateli nepřístupný!!! Uživatelé si jej pletou s programem Setup, ten se ukrývá pod klávesami F9 až F12 při spouštění počítače a umožňuje konfiguraci.



2.3.10.7. Magnetický záznam dat

Paměti s rotujícím médiem ve tvaru disku – dříve i pružné (floppy) dnes již jen pevné (hard) disky. Nejdůležitější části disku jsou samotné záznamové médium, zapisovací a čtecí zařízení.

Záznamové médium

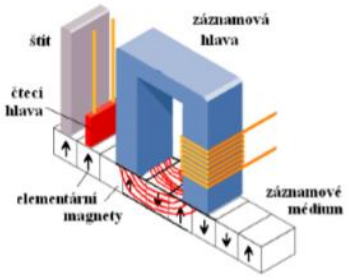
Pevná plotna je opatřena vrstvou feromagnetické látky s tloušťkou v řádu max. desítek nm. V ní se magnetizací vytvářejí elementární magnety. Jejich orientace určuje hodnotu logické 1 nebo 0. Dříve se používal podélný zápis, dnes již jen kolmý (šipky na obrázku), protože má vyšší hustotu.

Zapisovací zařízení

Zapisovací hlava je umístěna (spolu se čtecí hlavou - obrázek) na raménku těsně nad plotnou. Zápis se provádí na induktivním principu podobně jako na magnetofonový pás.

Čtecí zařízení

Původně se používaly hlavy na induktivním principu. Pro zmenšení rozměrů a zlepšení účinnosti čtení byly vyvinuty magnetorezistivní (MR) hlavy, které snímají bity na principu změny odporu senzoru s magnetickým tokem.
V současné době je používají GMR (Giant – obří) hlavy, pojmenované podle jevu (Giant MagnetoResistive Effect - 1988), nikoliv podle rozměrů – jsou naopak menší. Vliv magnetického pole na změnu odporu je výrazně vyšší, což umožňuje zmenšit plochu potřebnou pro záznam bitu. Jde o nanotechnologii, kdy se využívá spinu elektronu – v magnetickém poli má spin vliv na to, zda elektron vrstvou látky projde či neprojde, neboli mění se zdánlivě její odpor.
Pevné disky dále doplňuje pohon rotujících ploten, mechanismus pro nastavení polohy hlav (kmitající cívky) a elektronické části, jako řadič, ROM pro firmware, RAM cache a rozhraní.





2.3.10.8. Vnější paměti – Disketa

V roce 1972 firma DEC přišla s 8“ disketou, která se užívala v prvních osobních počítačích. Byla jednostranná (zápis z jedné stany) (též také označení SS). Disketa je vlastně fólie s vrstvou magnetického materiálu, ve kterém se magneticky provádí zápis dat.

5 ¼“

- Nejprve jednostranná, později i oboustranná (DS)
- Oboustranné ve dvou variantách, lišily se hustotou záznamu (DD a HD (1,2 MB))



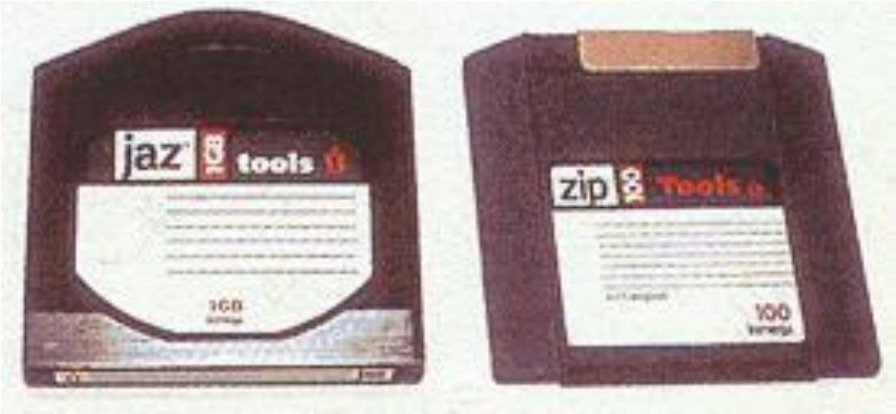
3 ½ ´´

- Verze DD a HD (1,44 MB)



2.3.10.9. Vnější paměti - ZIP disk

Plastový kotouč potažený feromagnetickým materiálem, který je magnetizován hlavou, která je elektromagnetem (magnetizují se plošky, čímž se zapisuje binární kód). Disk je vyroben firmou Iomega. Rozměry odpovídá klasické trošku silnější disketě, ale kapacita zápisu je 100 a 200 MB.



2.3.10.9. Vnější paměti - Bernoulliho disk

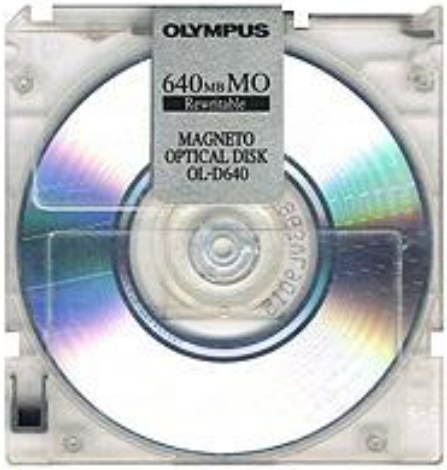
Další svérázný produkt firmy Iomega je Bernoulliho disk, který využívá stejno jmenného jevu. Kotouč je přitlačován podle Bernouliho jevu působícího ve vzduchu, ve kterém se disk otáčí. Dochází k prohnutí disku a tím k jeho přiblížení k hlavě, která se pohybovala na přímočaře na vnitřní straně obalu. Bernoulliho disky se vyrábějí ve formátu 51/4" a jejich kapacita se pohybuje od 20 MB do 200 MB. Připojení k počítači je provedeno pomocí paralelního portu nebo SCSI rozhraní. Mezi výhody Bernoulliho disků patří poměrně vysoká odolnost média proti přetížení (pád, náraz apod.).



2.3.10.10. Vnější paměti - Magneto-optický disk

Magnetooptické disky jsou vyrobeny ze silně magnetických materiálů. Orientaci magnetických částic nelze u těchto disků za normálního stavu ovlivnit. Čteny jsou pomocí magnetooptických mechanik. Jedním z výrobců těchto disků jsou firmy 3M a SONY. Jedním z principů použitých v prototypech je využití rozdílu magnetických vlastností některých materiálů (např. slitiny fermia, železa a kobaltu) za různých teplot - kritický je tzv. Currieho bod 240 °C. V čtecí a zápisové hlavě je kromě optického systému také malá cívka vytvářející slabé magnetické pole. Při záznamu informace se využívá toho, že lokální zmagnetování vrstvy se provede jen v těch místech, která jsou laserem ohřáta na Curieho bod. Čtení je založeno na tom, že odrazivost zmagnetovaných a nezmagnetovaných míst na povrchu disku, se liší. Kapacita

těchto disků je od 128 MB do 9,16 GB. Rozměry jsou podobné DVD diskům, ale běžně se nevyužívají.



2.3.10.11. Vnější paměti - Páskové paměti (streamer)

První mechanické kazetové jednotky firmy IBM byly velmi sofistikované. Pásky mají poměrně dlouhou životnost. Pásky zůstávají konkurenceschopnou alternativou pevným diskům vzhledem ke své nízké ceně. Ačkoli hustota záznamu na cm² je nižší než u pevných disků, dostupné místo na pásce je obvykle větší (záleží na typu). Největší kapacita páskových médií je obecně na stejné úrovni jako kapacita největších dostupných pevných disků (cca 1,6 TB v roce 2010). Během celé své existence pásky poskytovaly velkou cenovou výhodu oproti magnetické diskové paměti, obzvláště pro vytváření záložních dat, kde je přenositelnost médií také velmi důležitá. Rychlý rozvoj velikosti disků a jejich ceny společně s pomalejší inovací pásek snížil podíl produktů magnetických pásek na trhu.

Pásky se používají pro zálohování v bankovnictví a velkých formách, aby byla dat snáze archivovatelná a také jejich zápis byl trvanlivý. Pásky mají vlemi pomalý přístup k datům, tudíž je lépe je užít ke kompletnímu záznamu a následnému obnovení dat. Páskové mechaniky mají integrované HDD pro urychlení procesu zápisu a čtení, ale hlavně pro vyrovnání rychlostí.



2.3.10.12. HDD

Pevný disk (zkratka HDD, anglicky Hard Disk Drive) je zařízení, které se používá v počítačích a ve spotřební elektronice (např. videorekordéry) k dočasnému nebo trvalému uchovávání většího množství dat pomocí magnetické indukce. První komerční pevné disky se objevily v roce 1956, nejprve pro sálové počítače. Předchůdcem pevných disků je magnetická páska a magnetický buben. Jejich současnými největšími konkurenty jsou SSD a USB flash disk, které využívají nevolatilní (stálé) flash paměti. Označení „pevný disk“ se v češtině uchytilo jako obecný pojem a obsahuje i SSD úložiště a další typy nemagnetických pamětí. Prvně byl u osobního počítače užit v roce 1980 (IBM XT) a měl průměr 5 ¼“ s kapacitou 10 MB a odezvou 87 ns. První disky dnešních rozměrů (3 ½“) se objevili o deset let později a měly kapacitu 200 MB. Dnes mají diky i menší rozměr 2 ½“ (externí a pro notebooky) a dostihují kapacit od 80 do 4000 GB.



.

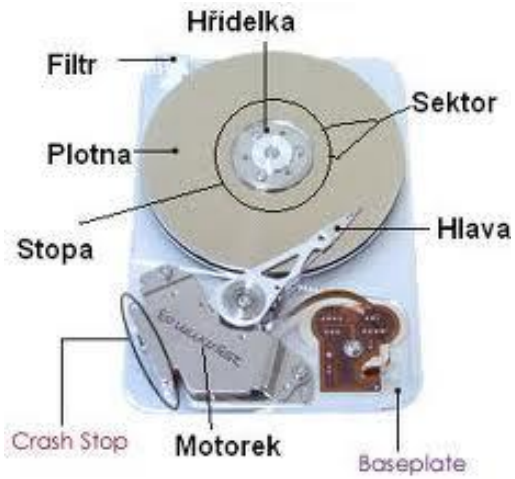
Parametry:

- Kapacita (80 až 4000 GB)
- Přístupová doba
- Typ řadiče (SATA III, SATA II, SATA, SCI, IDE)
- Počet otáček za sekundu (5400, 7200, 10000)
- Velikost diskové cache (8,16, 32, 64 MB)

Diskové médium je z ploten (pevných kotoučů) s feromagnetickým povrchem. Tyto plotny jsou souose uloženy nad sebou a otáčí se 7200 otáček/min (pro stolní počítače). O zápis a čtení se starají elektromagnetické hlavy. Čtecí hlava se nedotýká povrchu, vznáší se 0,3 až 0,5 mikrometru nad plotnou. Pro nečekaný výpadek proudu je určen parkovací sektor, pakování hlaviček. Celý proces se odehrává v hermeticky uzavřeném pouzdře.

Části:

- Médium (plotny)
- Elektromagnetické hlavy
- Mechanika pro pohyb hlav (možnost elektronického řešení pohybu)
- Krokový motor, otáčí plotnami
- Řadič – elektronika, která řídí práci disku – odpovídající řadič je na základové desce



Řadič odpovídá za správné vystavení hlav (nasměrování hlav). Organizuje vlastní čtení a zápis dat, stará se o kódování a dekódování dat. Přenos dat mezi diskem a operační pamětí pomocí DMA kanálu (sběrnice určena jen pro tento přenos).

2.3.10.12.1. Technologie Raid

Současné počítače umožňují instalaci více disků. Je možné disky zapojit více způsoby pomocí technologie RAID. Předpokladem je připojení minimálně 2 HDD. Tehdy můžeme mluvit o jednoduchém diskovém poli. Základních typů polí je 6.

RAID 0 rozděluje data rovnoměrně na všechny disky, v systému se tváří jako jeden disk, zvyšuje se rychlost čtení a zápisu (proužkování)

RAID 1 – miroring, data jsou automaticky ukládána zrcadlově na dva disky a data jsou v případě kolapsu automaticky obnovena z druhého

RAID 5 – data jsou zapisována běžným způsobem, ale na jednom disku je paritní pole, kde jsou specielně zapisována data a mohou být vždy obnovena

RAID 7 – naprosto nezávislé disky, zaručuje pomocí OS nezávislost jednotlivých cest a disků (některé externí boxy)

RAID 2 – uplně samostatná uložiště bez řízení

2.3.10.12.2. Řadič IDE

- K dispozici má 2 DMA kanály
- Přenos je paraelní
- Celkem je možno připojit celkem 4 zařízení
- Nastavení pomocí jumperů na disku a pomocí setup
- Bylo nutno zvolit master a slave (primár a sekundár)

2.3.10.12.3. Řadič SATA

- Seriál ATA
- Přenos sériový
- Každé zařízení má svůj DMA kanál
- Lze připojit 8 zařízení

2.3.10.12.3. Řadič SCI

- Je pro vysoko rychlostní disky
- Vysokorychlostní přenos velkých objemů dat
- Umožňuje připojit až 16 zařízení
- Umožňuje i periferii (tiskárny, páskové mechaniky, skenery)



2.3.10.12.4. Formátování

Každé médium je nutné pro zápis připravit formátováním. Formátování má dvě úrovně.

1. fyzický formát (low formát), který udává kapacita disku, dělí jeho prostor na stopy (soustředné kružnice), které se dále dělí na sektory o jednotné velikost 512 B a tvoří nejmensí jednotku, na kterou lze směrovat hlavy. Stopy o stejném poloměru tvoří cilindr, ve kterých probíhá zápis dat.
2. logický formát, který se dělá pomocí nástrojů OS. Provádí ho běžný uživatel. Vytvoří se Master boot sektor, alokační tabulka, tabulka oblastí, root (kořenový adresář) – první lomítko v cestě je kořenový adresář. Alokační tabulka obsahuje informace o všech klastrech a jejich přiřazení k souborům. Cluster – je užíván jako k ukládání souborů a adresářů v systémovém souboru na daném paměťovém médiu, velikost Clusteru je různá na velikosti HDD a typu systému, 1 cluster je více sektorů. Typ alokační tabulky určuje typ souborového systému.

2.3.10.12.5. Alokační tabulka

Je základem souborového systému. Souborový systém je způsob organizace dat ve formě souboru, tak aby následný přístup k souborům byl co nejefektivnější. Umožňuje uživatelům práci se soubory a vytváření složek (adresářů) a podsložek (podadresářů). Pro souborový systém je určující typ alokační tabulky (FAT-16, FAT-32, NTFS). Soubor je ucelená skupina dat jednoznačně určená svým jménem a umístěním. Jméno souboru má dvě části – jméno a přípona (nativní – typická přípona). Typy: FAT-12 – diskety Windows – je 12 bitová => může obsahovat 212 clastrů, FAT-16 – Windows 95 – umožňuje nax. 2 GB HDD, FAT-32 – Windows 98 – max. 4 TB HDD, ale velikost souboru nemůže přesáhnout 4 GB, NTFS – od Windows 2000 – umožňuje ukládat informace o přístupových právech uživatelů. Ext3, nfs – Linux.

Umístění souboru – popis umístění = cesta úplná (i s kořenovým adresářem) / relativní (vztažená ke složce, ve které je soubor s odkazy). V případě publikování dat se užívají universální formáty (text – RTF a pdf). Strukturu adresářů vytváří uživatel v root. Adresáře mají stromovou strukturu.

2.3.10.13. Vnější paměti - Paměťové karty

Slouží k uložení dat v mobilních zařízeních, kamerách a fotoaparátech. Všechny kapacity paměťových karet mají formát FAT32, sice bychom ji mohli naformátovat na jiný, ale většina mobilních zařízení podporuje pouze formát FAT32. vyrábí se v různých kapacitách od 128 MB do 64 GB (128 GB je vzácné, běžné kapacity jsou: 2, 4, 8, 16, 32 a 64 GB). Běžně se užívají formáty SD (fotoaparáty a kamery), microsd (mobily a tablety), kromě těchto se vyrábí i mini SD, některé mobilní telefony, a XD (fotoaparáty Olympus).

Flash paměti - nevolatilni paměti (nepotřebuji napájení), jsou přepisovatelné. programovatelné po blocích. Základem struktury je tranzistor s plovoucím hradlem - Floating Gate Transistor. Je to unipolámí tranzistor strukturou podobný MOSFETu. Elektricky izolovaná plovoucí elektroda FG má schopnost trvale udržet náboj. Logický stav je dán přítomnosti nebo absencí náboje na FG.

2.3.10.13.1. Programování paměti

Jde o přivedení náboje do FG.

- Fowler-Nordheimovo tunelováni - přivedením dostatečně velkého kladného napětí (+13 V) na CG dojde k silovému působení pole na elektrony ze substrátu P, které „propálí“ oxidovou vrstvu a usadí se na CG; užiti u řazeni NAND.
- hot carrier injectíon - mezi Drainem a Sourcem se vytvoří el. pole pomoci kladného napětí na D (-10 V): elektrony postupuji od S ke D a získávají kinetickou energii; na

CG je však ještě kladnější napětí (+15 V) a elektrony „vstříkne“ do FG; užití u řazení NOR.

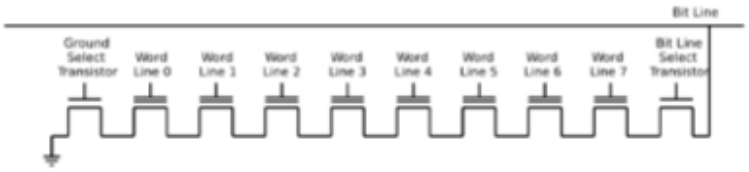
2.3.10.13.2. Mazání paměti

Fowler-Nordheimovo tunelováni s opačnou polaritou napětí na CG odvede náboj do substrátu; mažou se naráz celé bloky. Čtení paměti. Při malém kladném napěti mezi D a S a malém kladném napětí na CG (aby nedošlo k tunelováni). Je-li na FG náboj, odpuzuje v substrátu elektrony a nevytvoří se kanál - logická 0; není-li na FG náboj, vznikne v substrátu kanál a tranzistor se otevře -logická 1.

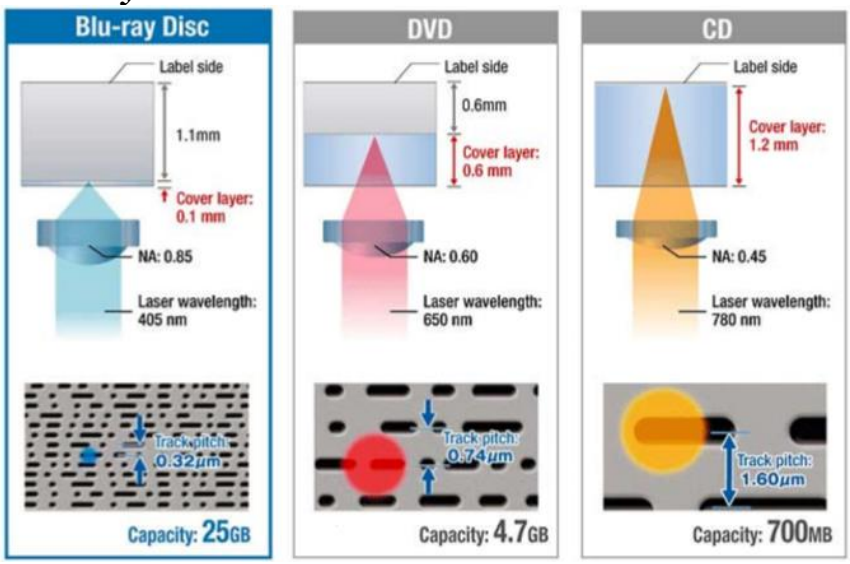
2.3.10.13.3. Zapojení pamětí

Tranzistory lze řadit do série (NAND) nebo paralelně (NOR). Každý způsob má své výhody a nevýhody. U paměťových karet se používá zpravidla NAND - má větší výdrž (až 10á cyklů, než se opotřebuje - poškozené sektory se však při používáni vyřazují), rychlejší zápis i čteni po blocích, pomalejší náhodný přistup.

Přiklad: čteni NANDu; Ground Select a Bit Line Select tranzistor}- jsou výběrové; po výběru řady se přivede na Word Line všech tranzistorů mimo čtený takové napětí, které zaručí, aby se otevřely; následně se na Word Line tohoto tranzistoru přivede čtecí napětí a příslušný stav se odrazí na Bit Line - logická 1 = referenční napětí (zem), tranzistor otevřen; logická 0 = hodnota přednastavená na Bit Line, tranzistor zavřen.

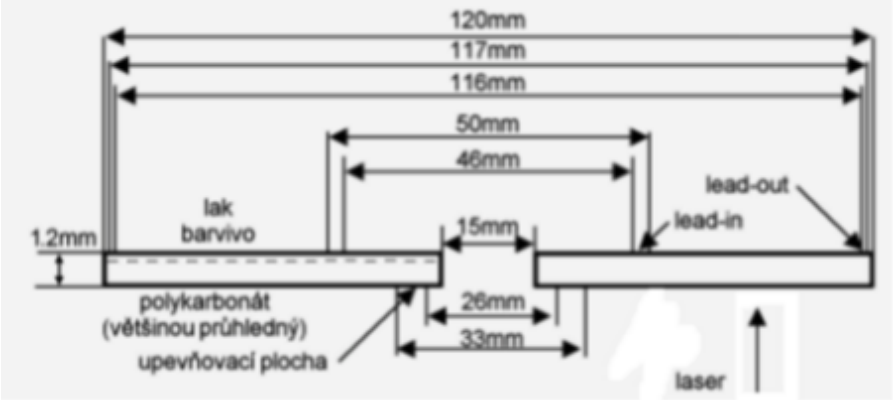


2.3.10.14. Optické disky



2.3.10.14.1. CD (Compact Disc)

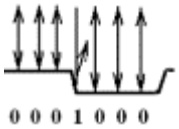
Nejstarší optický disk pro ukládání digitálních dat. Obvyklá kapacita 700 MB. Varianty CD či CD-ROM (lisované), CD-R (zapisovatelné), CD-RW (přepisovatelné). Nejběžnější rozměry disku jsou na obrázku.



Uložení dat je ve spirále – předlisované drážce, ve které jsou pity (prohlubně) a landy. (viz obrázek).



Spirála vede od středu k okraji disku; vzdálenost sousedních stop bývá 1,6 µm; čtecí rychlost (obvodová) je konstantní, mění se tedy otáčky. Délka spirály je 5 – 6 km. Záznam je přístupný pouze ze spodní strany disku – je jednostranný. Pro čtení kompaktních disků se používá laserový paprsek s vlnovou délkou 780 nm. Odraz paprsku od pitu nebo landu (vždy zpět, byť s různou intenzitou) znamená logickou 0.



Odraz od spádu mezi pitem a landem je mimo, znamená změnu, tedy logickou 1 . Z pochopitelných důvodů nemohou být za sebou dvě 1, musí mezi nimi být nejméně dvě 0. Proto se datový tok kóduje pomocí EFM (Eight to Fourteen Modulation).

Lisovaná CD – pity se vytvářejí lisováním, záznam je kvalitnější a trvanlivější. Vypalovaná CD – vypalováním se mění struktura záznamové vrstvy (fyzický stav barviva), vypalování se provádí laserem vyšší intenzity.

2.3.10.14.2. DVD (Digital Video Disc)

2. generace disků na principu podobném CD. Mají vyšší hustotu záznamu (stopy 0,74 µm, kratší vlnová délka laseru 650 nm), 4,7 GB na jednu vrstvu. Varianty DVD či DVD- ROM, DVD-R, DVD+R, DVD-RW, DVD+RW, DVD- RAM. DVD Dual Layer – dvojvrstvý disk; vrchní vrstva je polopropustná, záznam od středu disku k okraji, pak se přeostří laserový paprsek a snímá se druhá vrstva od okraje ke středu, 8,54 GB. Druhy DVD DL, DVD-R DL atd. Vyrábí se i v provedení se záznamovou vrstvou z obou stran. Originální DVD s filmy se dělí dle tzv. regionů. Region odpovídá nějakému většímu zeměpisnému celku (např. region Euroasijský). Svět je rozdělen celkem do 6 regionů. Záznam určený pro určitý region je nečitelný v jiném regionu, tedy pokud nedojde ke změně regionu ve firmware mechaniky, tato změna je možná pouze 5krát, při dalším pokusu o změnu se zablokuje mechanika a je nutné koupit novou. Pokud si fitmy kupujeme v evropských zemích, tak tento problém nemusíme vůbec řešit, problém nastává, pokud si necháme poslat film z USA a jde o disk určený pro jejich zemí, nikoli pro vývoz do Evorpy.

2.3.310.14.3. BD (Blu-ray Disc)

3. generace optických disků. Došlo k dalšímu zhuštění záznamu – stopy 0,32 µm. Užívá se laser s vlnovou délkou 405 nm (odpovídá modré barvě). Kapacita jednovrstvého disku je 25 GB, dvouvrstvý 50 GB. Druhy BD (-ROM), BD-R, BD-RE (Recordable Erasable - přepisovatelný).

2.3.10.14.4. Vrstvy média

- Nosný polykarbonát
- Na zápisové straně je předlisovaná drážka (i nelisovaná média)
- Záznamová vrstva (R – záznamové barvivo) a (RW – slitina kovů)
- Reflexní vrstva – látka dobře odrážející světlo
- Ochranný lak

2.4. Jak postavit počítač?

Pokud se rozhodneme postavit si počítač, tak si musíme uvědomit několik základních věcí. Dobrým pomocníkem nám, obzvláště pokud to není náš denní chléb, je tužka a papír, nebo poznámkový blok či MS Word. Na začátku si musíme určit, kolik peněz máme na počítač k dispozici a k čemu nám ten počítač má sloužit. Jsou účely, které pokud potřebujeme, tak nám bohužel nezbyde nic jiného nežli si koupit notebook, či tablet. Je vhodné si na internetu najít požadavky programů, které budeme využívat a počítač těmto požadavkům naddimenzovat.

Pokud jsme slečínka, která studuje ekonomku nebo medicínu a počítač potřebuje na sociální sítě a filmy, tak postačí procesor z rodiny intel i3. Pokud chceme hrát hry, zpracovávat video, virtualizovat, programovat, psát webové stránky, tak volíme procesor z řady i5. pokud chceme na zařízení renderovat, provozovat webserver nebo jiný typ serveru, pohodlně rýsovat, virtualizovat a programovat ve virtálu, a celkově provádět náročné úkony, nebo je počítač zdrojem naší obživy, tak doporučuji zvážit čadu i7. Procesory jsou základem počítače, na procesoru a základové desce se nevyplatí šetřit, proto nedoporučuji procesory AMD, které jsou náročné na chlazení a mají krátkou životnost. Osobně mám s Intelem kladné zkušenosti, protože mám i 19 let starý procesor od Intelu a není s ním žádný problém. S řadou i5 mám též dobré zkušenosti, virtualizuji úspěšně na tomto procesoru jeden klasický systém a jeden server, kdy i ve virtuálním systému se pracuje s Autodsk Inventor. Když si vybereme procesor dle parametrů popsaných výše, i v kapitole CPU, tak si musíme zvýraznit na našem papíru jeho soket (příslušná patice), což je rozhodující parametr pro výběr základové desky. Nezapomenem, pokud nám to dovolí finance, koupit výkonnější externí chladič.

Základovou desku vybereme na základě soketu a požadovaných parametrů. Doporučuji desku se čtyřmi sloty pro RAM moduly a podporou taktování RAM na 2000 MHz, nebo alespoň 1600 MH. K výběru je nutná znalost odpovídající kapitoly a také naše požadavky, např. na USB. Pokud volíme užití integrované grafiky (kancelářská práce a internet), tak je vhodné, aby deska DVI, nebo HDMI výstup.

Podle základové desky vybereme skříň. Dalšími požadavky pro výběr skříně jsou úložné prostory pro disky a mechaniky, chlazení a také rozměry, aby se nám počítač vešel do počítače.

Při výběru všech dalších komponent se musíme řídit podporovanými parametry námi vybrané základové desky. Obecně RAM moduly volíme tak, aby byl užit jeden modul, nebo byly ukládány v párech. Taktování RAM modulu doporučuji 1600 MHz nebo vyší v závislosti na desce a kapacitu modulu doporučuji 4 nebo 8 GB na modul, aby bylo využití desky efektivní. Zde si musíme dát pozor, abychom nepřekročili nejvyšší možnou kapacitu RAM, která je deskou podporována.

Při výběru disku doporučuji koupit terabytový nebo dvouterabitový klasický disk značky Western Digital. Pro stolní počítače jsou určeny tři a půl palcové disky, 7400 ot/min. Disky vybíráme dle velikosti Cache (ideálně alespoň 16 MB, přeživší je i 8 MB). SSD disky nedoporučuji pro jejich nízkou životnost a problémové zpracování. Pokud by někdo byl takový, že musí mít OS na SSD, tak poté doporučuji SSD značky Intel a klasický disk pro data.

Nesmíme zapomenout na výkonné chlazení a naddimenzovanký zdroj (přebytek výkonu cca 15 až 25%). Ostatní komponenty vybíráme dle předešlých kapitol. DVD vypalovací mechaniku doporučuji pořídit také, není to až taková položka. Nesmíme zapomenout na propojovací kabely!

2.4.1. Sestavení

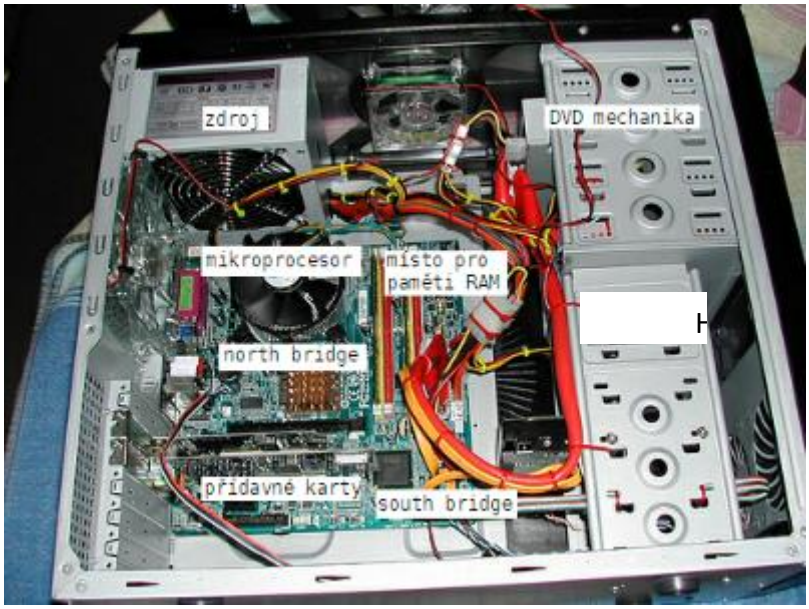
Na internetu je spousta video návodů. Moje doporučení je malounko jiné, ale v praxi mnohokrát ověřené. Jak jsme si říkali v kapitole věnované napájecím zdroji, tak toto zařízení poskytuje uzemnění, tudíž ochranu před statickým nábojem i když není připojeno k zásuvce. Doporučuji položit si skříň na bok a otevřít ji tak, aby byla připravena na vsazení základové desky. Následně rozbalíme zdroj a osvobodíme jeho hlavní napájecí konektor. Krabici se základovou deskou otevřeme a napájecí konektor připojíme na základovou desku. Následně usadíme procesor, který držíme jako mikroskopové sklíčko za hranu dvojice protějších prstů do patice. Patici zaklapneme. Připravíme si chladiče na montáž. Procesor potřeme teplovodivou pastou a připevníme na něj chladič pomocí šroubů. Takto připravenou desku přesuneme do skříně a přišroubujeme.

Po přišroubování základové desky do skříně přimontujeme ventilátory na chladiče a zapojíme je do označených konektorů. Následně nacvakneme RAM moduly (zobáčky musí zapadnout do modulu). Poté můžeme připevnit zadní masku skříně pro porty a vyvést USB porty na přední stranu desky. Zapojíme zapínací a resetovací tlačítko, připojíme signalizační diody. Následně zasuneme do PCIE grafickou kartu a pokud jsme koupili, tak připojíme i síťovou kartu.

Následně uložíme dle návodu skříně pevné disky, které se šroubují křížovým šroubovákem do svého lůžka a připojíme je k základové deskce, stejnou operaci provedeme s optickou mechanikou, zde pozor na kolizi se základovou deskou. Na své místi přišroubujeme a dobře utáhneme zdroj.

Zkontrolujeme, zda jsou opravdu správně zapojeny všechny větráčky chlazení a následně zkusíme připojit napájecím kabelem počítač do zásuvky a zapneme jej. Pokud se nám rozsvítí všechny kontrolky, které mají, chlazení se rozběhne a disky vrní, tak můžeme stroj opět vypnout a uzavřít bočnici.

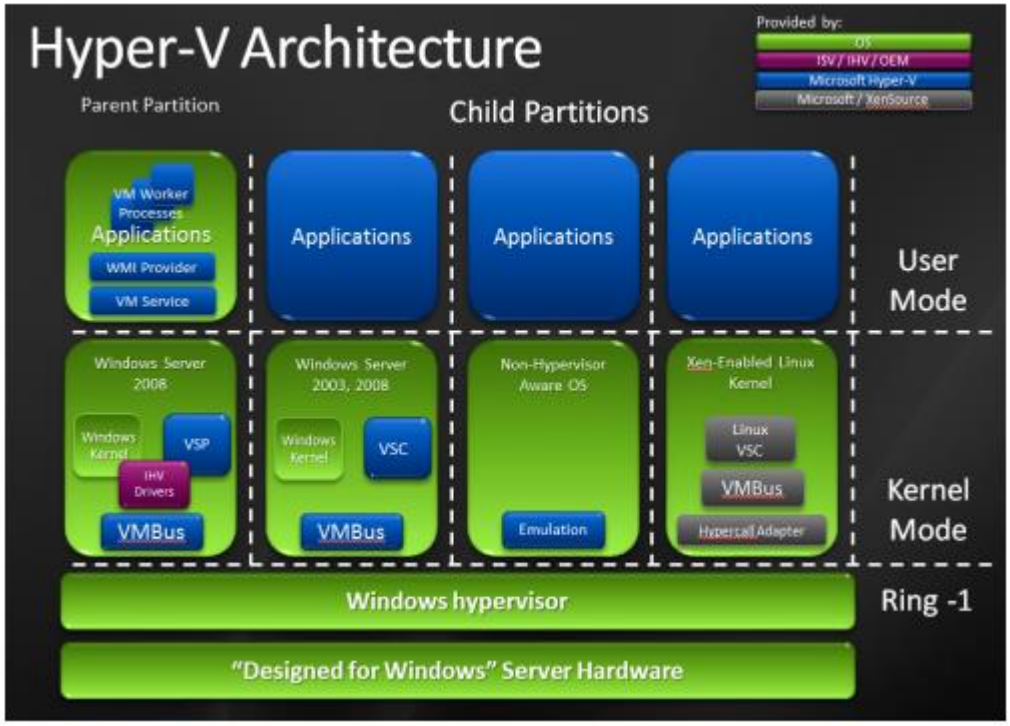
Počítač uložíme na místo určení, neměl by ležet přímo na podlaze a připojíme k němu periferní zařízení. Následně zbývá jen nainstalovat OS a aplikace, aby byl počítač hotov.



2.5. Virtualizace

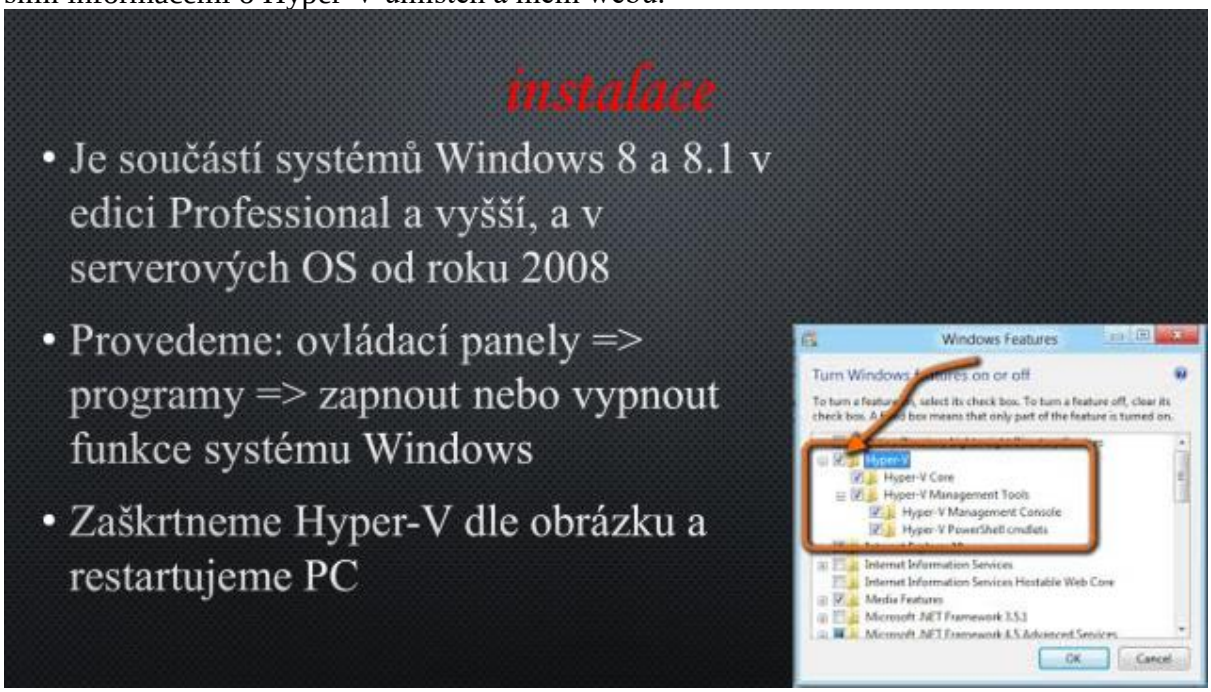
Určitě jste již někde slyšeli o virtuálním systému, nebo virtuálním PC. Relativně do nedávna to byla záležitost týkající se přednostně vývojářů a serverů, které tak šetří náklady za pořízení HW. Dnes již běžně edice professional a to do Windows 7, kam se dal stáhnout dodatečně formou aktualizace MS Virtual PC, ale hlavně u systémů Windows 8 a 8.1, kde se na pár kliknutí bez jakékoli dalšího stahování dá aktivovat serverová platforma Hyper-V.

Virtualizace funguje velmi jednoduše, vytvoří se „imaginární“ kopie vašeho HW, která není přesná, lze ji upravovat dle potřeb virtualizovaného systému. Tento virtuální HW je na takovém pomyslném polštáři, který se jmenuje Hypervizor a pro jeho provoz jsou takřka nulové výpočetní nároky. Hypervizor prostřednictvím ovladačů a nosného OS komunikuje se skutečným HW. Služeb nosného OS je využíváno záměrně, protože HDD je tvořeno souborem (podobně jako třeba rar soubor) ve formátu VHD (pro MS Virtual PC), nebo kvalitnějšího komprimovaného VHDX (Hyper-V), kdy tento soubor na fyzickém disku zabírá jen tolik, kolik je v něm obsaženo (VHDX o něco málo méně), nikoliv kolik je jeho zobrazovaná (nastavená) velikost, virtuální disky se též mohou zvětšovat dle potřeby (jsou tzv. dynamické). RAM je vyhrazena ze skutečné jen nějakou částí (nemůže být celá, musí nosnému systému zbýt alespoň 1,6 GB z celkové kapacity), i tato velikost se mění, nastavená hodnota je jen pro start, proto nastavujeme doporučené hodnoty pro virtualizovaný systém. Ostatní HW je již obrazem své fyzické předlohy. Výhodou je, že ve virtuálním systému není nutné instalovat ovladače pro HW. Na serveru vypadá virtualizace následovně:



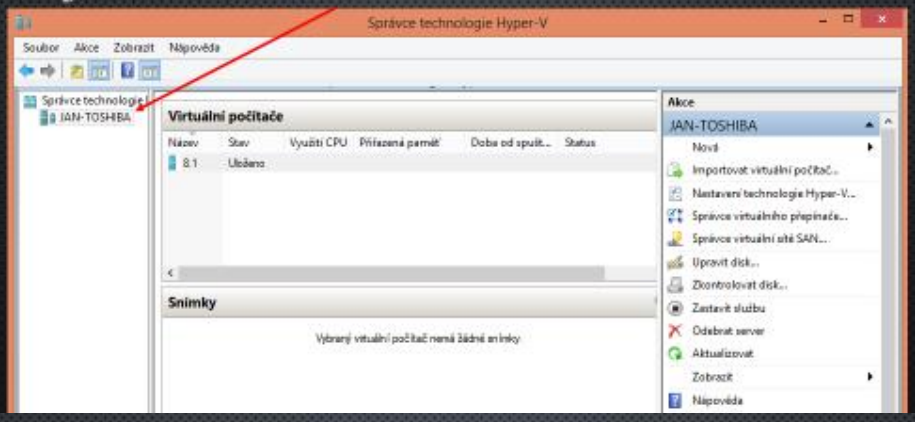
2.5.1. Hyper-V krok za krokem (Windows 8 a 8.1 Pro)

Protože jsem typicky líný a také jsem chtěl tuto nudnou část knihy trošku ozvláštnit, tak zde vkládám prezentační návod pro začátečníky s technologií Hyper-V, který je spolu s dalšími informacemi o Hyper-V umístěn a mém webu.



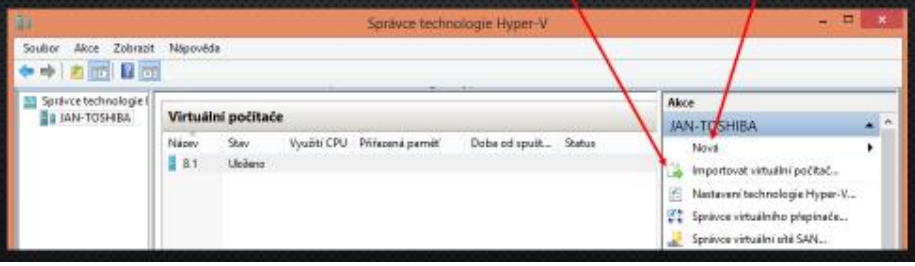
Prvotní spuštění

- Nejprve se nám zobrazí správce technologie
- Přejdeme k serveru



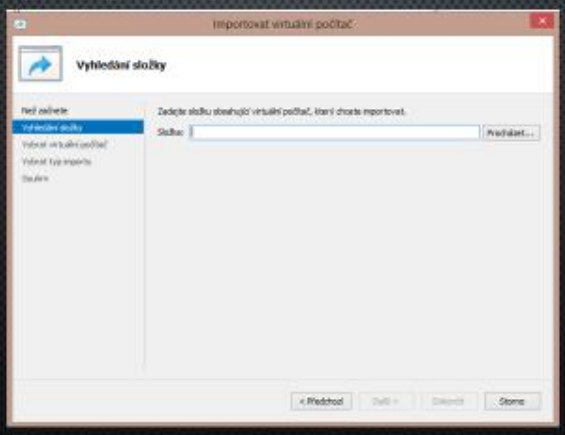
Vytvoření virtuálního PC

- Pokud jsme si přinesli virtuální stroj od kamaráda či z práce, nebo stáhli z TechNetu, tak zvolíme importovat virtuální počítač
- Pokud začínáme na čisto, tak zvolíme „Nová“



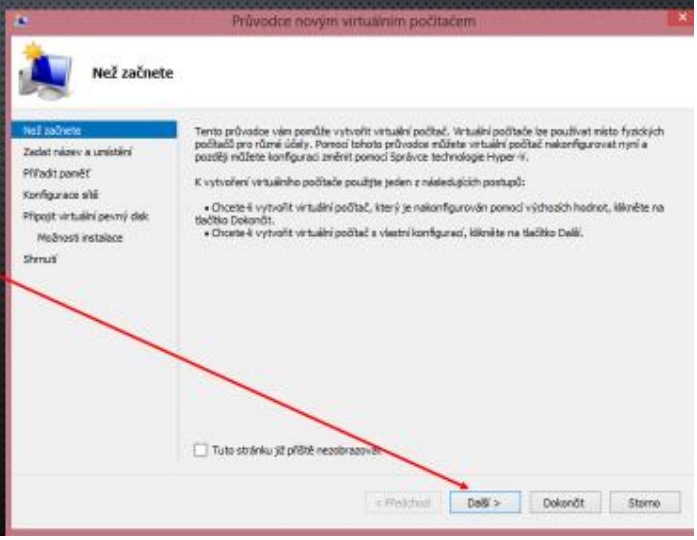
Import virtuálního PC

- Řídíme se pokyny průvodce, není to o nic těžší, nežli připojení přílohy k e-mailu

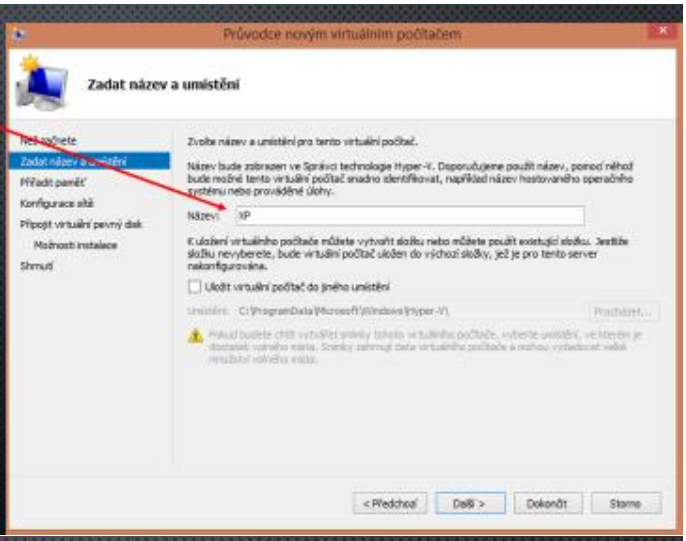


Vytvoření virtuálního PC

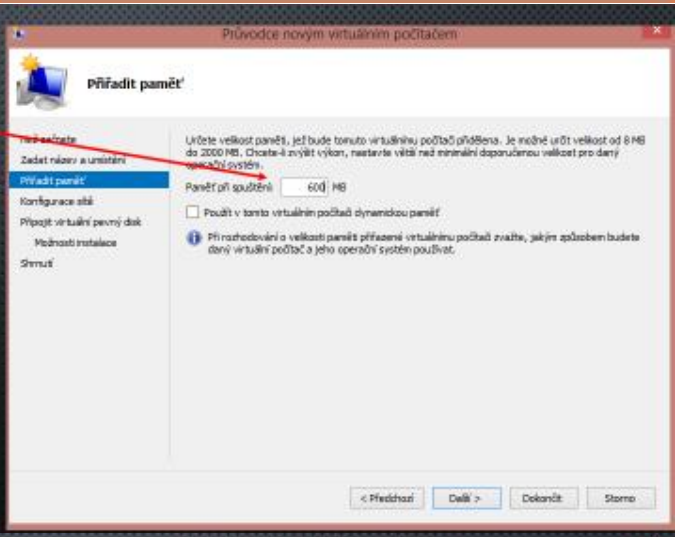
- V nabídce nová zvolíme virtuální počítač a potvrdíme šipkou další



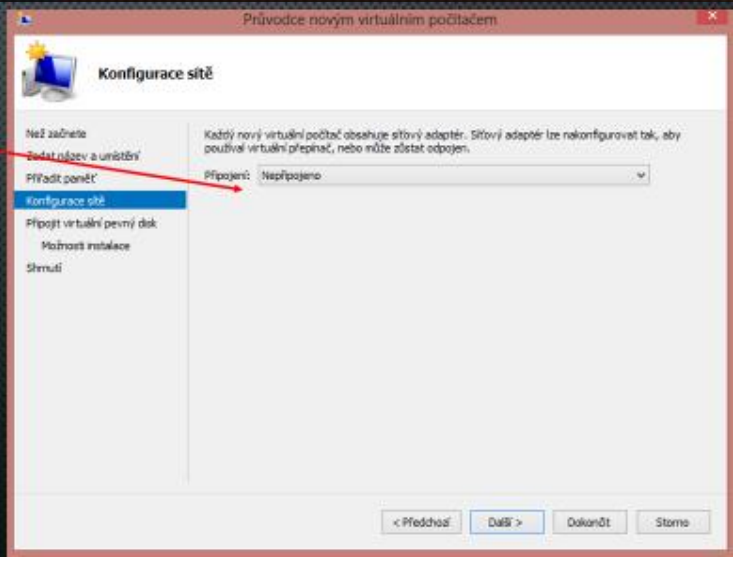
- Vyplníme název počítače a pokud máme na disku C dostatek místa (desítky GB), tak nic dalšího neměníme. Vše stvrdíme tlačítkem další.



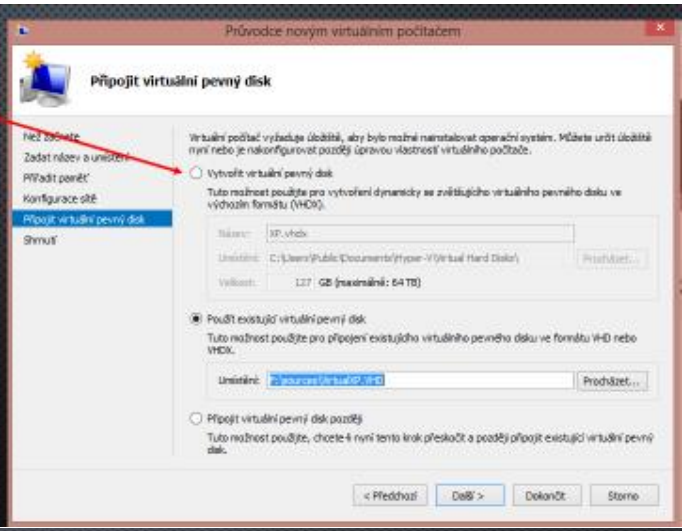
- Vyplníme velikost RAM paměti přidělenou při spuštění, tady je nutné znát nároky systému, který chceme virtualizovat



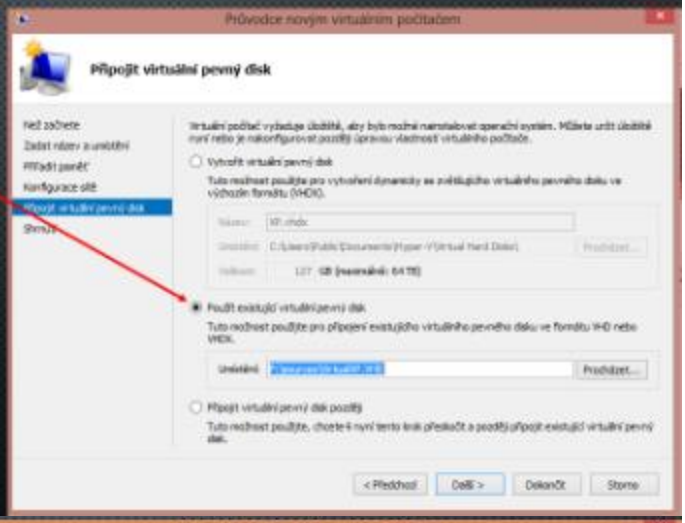
- Zvolíme připojení PC k internetu z nabídky (připojení nemusí být nabídnuto, pokud vytváříme úplně první PC)



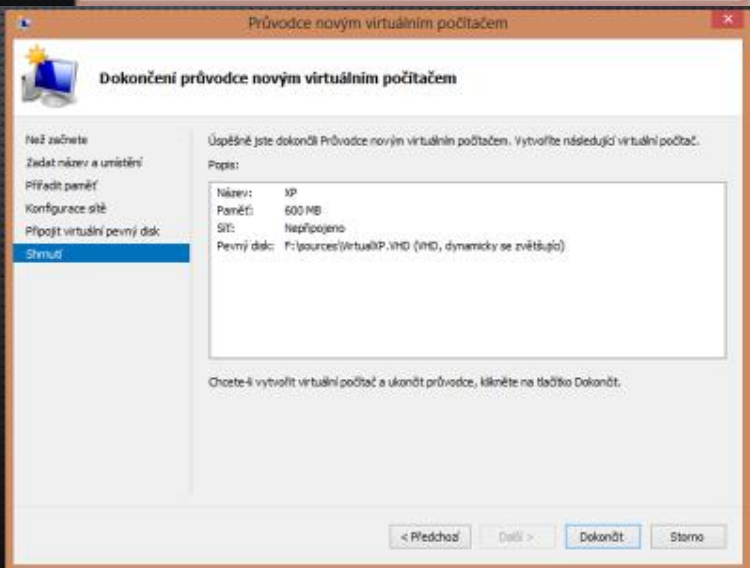
- Vytvoříme virtuální harddisk, nebo přiřadíme již existující disk se systémem či bez něj. I zde je nutné znát nároky sytému, který chceme virtualizovat



- Pokud chceme spouštět bitovou kopii systému Windows 7/8/8.1, tak zvolíme použít existující disk a médiu kde máme zálohu provedeme cestu: "WindowsImageBackup\“jméno počítače“\Backup „název zálohy“*.vhdx„ systém Windows 7 má konec: *.vhd

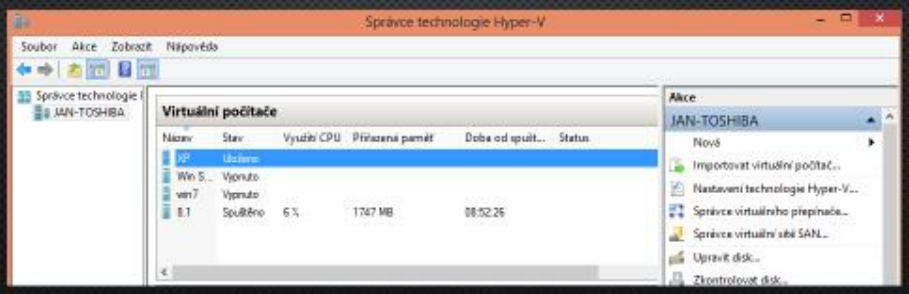


- Po kontrole zadání v souhrnném výpisu dáme dokončit

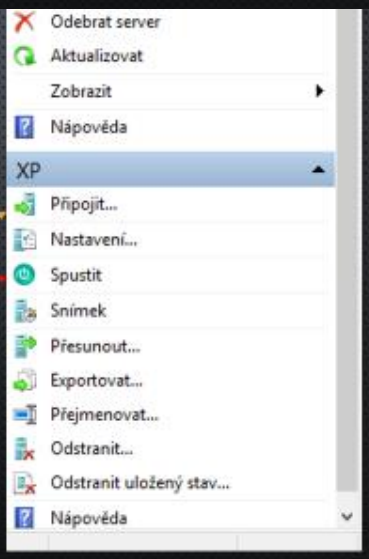


První spuštění

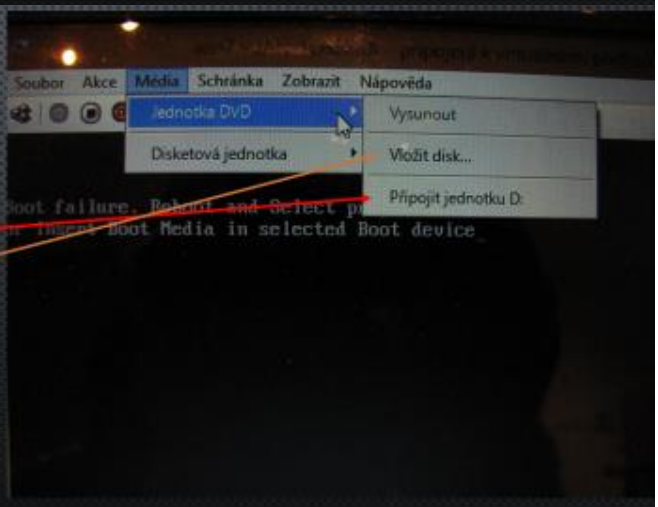
- ve správci klikneme na námi vytvořený počítač (označí se)



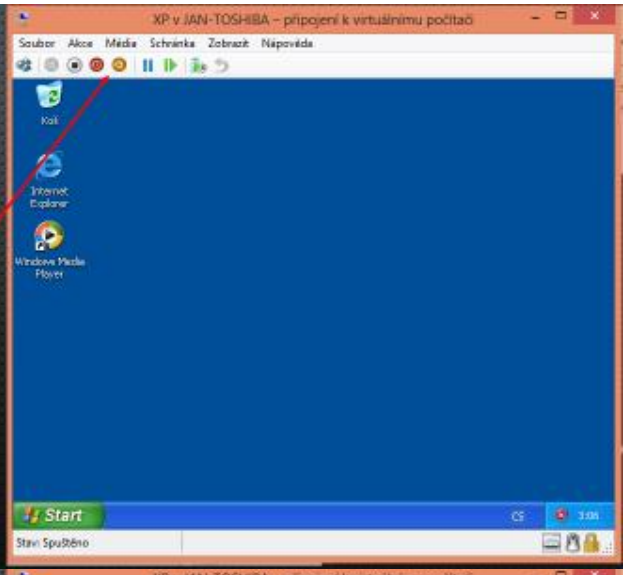
- V pravém sloupci (spodní polovině) zvolíme položku spustit.
- Po objevení nenulové doby od spuštění u názvu PC klineme na položku připojit



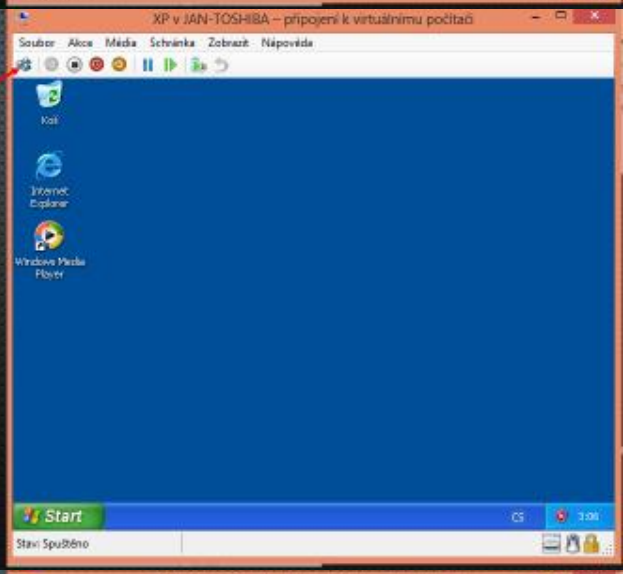
- Pokud jsme nenastavili při vytváření, či jsme zadali cestu na USB, tak nastavíme instalační médium
- Fyzický disk DVD
- Otisk formátu ISO
- Tento krok neprovádíme, pokud spouštíme bitovou kopii, pro nás práce končí předchozím krokem



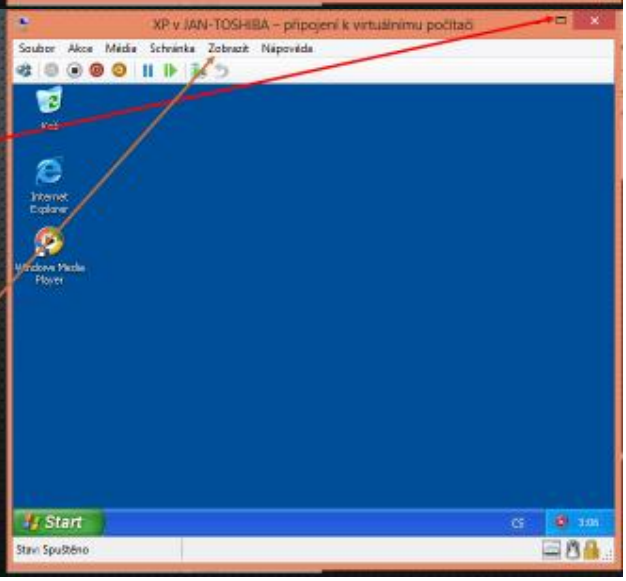
- Dokončíme instalaci OS a můžeme pracovat
- Po dokončení můžeme „hibernovat“ i systémy, které to neumí, klikneme na oranžové kolečko uložit na liště pod nabídkou média
- Možnost uložit je i ve správci

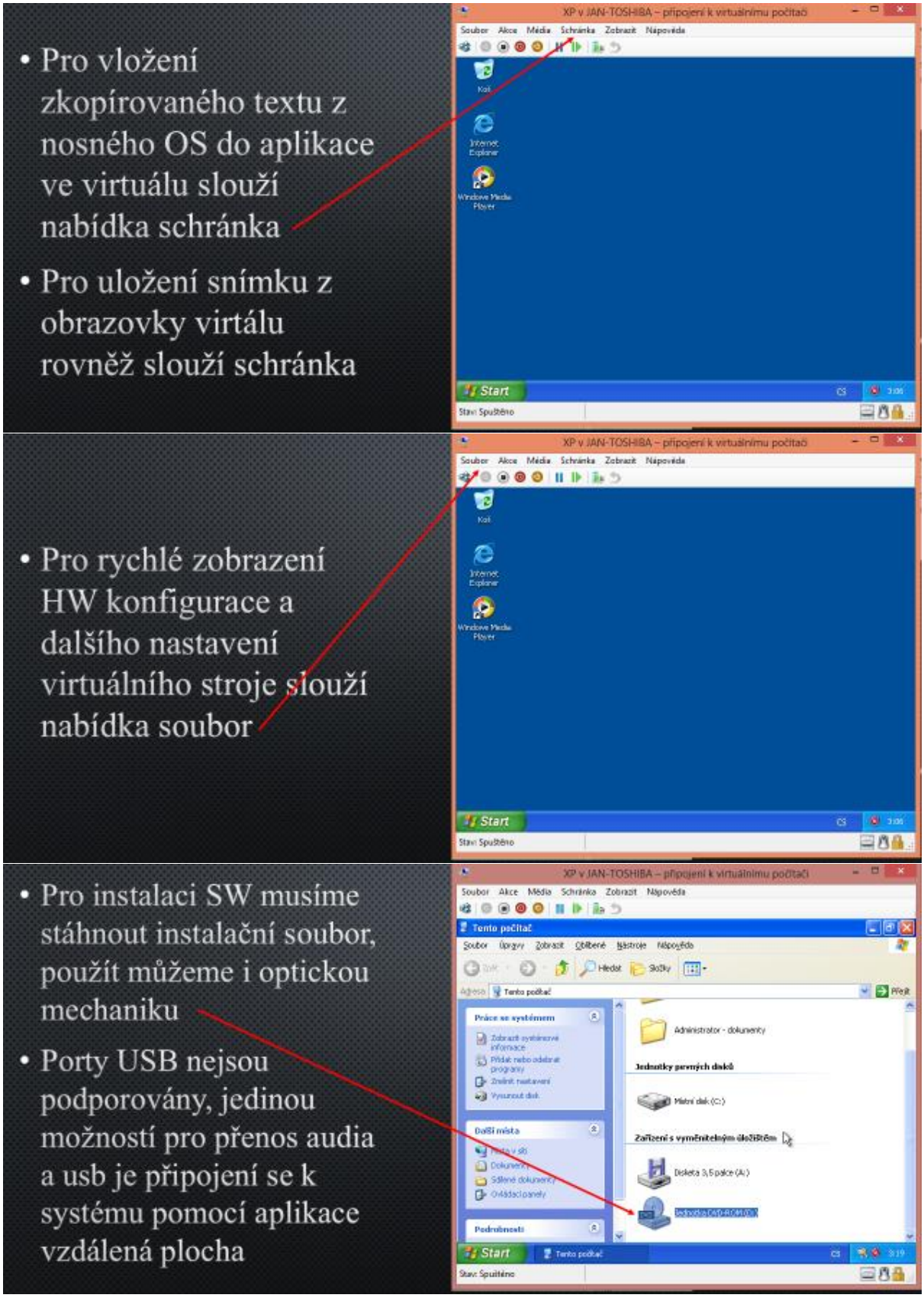


- Pro vložení zkratky ctrl+alt+dele užijeme symbol (klávesnicí bychom to zadali do nosného systému)



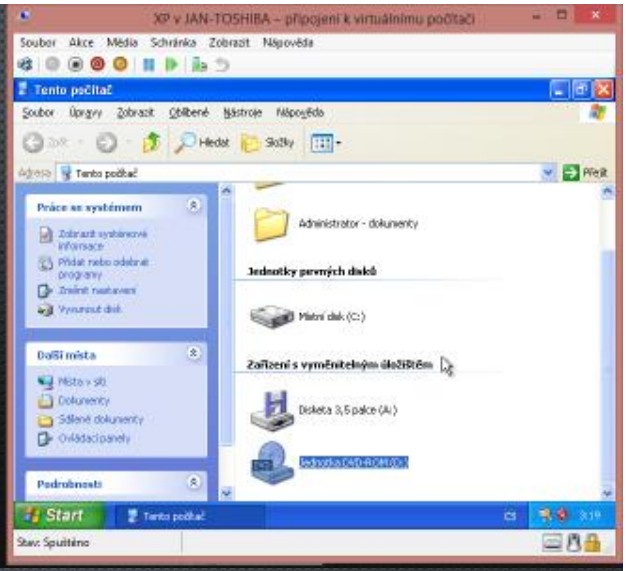
- Pro zobrazení přes celou obrazovku zvolíme celá obrazovka na rámu okna (pokud je stejné rozlišení zobrazení ve virtualizovaném a nosného OS)
- Druhá možnost je v nabídce zobrazit





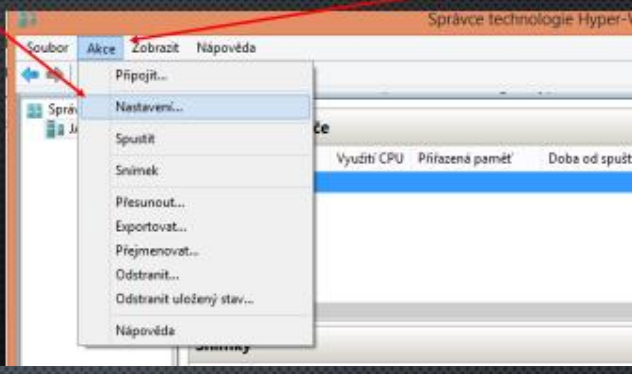
Zde se sluší dodat, že USB porty jsou již podporovány od Windows 8.1, ale uvnitř virtuálního PC musí být minimálně Windows 8, pak je možné použít rozšířenou relaci pro připojení k virtuálnímu počítači a zde je možné sdílet i USB zařízení.

- Tiskárnu není nutné instalovat, nalezneme ji jako síťové zařízení
- Na nosném OS musí mít tiskárna povoleno sdílení

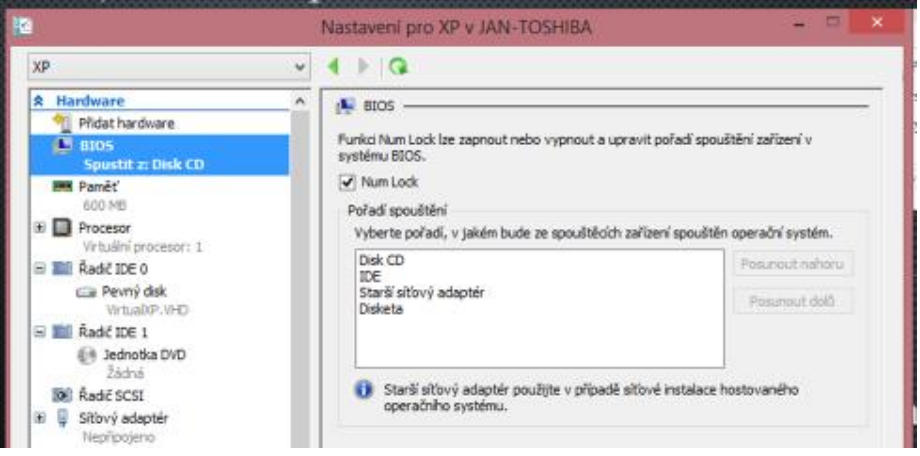


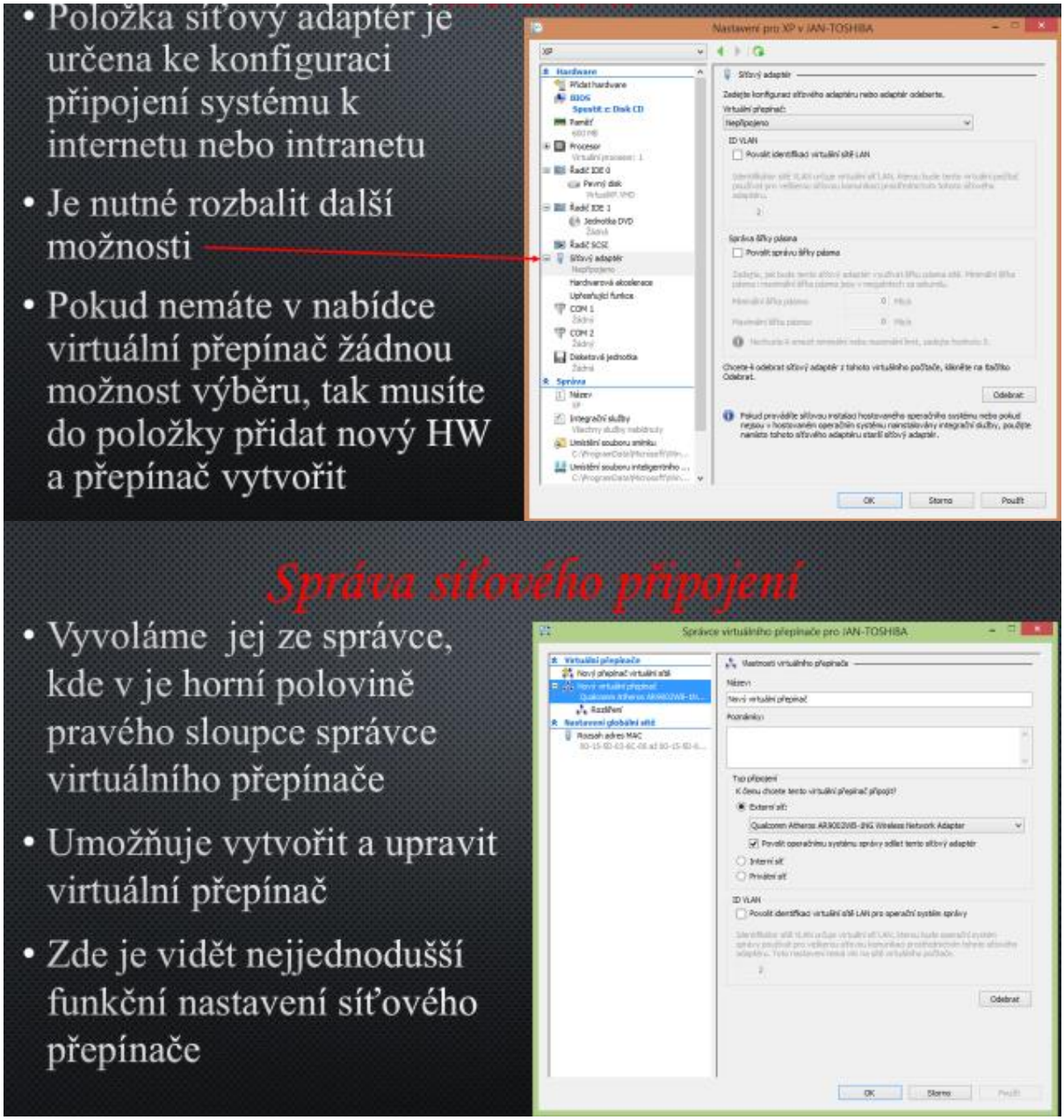
nastavení

- Počítač musí být ve stavu vypnut, jinak je možné nastavení jen zobrazit
- Přejdeme do správce, klikneme na počítač a na kartě akce vybereme položku nastavení



- Položka BIOS umožňuje nastavit klávesnici (aktivní nebo neaktivní numlock) a nastavit pořadí boot





Zbývající možnosti snadno objevíte sami, nebo s využitím návodu, který je na mém webu (http:elektrotechnika.vesele.info).

2.5.2. Proč Virtualizovat?

Pro běžného člověka na první pohled nemá smysl virtualizovat, ale zdání klame. Je možné pro ulžení souboru Virtuálního disku užít i externí zařízení jako je USB HDD nebo USB Flash o dostatečné kapacitě. Hlavní výhoda se skrývá v tom, že velmi snadno oddělíte problémové aplikace, užívání virtuální systému k prohlížení interntu, rizikovému stahování a instalaci rizikových aplikací, zvyšuje ochranu vašich dat, protože v případě kolapsu virtuálního systému přijdete jen o data v tom virtuálním souboru a ještě ani to ne, pokud je mateřským systémem Windows 8 nebo 8.1. tak lze dvojklikem připojit tento soubor jako virtuální jednotku a data si zkopírovat, takže vlastně o nic nemůžete přijít. Pokud užíváme hodně citlivá data, tak je vhodné užít odkaz na síťovou jednotku ve virtuálním systému, tou virtuální jednotkou je fizický disk a data uložit tam.

Rychlost virtuálního systému závisí na zatížení mateřského systém a také na rychlosti připojení média, které nese soubor virtuálního disku. Virtuální systém si také můžeme nosit s sebou na USB klíčence a máme k dispozici vše od dat přes aplikace až po nastavení uvnitř virtuálního systému, čímž předcházíme problémů s kompatibilitou programů a soubrů v nich vytvořených. Jedinou podmínkou jak si vzít počítač na USB s sebou je, aby hostitelský počítač měl prostedí Hyper-V, nebo (pokud máme soubor VHD) jiný virtualizační program (MS Virtual PC).

Dalším důvodem prč virtualizovat je ochrana vašeho fyzického počítače, protože i závažné chyby vzniklé ve virtuálním systému nemohou poškodit fyzický systém ani fyzický HW, právě díky Hypervizorovi.

A poslední výhodou je možnost užívat serverový OS na domácím počítači či noteboku, což je výhoda spíše pro zvědavce, či správce sítí a vývojáře.

3. Operační systémy

V úvodu si musíme uvědomit, co je to ten operační systém. Pro naše potřeby bude stačit stručná charakteristika, tedy OS je soubor systémových programů, které uživateli umožňují spouštět aplikační programy. Je tvořen sadou samostatných dílů (např.: jádrem, knihovnami, nebo příkazových interpretů...) a jde o rozhraní, jehož prostřednictvím uživatel komunikuje s hardwarem. Hodně zjednodušeně by se také dalo říci, že hardware je tělem a OS duší počítače.

3.1. Rozdělení

OS můžeme dělit do těchto základních skupin:

1. Počet úloh: jednoúlohové, multitaskové
2. počet uživatelů: jednouživatelské, víceuživatelské
3. účel použití: univerzální, RT-systémy, databázové systémy, síťové

Na svých počítačích, ať máme Windows nebo MacOS (Apple) nebo Linux, tak máme systém multitáskový, universální jednouživatelský. Můžete mi namítnout (pravda u Windows 7, 8 a 8.1 to již není úplná pravda), že máte na počítači více uživatelských účtů. Více účtů tedy logicky znamená více uživatelů, problém je v tom, že reálně v jednu chvíli může pracovat jen jeden z uživatelů, proto domácí systémy obecně považujeme za jednouživatelské.

3.2. Obecná struktura

V bodech je možné strukturo OS popsat následovně:

1. řídící program (executive)
2. příkazový procesor (command processor)
3. řídící V/V programy
4. obsluha souborů
5. pomocné programy (utilities)

Ke struktuře, kterou máme uvedenou víše, musíme ještě připočítat zavaděč systému, který ovšem většinou není součástí systému.

3.3. Terminologie

Jak už to tak je v těchto nudných základech, které musíme znát, abychom mohli nějak pracovat, musíme si před další cestou za poznáním vysvětlit několik termínů, protože bychom si později nerozuměli. Neříkám, že by ta situace nastala hned, ale určitě se tak stalo. Tak pojďme na to, snažil jsem se být stručný a psát spíše heslovitě.

Přerušení - způsobeno událostí vně procesu, dochází k němu mezi dvěma instrukcemi, obslouženo příslušným obslužným programem, startovací adresa dána přerušovacím vektorem

Výjimka - chybový stav způsobený událostí uvnitř procesu (dělení nulou, adresování zakázané paměti, …), během provádění instrukce, po ošetření snaha o restartování instrukce

Maskování přerušení - zákaz přerušení, zpravidla nižších priorit

"Holý počítač" - základní hardwarové vybavení

Virtuální počítač - "holý počítač", na kterém běží jádro OS

Uživatel - ten, kdo předkládá úlohu výpočetnímu systému

Úloha - souhrn činností potřebných k provedení zadání od uživatele (dělí se na kroky, provádí se sekvenčně)

Proces - vlastní provedení kroku úlohy

Vlákno (thread) - větev procesu, společné systémové prostředky (paměť, soubory, globální proměnné, …), vlastní zásobník

Zásobník (stack) - paměťový prostor pro předávání parametrů, adres, obsahu registrů, lokální proměnné, …)

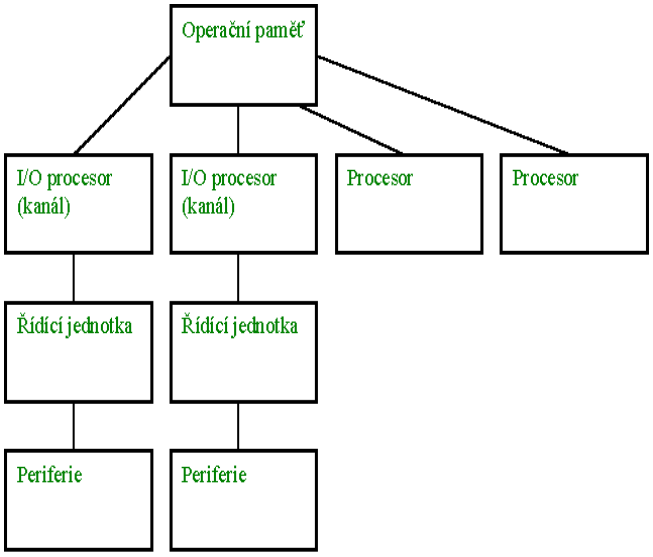
Adresový prostor - paměťový prostor procesu, který se musí mapovat do fyzické paměti

Režim jádra - zpracování procesu, kdy se volají služby jádra OS, dochází k němu i při práci v uživatelském režimu (ošetření přerušení, plánování procesů, správa paměti, …), procesy přistupují k adresám jádra i uživatele

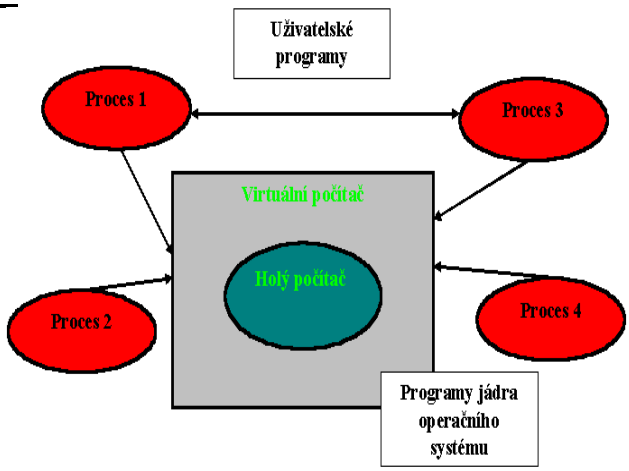
Uživatelský režim - procesy přistupují ke svým datům a instrukcím, nikoliv k adresám jádra nebo jiných procesů

Privilegované instrukce - Některé instrukce by při nevhodném použití mohly vést ke zhroucení operačního systému, k chybám, nebo k narušení bezpečnosti. Jedná se o I/O instrukce a některé instrukce řízení procesoru. Aby se znemožnilo jejich použití uživatelskými programy, má většina procesorů dva nebo více stavů s různým stupněm privilegií. Příklad dva stavy: privilegovaný a uživatelský. Po startu je procesor v privilegovaném stavu a je spuštěn OS. V privilegovaném stavu je dovoleno provádět i privilegované instrukce, běží v něm operační systém. V uživatelském stavu pokus o provedení privilegované instrukce skončí chybou, používají jej uživatelské programy. Při vyvolání služeb operačního systému přechází procesor do privilegovaného stavu, při návratu ze systémové služby přechází do uživatelského stavu.

Holý počítač –



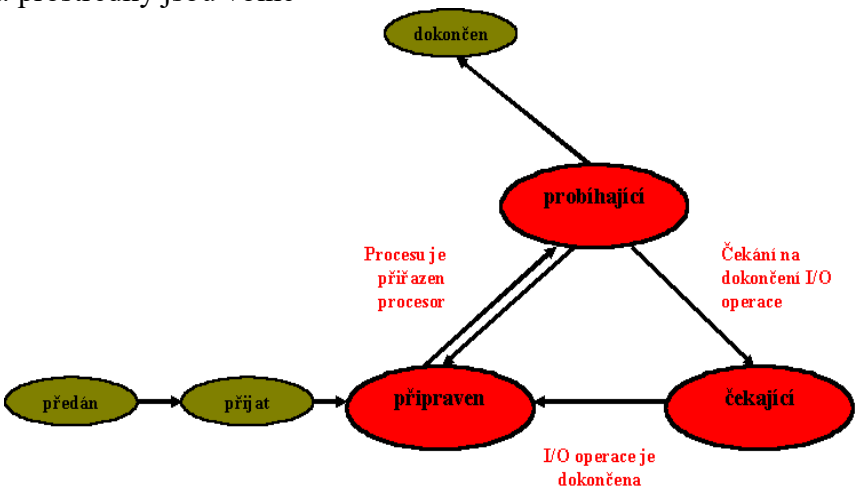
Virtuální počítač –



Uváznutí (deadlock) - Pokud je ve víceúlohovém systému jednomu procesu přidělena tiskárna a druhému magnetická páska a potom první z procesů požádá o přidělení pásky a druhý tiskárny, žádný z těchto procesů nemůže pokračovat v běhu. Této situaci říkáme uváznutí neboli deadlock. Řešením této situace je odebrat jednomu z procesů prostředek, který požaduje druhý proces. Většina operačních systémů však násilné odebrání prostředků nedovoluje. Uváznutí (deadlock) - je situace, kdy dva nebo více procesů čekají na událost, ke které by mohlo dojít pouze pokud by jeden z těchto procesů pokračoval (vyřešení dopravní situace couváním).

3.4. Stavy procesoru

- probíhající: je přidělen procesor
- čekající: čekání na určitou událost
- připraven: čeká na přidělení procesoru
- předán: očekává se reakce OS na předání úlohy
- přijat: úloha převedena do vnitřního tvaru, procesům nejsou přiděleny žádné prostředky
- dokončen: prostředky jsou volné



3.5. Operační paměť

Správce OP - mezi nejdůležitější, ne však nejkomplikovanější, moduly, umožňuje efektivní práci ostatním prvkům, má smysl pouze v multitaskovém prostředí.

-přidělovat OP procesům
-zařazovat uvolněnou OP
-odebírat OP
-zajišťovat ochranu OP

3.5.1. Přidělování OP po blocích

Strategie pro jednoduché OS nebo jednorázové a krátkodobé úkoly, proces musí vědět, kolik OP potřebuje a explicitně si ji vyžádat, požadavek je buď akceptován, nebo odmítnut, je třeba udržovat informace o vlastnících bloků, o volných blocích (tabulky s informacemi), potřebná znalost délky bloku a vlastníka, adresa dalšího bloku nemusí být explicitně uvedena (bloky navazují).

Informace o blocích v datových strukturách správce, potřeba další paměti a zvýšení režie, počet bloků předem neznáme, řešením je přidělení většího bloku, do volného místa uložit informace o samotném bloku i o adrese následujícího. Nevýhodou je promíchání systémových informací mezi paměťové bloky. Při nedokonalé ochraně paměti hrozí zhroucení OS.

Hlavní nevýhoda - fragmentace paměti, správce dokáže spojit dva volné bloky, někdy je využitelný pouze největší blok, celková volná OP může být několikanásobně větší. Fragmentaci do jisté míry můžeme ovlivnit alokační strategií.

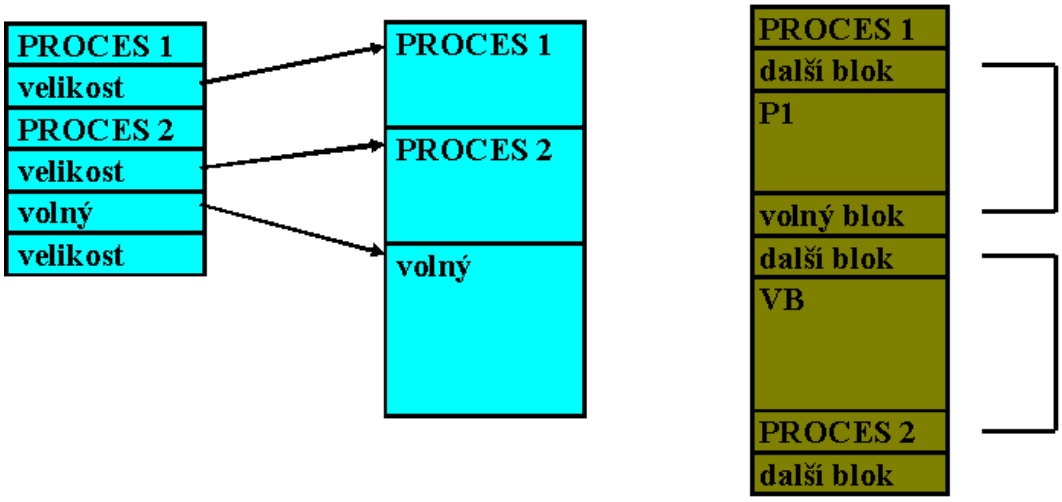
Fragmentace - důsledek přidělování nestejně velkých bloků paměti, pro úplné zamezení - přidělování stejně velikých bloků, uvolňování v opačném pořadí než se alokuje, nepřijatelný diktát chování procesů, některé procesy si OP potřebují držet stále, drasticky by se snižovala průchodnost OS, musí být vnitřní věcí správce OP.

First fit - jednoduchá, rychlá, nemá vliv na fragmentaci

Best fit - mírně omezuje fragmentaci, větší režie

Last fit - někdy výhodné obsazovat OP odzadu (stack)

Přesunování bloků (defragmentace paměti) alokovaných bloků za sebe, rychlostně zcela zvládnuté pomocí blitteru, nepoužívá se zcela rutinně, nutnost zachování transparentnosti pro procesy - požadavek na překlad adres, pokud je k dispozici, není důvod nepoužít virtualizaci - virtuální paměť. Popis defragmentace je vidět na následujícím obrázku.



3.5.2. Virtuální paměť

Přemapování paměti, zajišťuje transparentnost paměti pro procesy, virtualizace je nejdokonalejší strategie správy paměti, musí být k dispozici jednotka řízení paměti (bývá integrována).

3.5.2.1. Virtualizace adres

- jednotka řízení paměti, využití nejen pro virtualizaci paměti, pomocí stránkování lze rozšířit virtuálně paměť na celý adresový prostor, musí se v něm srovnat všechny procesy, nejjednodušší řešení pro OS.
- každý proces svůj adresový prostor- řešením je zvláštní tabulka stránek pro každý proces, využívá se mechanismu stránkování na žádost, práci tím na sebe přebírá správce virtuální paměti, přiřazení mezi logickými a fyzickými stránkami se vytváří dynamicky, stejné jako stránkování na žádost pro jediný proces
- je potřeba zajistit výměnu tabulek, které popisují stránky - upozorňuje správce procesů jedná se o virtualizaci adres - každému procesu se umožní využívat vlastní adresový prostor

3.5.3. Ochrana paměti

Žádný proces by neměl narušit adresový prostor jiného procesu nebo dokonce OS. Vv multitaskovém prostředí je tento požadavek jedním z nejdůležitějších, proto je nutná jednotka řízení paměti, která nabízí tři mechanismy. Paměť, která není dosažitelná nemůže být poškozena, Jedná se o vedlejší efekt virtualizace (každý proces má jiné stránkové tabulky). Ochrana sdílených dat, jedná se o knihovny, které jsou sdílené více procesy, jednotka řízení paměti má k dispozici prostředky, kterými označí stránku pouze pro čtení (1 bit), v případě snahy o modifikaci dojde k vyvolání výjimky a zabití procesu. Ochrana systémových dat před procesy, procesor rozlišuje běh v režimu jádra OS a režimu uživatelském, jednotka řízení paměti nabízí prostředky pro označení stránek pro běžný proces a pro OS, při přístupu do paměti se dělá komparace, v jakém režimu se nachází procesor, při porušení se generuje přerušení - mohou být dvě skupiny tabulek stránek (Motorola).

3.6. Obsluha požadavků na přístup k HDD

Při vyřizování požadavku OS na disk je potřeba nejprve vystavit hlavy na příslušnou stopu a pak počkat, až bude požadovaný sektor pod hlavami. U víceúlohových systémů mohou přicházet požadavky na disk rychleji, než je možné je vyřizovat. Vyřizování požadavků v pořadí, jak přicházejí (tzv. FIFO nebo FCFS - First In First Out, First Come First Serve), není optimální. Používají se jiné algoritmy (oba lze modifikovat v případě, že existuje rychlý způsob, jak se dostat na stopu 0):

SSF (Shortest Seek First) - máme-li hlavu na 50 cylindru a přijdou požadavky na 52, 80, 7, 49, 45 -> SSF -> bude vyřízeno v pořadí 50, 49, 52, 45, 80, 7 dochází k diskriminaci okrajových cylindrů.

Elevátorový algoritmus - totéž co automatické výtahy (nahoru -> dolů) pohyb hlavy je změněn na 50, 52, 80, 49, 45, 7.

3.7. Časovač

Časovač umožňuje plánovat události -> proces může vyvolat službu OS, která jej zablokuje ("uspí") do uplynutí požadovaného času. Též se používá pro některé systémové akce (přidělování časových kvant, zastavování motorů disketových mechanik apod.).

Není možné předpokládat, že počítač má dostatek hardwarových časovačů pro všechny účely. Zpravidla je k dispozici jen jeden časovač a SW (programová) časová fronta. Hardwarový časovač se pak používá pro odměření času do nejbližší události ve frontě. Do fronty jsou události řazeny tím způsobem, že se zjistí po které události má k nově přidávané události dojít a spočítá se a zaznamená čas (který má uplynout mezi těmito dvěma událostmi). Při vynulování časovače se vybere první událost z fronty, provede se příslušná akce a hardwarový časovač se přeprogramuje na čas následující události.

3.8. Něco málo z historie

První OS vznikly začátkem 70. let kdy nejprve firma IBM, která v té době měla téměř monopol na sálové počítače (a jiné v té době nebyly). Roku 1960 však firma Digital Equipment Corporation (DEC) vytvořila první komerční počítač vybavený obrazovkou a klávesnicí (tzv. obrazovkovým terminálem). Se vznikem minipočítačů v polovině 60. let, což byly menší stroje, které nevyžadovaly tak specializovanou obsluhu jako mainframe a byly rovněž vybaveny obrazovkovým terminálem, vyvstala potřeba operačních systémů tak, jak je známe dnes.

Všechny systémy uvedené v přehledu (viz dále) jsou určeny pro sálové počítače. Do roku 1981 jsme neznali standartní OS, ale vše fungovalo na bázi jazyka Basic. Do dnešních dnů se nám skrytě dochoval UNIX (tvoří jádro Linux a MacOS).

název	Rok	vydavatel
VMS	70. léta	DEC
UNIX	70. léta	AT&T
BSD	2./2 70. let	platforma AT&T, dokončená universitami USA
IBM AIX, HP-UX, SGI IRIX, Cray Unicos, Sun Solaris	Přelom 70 a 80. let	University USA a další počítačové firmy na platformě UNIX

3.8.1. CP/M

Tento systém zavedl některé podstatné věci, bez kterých by nebyl možný rozvoj Windows, Linux ani Mac OS. Lze jej zařadit do rodiny systémů, které stály u zrodu Linux, protože užívají podobné jádro, ale díky povaze MS DOS i Windows. Vznikl na přelomu 70. a 80. let jako drastická simplifikace UNIXU. Zaváděl konvenci osmiznakového jména souboru a trojznakové přípony spolu s označováním diskových jednotek písmeny.



3.8.2. MS-DOS

MS-DOS nebyl skutečným produktem Microsoftu. V zásadě se jednalo o mírně upravený CP/M, jehož původní verze dokonce na prvních PC fungovala. Proti MS-DOSU skrýval Unix výhodu v tom, že dokázal obsluhovat více uživatelů a programů v reálném čase. MS-DOS komunikuje prostřednictvím příkazového řádku. Firma IBM, která byla největším odběratelem, nebyla spokojena se systémem MS-DOS a proto začala vyvíjet vlastní PC-DOS, který se lišil jen v maličkostech. Ona i spousta dalších firem vyvíjela Dosy z niž nejvýznamnější byl DR-DOS. Ve stejné době firma Apple vyvinula systém s grafickým uživatelským rozhraním (podobná platforma jako Windows 1.0) a byl celý 32 bitový, narozdíl od 16 bitového PC.

MS-DOS byl již v době svého vzniku velice nepohodlný a navíc i z hlediska návrhu nedostatečný a také zastaralý. Podporoval jenom jednoho uživatele, který mohl v reálném čase

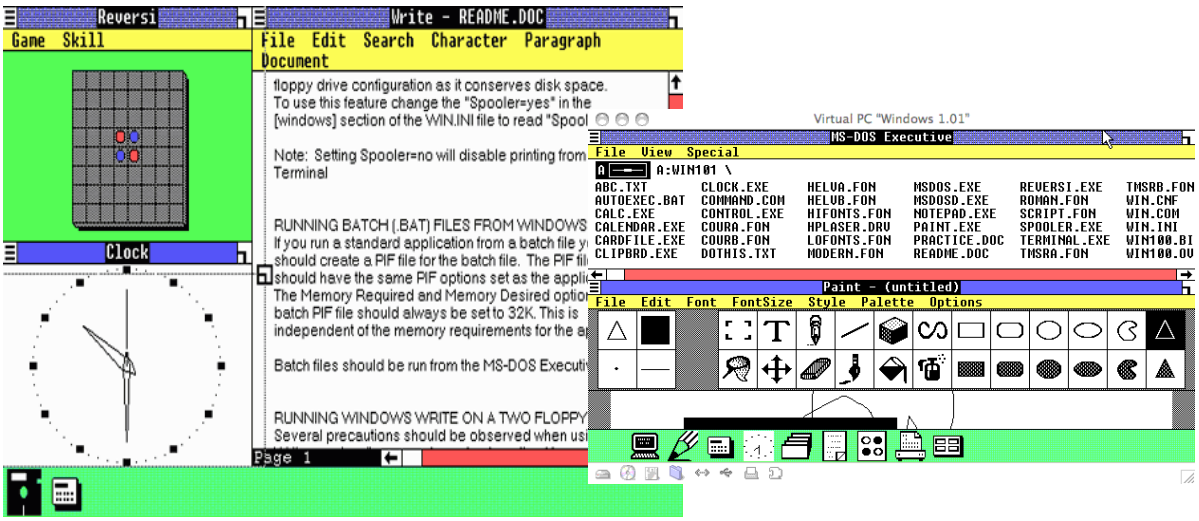
pracovat s jedním programem a nedokázal pracovat s pamětí vyšší než 640 kB nebo s disky většími než 30 MB.

3.8.3. Rodina Windows obecně

Ještě před nástupem Windows zaznamenalo úspěch rozhraní firmy hp, které bylo graficky dost podobné, ale bylo převálcováno Windows na plné čáře. Vydavatelem systému je společnost Microsoft Corporation, která se stará o celý vývoj i podporu systému. Windows je nástupcem MS-DOSU, který byl až do verze Me integrován v jádře OS. Windows je jeden z nejrozšířenějších OS na světě a v Evropě tvoří více jak 85% OS instalovaných na uživatelských počítačích a s vyjímkou webových serverů tvoří téměř 50% serverových OS.

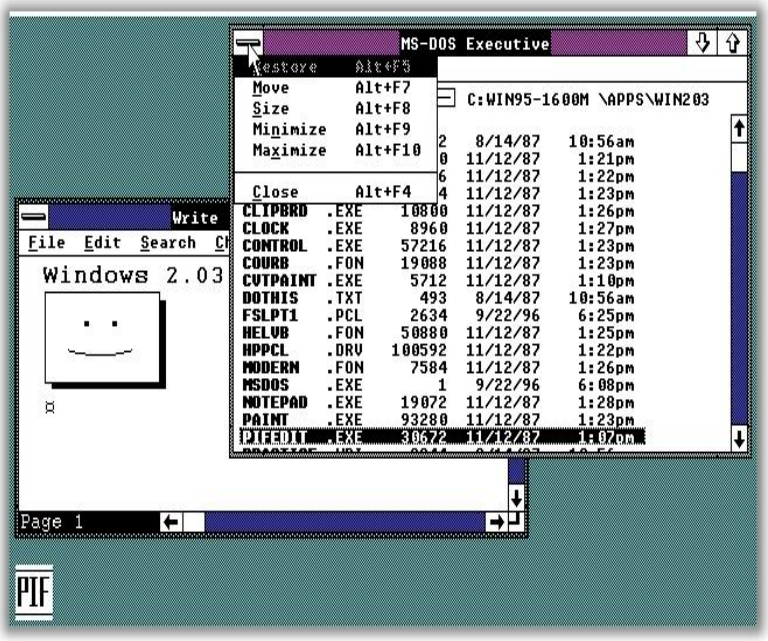
Všechny systémy jsou v určitém režimu kompatibilní se svými předchůdci až po MS-DOS (po MS-DOS je to po Windows Me). Systémy Windows podporují architektury 16, 32 a 64 bitů, chystá se podpora 128 bitů (mám na mysli CPU a základové desky). Jednu část DOSu dědí všechny Windows bez rozdílu, tou je přípona .exe u instalačních a aplikačních souborů. Již od počátků byli Windows multitaskingový systém.

3.8.4. Windows 1.0



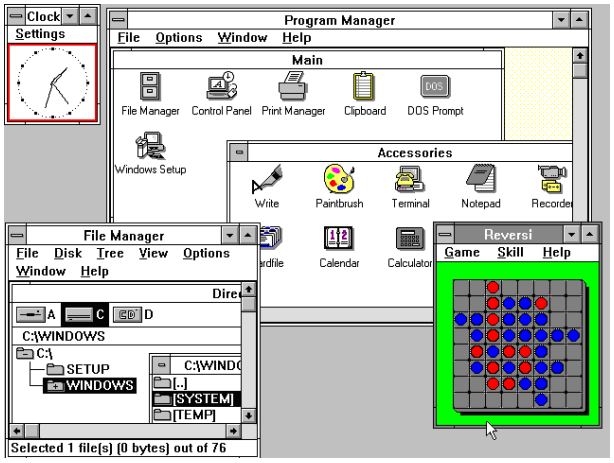
Systém byl vydán 20.11.1985, jako reakce na novou verzi MacOS instalovaný společností Apple v novém počítači Lisa. Byl plný chyb a přežil díky masivní marketingové kampani. Nové ovladače byly především grafické, na myši, klávesnice, tiskárny atd. Grafické rozhraní obdobně jako u Aplle pocházelo ze zamítnutého projektu společnosti Xerox a bylo zdrojem mnoha sporů mezi Apple a Microsoft. Okna jsou řazeny do formy a neuměla se překrývat. Ukončením správce oken se mohl člověk vrátit do MS DOS. Celý systém měl architekturu 16 bit. Také obsahovaly velké množství vlastních ovladačů pro různá zařízení.

3.8.5. Windows 2.0



Požadavky systému: MS-DOS 3.0 nebo vyšší, 512K paměti nebo více, grafická karta, disketová mechanika a pevný disk. Propracovanější klávesové zkratky (potržené písmeno v nabídkách). Vývojářská podpora se podstatně zvýšila s touto verzí. Windows byly vydány 23.11.1987. Nově umožňují překrývat jednotlivá okna aplikací přes sebe a objevují se První verze Microsoft Word a Microsoft Excel. Tato verze také představila manipulaci s okny jako je minimalizovat a maximalizovat.

3.8.6. Windows 3.1



Systém podporoval české jazykové nástroje. 1. verze Windows 3.0 byla vydána roku 1990.Správce programů umožňoval vytvářet ikony programů, seskupovat je, vyvářet textová menu atd., jeho ukončení znamenalo návrat do dosu. Chyba jediné aplikace v kooperativním multitaskingu mohla zblokovat chod celého systému, tato bolest provázela všechny 16 bitové Windows.

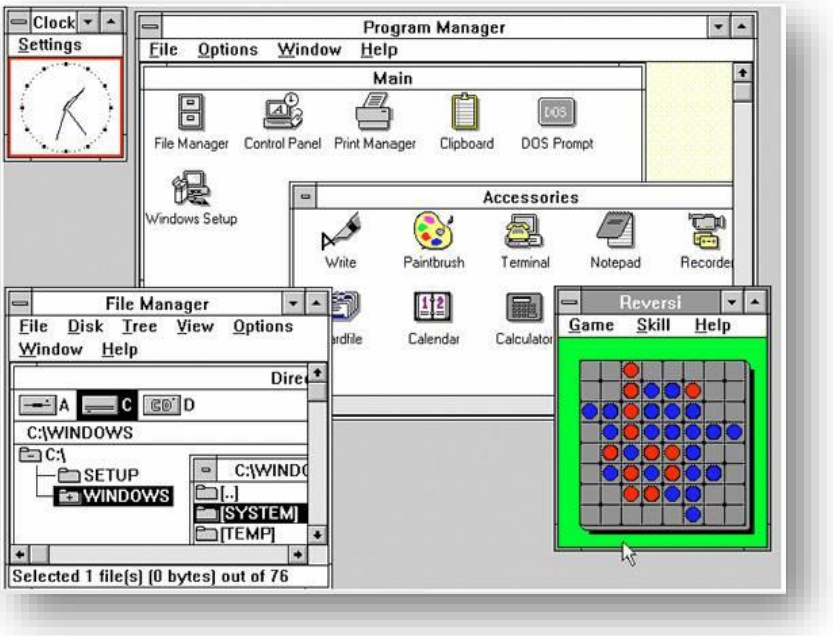
Programové vybavení: Správce souborů (winfile.exe) umožňoval správu souborů a adresářů (podobal se dnešnímu Průzkumníku – každé okno obsahovalo dvě části – strom složek a obsah aktuální složky). Kopírování probíhalo přes přes schránku, bylo podporováno filtrování. Windows 3.1x obsahovaly různé základní aplikace: Poznámkový blok (notepad.exe), Write (write.exe, dnešní WordPad), Malování (Paintbrush, pbrush.exe) atp. Obsahovaly také

Windows Media Player (uměl přehrát WAV, MIDI a AVI), Přehrávač CD a Záznam zvuku (ten je stejný dodnes, dříve sndrec.exe, v např. WinXP sndrec32.exe). Nástroj Ovládací panel (control.exe) sloužil ke konfiguraci Windows. Mohlo to být např. barevné schéma, nastavení myši a klávesnice, instalace ovladačů, ale také celkem dost podrobné nastavení portů a podobně.

Podpora počítačových sítí, a to nejen sítí Microsoftu, ale i těch ostatních (jako od IBM, Novellu atp.). Umožněno bylo také sdílení souborů a tiskáren nebo třeba posílání e-mailů.

3.8.7. Windows 3.11

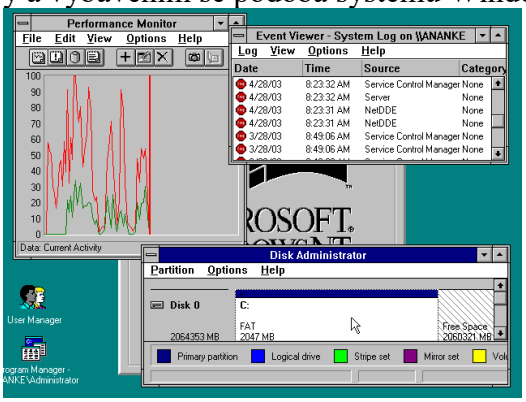
Jde o legendární systém, který teprve znamenal upevní systémů Windows na trhu a tím tehdy jen relativně dominantní pozici Microsoftu. Jde nástavbu systému Windows 3, která je zbavena chyb. Tento systém se jeko první více prosadil i v ČSR, takže si jej mohou pamatovat i vaši rodiče či prarodiče.



3.8.8. Windows NT 3.1

Jde o systém, který vyšel těsně po 3.11 a je jakousi spojnicí 3.11 a 95, která na rozdíl od obou jmenovaných systémů pracuje na zcela novém, bezdosovém jádře. Toto jádro je určeno do firem a na servery a až po Windows XP to platilo na 95 %. (pro XP již ne, protože ty na stejné technologii jádra byly určeny již i do domácností).

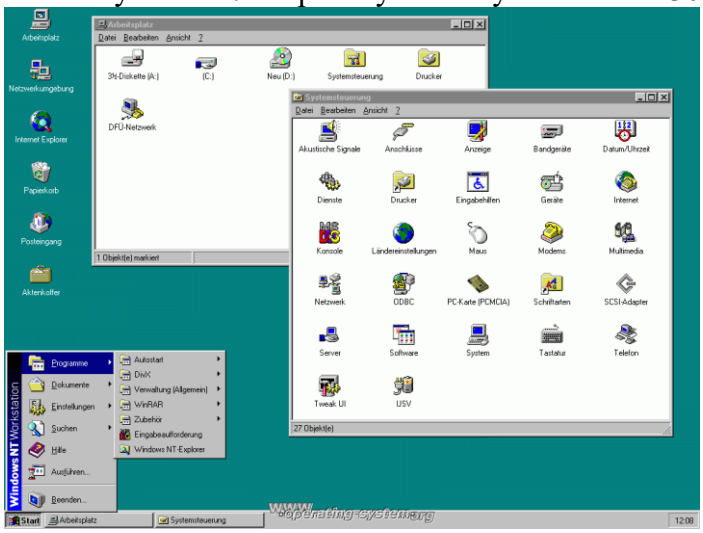
I tento OS byl 16 bitový a vybavením se podobá systému Windows 95.

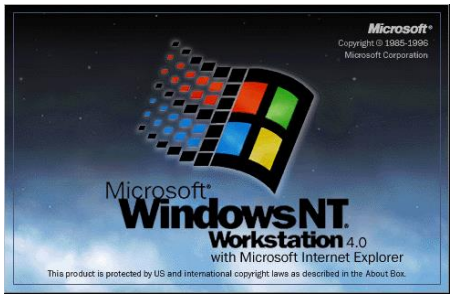


3.8.9. Windows NT 4

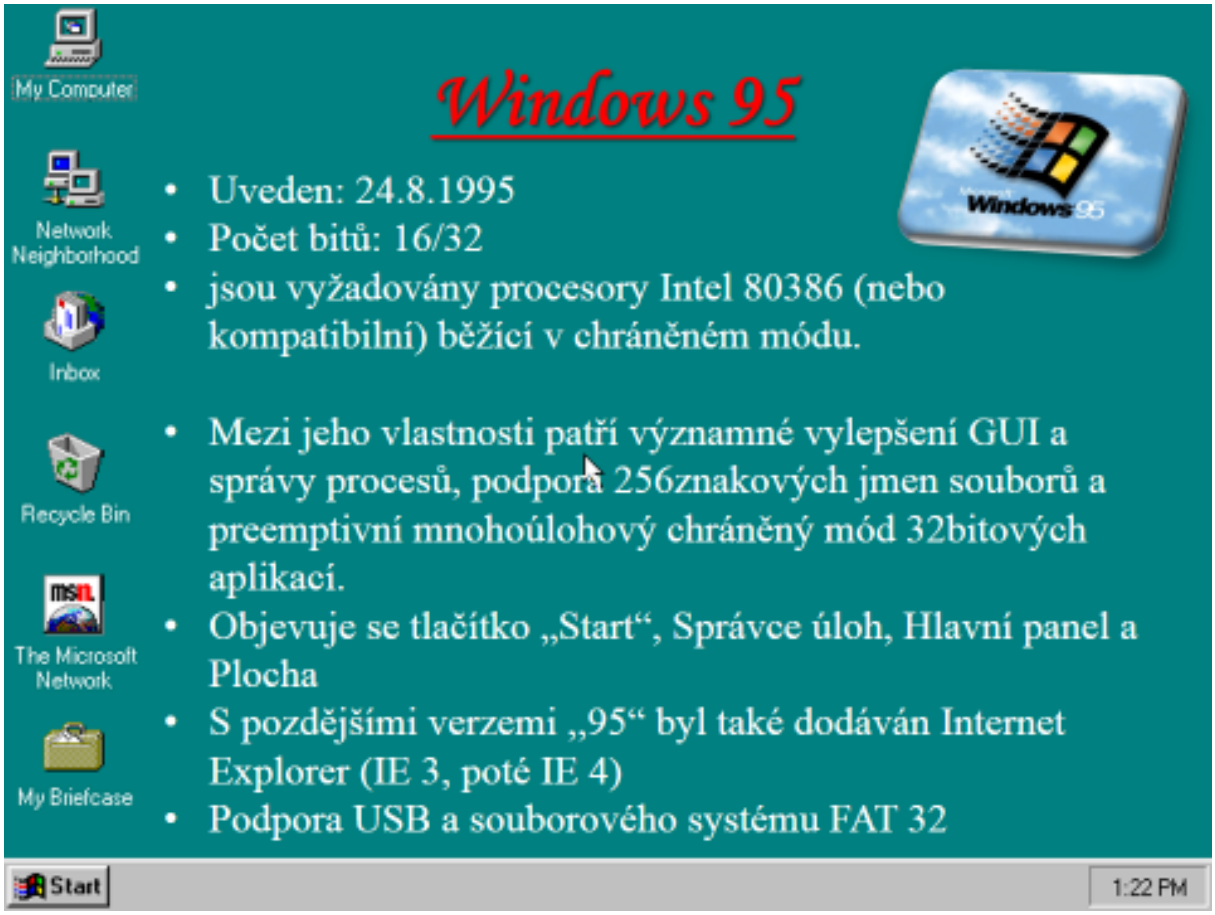
Vydáno bylo 16 servisních balíčků ten poslední s označením 16a byl vydán v roce 1999. Jde o systém, který spojuje 95 a 2000. Některé technologie a aplikace přebírají 98, které vychází na dosovém jádře o 2 roky později. Celkově se systémy NT vyvíjejí samostatně, nezávisle na systémech s dosovým jádrem.

Systém byl vydán v roce 1996. a k dispozici byla verze pro firemních počítače a několik serverových verzí. Podpora systému byla ukončena 30.6.2004





3.8.10. Windows 95



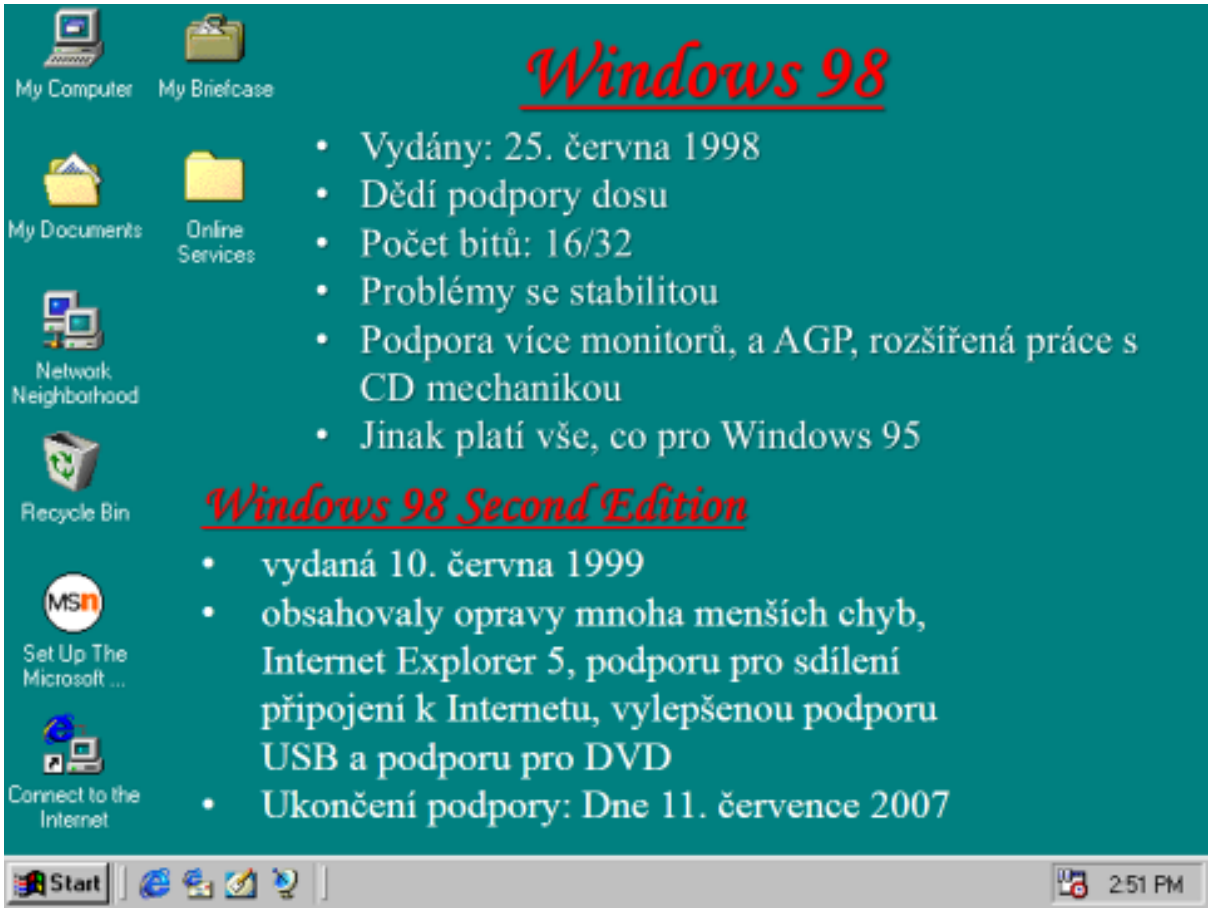
Pracovním názvem těchto Windows je Chicago. Bylo již upuštěno od podpory starších 16bitových x86 procesorů. Windows 95 používá ke spuštění MS-DOS 7.0, který je v něm obsažen, ale většinou je před uživateli ukrytý. Windows 95 se stal prvním produktem Windows pevně svázaným s určitou verzí operačního systému DOS. Přestává být starší 32bitový přístup k disku přes BIOS používán. To mimo jiné redukuje roli MS-DOS na pouhý zavaděč pro chráněný mód jádra Windows. Tímto systémem systém předbíhá Microsoft Apple a IBM, která se snaží držet při životě BeOS 5. verzí, o celé kolo díky podpoře 32 bitů a definitivně si dělá „Monopol“ na OS. (většinové postavení na trhu, nikoliv plně monopol, protože konkurence existuje ještě dnes).

Jen pro zajímavost, toto byl první systém, se kterým jsem se jako kluk setkal a dost možná jej zá spousta z vás osobně. Počítač, na kterém tento systém běžel z nostalgie používám stále, jen již v něm nejsou 95, ale 98, které se statečně drží.

3.8.10.1. Systémové požadavky

Procesor Intel 80386 kompatibilní (jakákoli rychlost)
4 MB operační paměti
50 MB místa na disku
Disketová mechanika nebo jednotka CD-ROM (pouze pro instalaci)

3.8.11. Windows 98



Systém je velmi kvalitní, řadí se spolu s Windows XP a Windows 7 k nejpovedenějším systémům společnosti Microsoft. Pro svou popularitu si uživatelé vyžádali dodatečné edice Titanic téměř 4 roky po ukončení distribuce. Osobně mohu říct, že tento systém je spolehlivý i po 16 letech používání a je takřka bezúdržbový. Již na něm pracuje MS Office 97, jehož dokumenty jsou podporovány (i vytváření) i poslední verzí Office.

3.8.11.1. Systémové požadavky

Procesor 486 DX 66Mhz nebo vyšší
minimálně 16 MB operační paměti, 24MB doporučeno
Místo na disku:
Ze systému Windows 3.1 nebo 95: 120-295 MB (obvykle 195 MB)
Nová instalace - FAT16: 165-300 MB (obvykle 225)
Nová instalace - FAT32: 140-220 MB(obvykle 175)
Windows 98 neumí obsloužit paměť nad 512 MB.

3.8.11.2. Nejvýkonnější sestava pro Windows 98SE

Základní deska s chipsetem nForce 3 Ultra se Socketem 939 (na žádné variantě nForce 4 už Windows 98 nefungují!)
Procesor AMD FX-57 (2,8 GHz)
1 GB operační paměti (jen s použitím Himemx.exe) jediné
grafická karta do slotu AGP 8x - ATI Radeon 9800 (neoficiálně až ATI Radeon X850) nebo nVidia GeForce 6800
137 GB pevný disk

3.8.12. Windows 2000 (NT5)

Systém byl vydán roku 2000 v edicích: Professional, Server, Advanced Server a Data-Center Server a podporován byl až do července 2010. systém je 32 bittový.

V prostředí je přednastavena modrá barva pozadí, poupraveny byly ikony Tento počítač, Místa v síti a Koš, položky v menu se automaticky skrývají podle jejich intenzity používání

Podpora Plug-and-play, DirectX (verze 7.0), Windows Internet Explorer 5.01 a souborového systému NTFS, vylepšeného o podporu diskových kvót. Byla také poprvé představena funkce spánku, což je uložení veškerého obsahu paměti na pevný disk a vypnutí počítače s možností opětovné reaktivace systému a pokračování ze stavu, ve kterém byl počítač uspán. Ovladače zařízení, které vyvíjeli výrobci hardware, byly schvalovány a digitálně podepisovány laboratořemi Microsoftu, což vedlo k vyšší stabilitě celého systému. Digitální podpis laboratoří Microsoft, bohužel přinesl i zápornou stránku a tou je zdražení hardware.

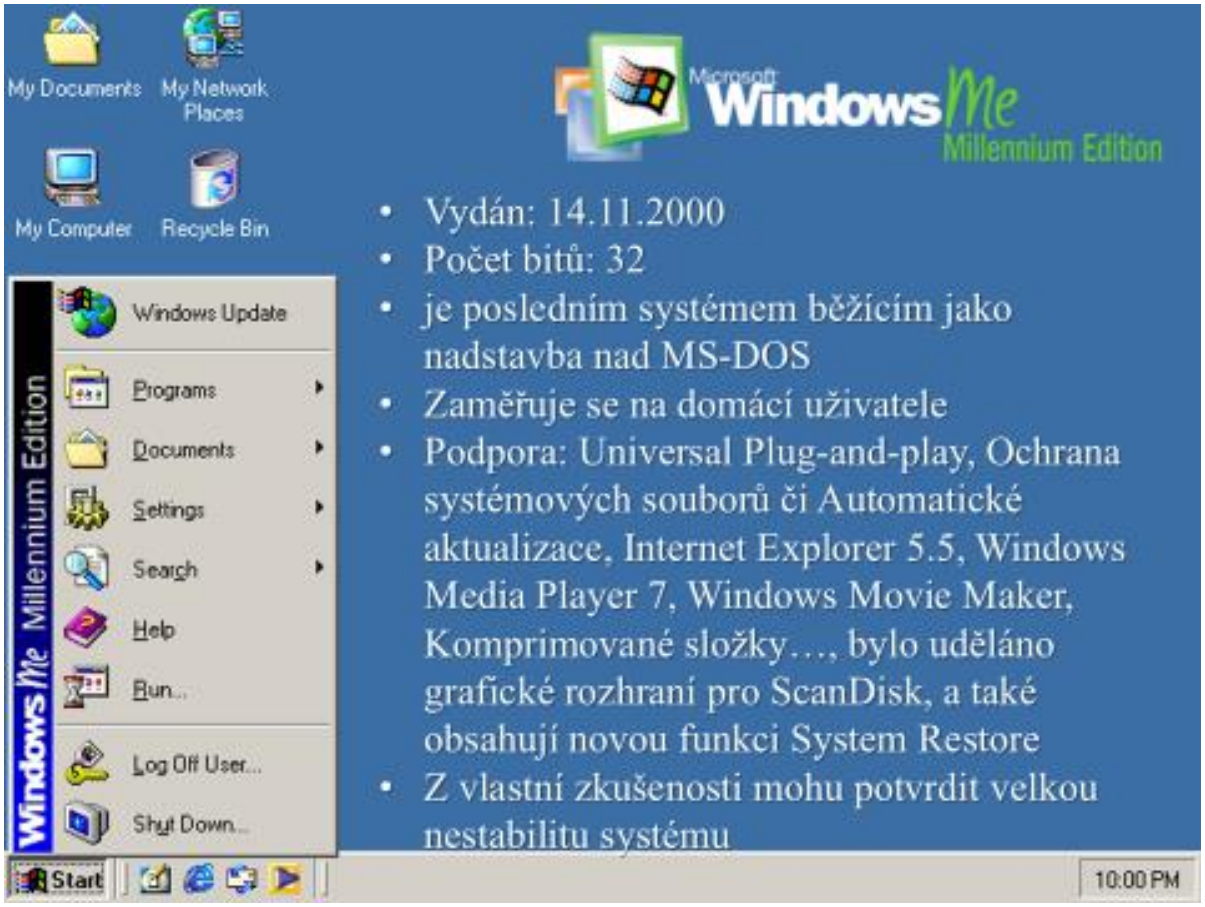
Chystaná odlehčená verze s pracovním názvem Windows Neptune nikdy oficiálně nevyšla. Celkem byly vydány 4 servisní balíčky (SP).



3.8.12.1. Systémové požadavky

Windows 2000 Professional:
133 MHz nebo vyšší Pentium-kompatibilní Procesor
32 MB RAM
650 MB místa na HDD
Windows 2000 Server:
133 MHz procesor
256 MB RAM
1 GB na HDD
Windows 2000 Advanced Server:
133 MHz Procesor
256 MB RAM
2 GB HDD

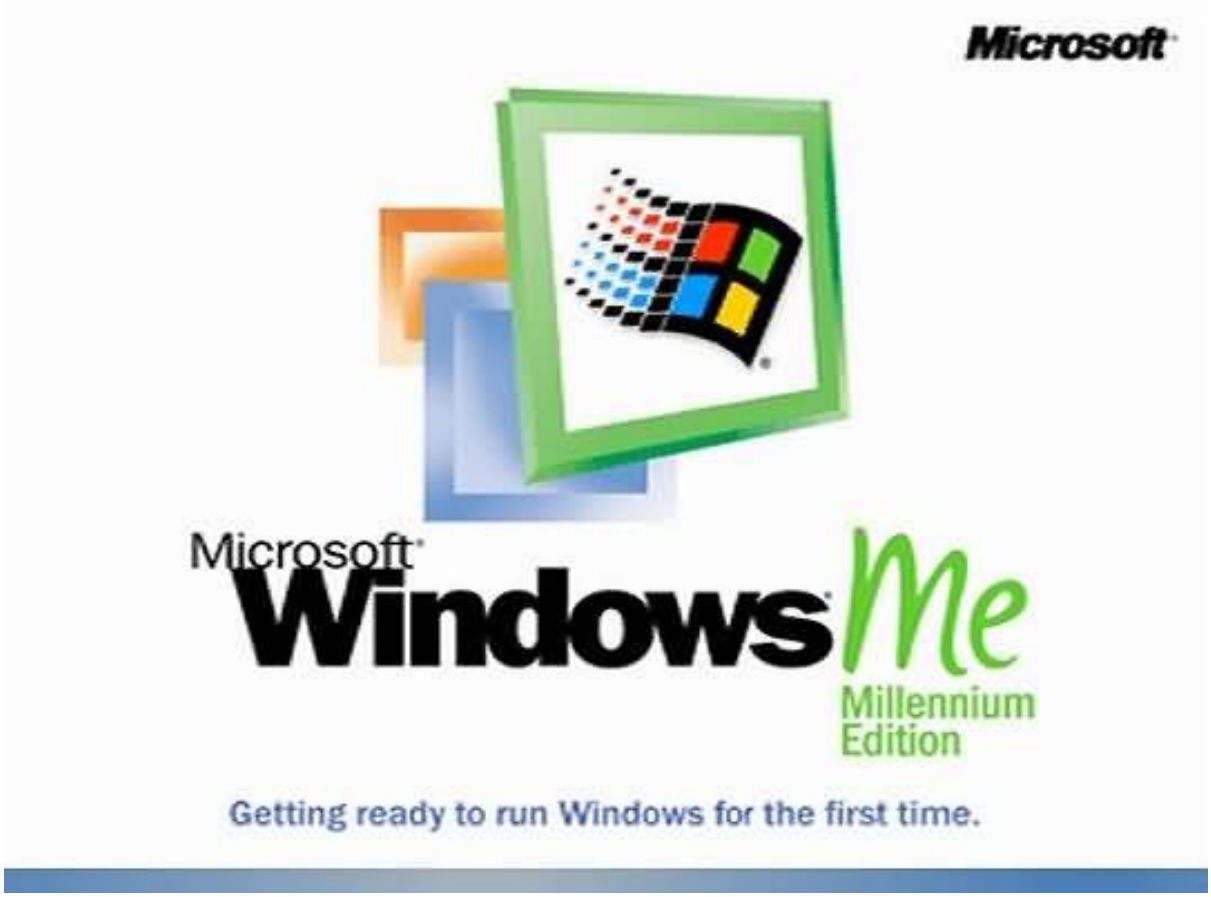
3.8.13. Windows Me



Mimo jiné také standardně obsahují ovladače pro USB Mass storage zařízení. System Restore = obnovení předchozího nastavení v případě havárie systému. Systémové požadavky Minimální:

Procesor: 150 MHz – Intel Pentium kompatibilní
Místo na HDD: 320 MB
Operační paměť: 32 MB
Doporučené:
Procesor: 300 MHz – Intel Pentium II kompatibilní
Místo na HDD: 2 GB
Operační paměť: 96 MB

Pozn.: Pokud se instalační program spustí s parametrem /nm, nebude se testovat, zda má počítač dostatečný výkon.



3.8.14. Windows XP (NT5.1)

Když se řekne Windows XP, tak nevím o člověku, který by je neznal. Jde o druhou z legend společnosti Microsoft, která nás po 13 letech s ukončenou podporou pomalu opouští, takže 

zmizí z povědomí generace našich dětí, které o této legendě budou již jen číst v knihách jako je tato.

Číslo verze znamená pořadí v řadě jader s technologií NT, předchůdcem byl Windows 2000 s označením NT5. Vývoj byl veden pod kódovým označením Whistler a na trh byl systém uveden 25. října 2001 pod obchodním názvem Windows XP (s číslem verze 5.1.). V české verzi byly vydány 1. prosince 2001. Představeny byly v Marriott Marquis Theatre v New Yorku. Windows XP jako první umožnil budoucím technikům IT simulovat pomocí aplikace celou síť v jednom PC. V roce 2005 byla vydána verze Windows XP Professional x64 Edition, která je určena pro 64bitové procesory (např. AMD Athlon 64, Intel Itanium atd.). Systém podporoval 4 GB paměti RAM a měl zdokonalenou správu napájení.

Minimální požadavky: procesor 300 MHz, 128 MB paměť, 1,5 GB na HDD, rozlišení 800x600. Vznikl systém kontrolování ovladačů, kdy Windows můžou zablokovat problematický ovladač, nebo vrátit předchozí.

Ještě trocha teorie. Aktualizační balíček integruje rozsáhlejší sadu oprav (bezpečnostních aktualizací) a nových verzí integrovaných programů, které byly k danému datu vydány. SP2: Zaměřen byl na bezpečnost.



NX bit je technologie, která zamezuje zneužití přetečení zásobníku a tím i běžný postup pro šíření virů. Aktualizován byl Windows Firewall, který je od SP2 implicitně aktivní. Podpora SP2 již byla ukončena.

Systémy Windows označené písmenem N jsou výsledkem antimonopolního řízení Evropské komise, po kterém kromě zaplacení pokuty 497 miliónu euro musela firma Microsoft vydat verzi Windows XP bez integrovaného přehrávače Windows Media Player. Provedení a cenová politika verzí N je však taková, že se nesetkaly se zájmem uživatelů.



3.8.14.1. V praxi

- Z ploch mizí: ikony Tento počítač, Dokumenty nebo Místa v síti
- na ploše naleznete jen Koš
- Systém je po grafické stránce kompletně přepracován
- Vznik dvousloupcové nabídky start
- Byla také upravena řada dalších maličkostí (jiné ovládací panely, přihlašovací obrazovka). Byla zařazena možnost rychlého přepínání uživatelů, funkce vzdálené pomoci, integrovaná podpora vypalování CD/DVD

3.8.15. Windows Server 2003

Mohli by jste mi vytknout, že serverové počítače při naší procházce opomíjím, tak to pojďme napravit.

Windows Server 2003, Web EditionWindows Server 2003, Web Edition, představující novou verzi v řadě produktů Windows Server, je webový server. Poskytuje organizacím platformu pro provoz webových serverů a hostitelských služeb, kterou lze snadno zavádět a spravovat. Díky technologii Microsoft ASP.NET, která je součástí platformy .NET Framework, poskytuje Windows Server 2003 Web Edition vývojářům platformu pro rychlé vytváření a zavádění webových služeb a aplikací XML.

Windows Server 2003, Standard EditionWindows Server 2003, Standard Edition je síťový operační systém. Windows Server 2003, Standard Edition nabízí řešení pro sdílení souborů a tiskáren, připojení k Internetu, centralizované zavádění osobních aplikací a spolupráci mezi zaměstnanci, partnery a zákazníky. Windows Server 2003, Standard Edition podporuje symetrické zpracování dvěma procesory a až 4 GB paměti.

Windows Server 2003 Enterprise Edition je server pro střední a velké organizace. Poskytuje funkce potřebné k zajištění provozu podnikové infrastruktury, použití komerčních apli-

kací a transakcí elektronického obchodování. Windows Server 2003, Enterprise Edition je multifunkční operační systém, který podporuje využití až osmi procesorů a poskytuje funkce pro podniková prostředí, jako je například vytváření clusterů se čtyřmi uzly, a až 32 GB paměti. K dispozici je také pro 64bitové počítačové platformy.

Windows Server 2003, Datacenter Edition je určen pro velké organizace. Poskytuje základ pro vytváření důležitých řešení pro databáze, software pro plánování podnikových zdrojů (ERP), zpracování velkého množství transakcí v reálném čase a slučování serverů. Jde o nejvýkonnější serverový operační systém, jaký kdy společnost Microsoft představila. Podporuje symetrické zpracování až 32 procesory a poskytuje možnost vytváření clusterů s osmi uzly a vyrovnávání zatížení jako standardní funkce. Windows .NET Datacenter Server je k dispozici také pro 64bitové počítačové platformy.

3.8.16. Windows Vista (NT6)

Všichni si říkáme pane bože, zase Vista. Tento v pravdě nedodělek systému Windows 7, který je produktem spíše obchodního oddělení nežli rozhodnutí vývojářů, zkřížil cestu ne v tom dobrém asi většině z nás. Pravda je ta, že alespoň naostro otestoval některé věci, které byly úspěšně implementovány do sedmiček a některé nápady pohřbil a tím umožnil vznik legendy. Tak se pojďme na tu černou ovci podívat.



Oficiální název Windows Vista byl představen 22. července 2005. Dne 27. července 2005 byla uvedena první betaverze, Windows Vista Beta 1 (build 5112), která byla k dispozici předplatitelům MSDN, vybrané části betatesterů a na některých konferencích pořádaných Microsoftem, jako např. Professional Developers Conference (PDC) a WinHEC. V květnu 2006 byla uvolněna Beta 2. V září 2006 společnost Microsoft uvolnila verzi Release Candidate 1 (RC1), od 14. září 2006 volně ke stažení na firemních webových stránkách.

Finální produkt (RTM) byl v anglické verzi do výroby uvolněn 9. listopadu 2006. Prodej byl v Česku zahájen v listopadu 2006 (pro firemní zákazníky prostřednictvím multilicenčních smluv) a 30. ledna 2007 pro všechny ostatní. Česká verze byla k dispozici od 1. března 2007.

Je určen pro obecné použití na domácích či firemních osobních počítačích, laptopech či mediálních centrech. Systém sdílí důležité části se systémem Windows Server 2008.První aktualizační balíček byl uvolněn 18. března 2008. Druhý (SP2) byl uvolněn 26. května 2009.

Dne 27. října 2009 byl vydán Platform Update for Windows Vista, který obsahuje některé komponenty vydané společně s Windows 7 včetně aktualizovaných běhových knihoven.

3.8.16.1. Základní technologie

V základní struktuře operačního systému byly provedeny zásadní úpravy, kam patří mimo jiné využití technologie NET Framework, nový grafický a zvukový subsystém, výraznější podpora pro digital rights management (DRM, ochrana proti neoprávněnému kopírování), lepší podporu pro instalaci programů (technologie ClickOnce a nová verze služby Windows Installer).

3.8.16.2. Novinky

Ve Windows Vista se objevuje zcela přepracované grafické rozhraní, označované názvem Aero, které využívá počítačovou 3D grafiku. Podporuje průhlednost oken a nabídek, trojrozměrné animace, ikonky přizpůsobené vyšším rozlišením apod. Pro funkčnost rozhraní Aero je nutné mít v počítači grafickou kartu, ke které musí existovat WDDM ovladač. Grafické rozhraní bylo přepracováno a obsahuje mnoho nových funkcí. Na mnoha místech je využíváno vyhledávání (kromě adresářů a souborů i na různé seznamy), které funguje na principu indexování. Ve vyšších verzích obsahuje funkci stínová kopie (anglicky shadow copy), která zajišťuje ukládání omezené historie obsahu měněných souborů (základ obnovování v sytému Windows 8 a 8.1). Obsahuje také nové funkce, jako např. kombinované filtry, nebo uložená vyhledávání, XML Paper Specification.

XPS je nový formát dokumentů založený na XML, podobný formátu PDF od Adobe Systems. Cílem formátu XPS je umožnit uživatelům prohlížení, tisk a archivaci dokumentů i bez programu, který dokument původně vytvořil.

Shell také obsahuje řadu vylepšení v zobrazování souborů – při pohledu do adresáře se zobrazují náhledy multimediálních souborů spolu s popisem typu souboru a možností zobrazení z měny odporovaných metadat.

Počítačové sítě - implementace sady protokolů počítačových sítí s podporou UPnP. Další novinky Podpora pro NX bit nabízený novými procesory s technologiemi AMD64 a EM64T, vestavěná podpora pro vypalování DVD, včetně podpory pro systém Mt. Rainier, systém Ximage pro ukládání diskových obrazů, zkracující instalaci Windows Vista na cca 15 minut, nativní podpora formátu RAW, který se používá ve většině profesionálních digitálních fotoaparátů, vestavěná podpora pro RSS, včetně API pro další programy a vylepšené šifrování souborů.

Obsahuje nástroj pro podpora diagnostiky problémů s připojením k počítačové síti. Podporuje UPnP, které umožňuje přehledně zobrazit ostatní zařízení v počítačové síti, nastavovat je a sledovat jejich činnost. NX bit nabízený novými procesory s technologiemi AMD64 a EM64T. Tato funkce umožňuje označit některé části paměti jako nepoužitelné pro provádění programu, což se využívá pro zvýšení odolnosti proti bezpečnostním chybám.

Další na první pohled viditelnou novinkou jsou uživatelské miniaplikace na ploše, které informují o využití HW, stavu baterie, počasí, emailech atd.

3.8.16.3. Praxe

Bohužel šlo o výrazný krok vedle, systém měl největší spotřebu aktualizací, klidně i 120 za den, což vedlo k jeho přezáplatovanosti. Systém jako takový byl velmi nestabilní a potřeboval velmi silný HW (platforma x86 potřebovala 3 GB RAM) pro relativně stabilní běh systému. Z vlastní zkušenosti mohu říct, že deklarované minimální požadavky jsou pro chod systém nereálné a systém potřeboval o instalaci velké množství úprav, aby byl použitelně rychlý.

3.8.17. Windows Server 2008



3.8.18. Windows 7 (NT 6.1)

Tak tuto legendu je zbytečné komukoliv představovat, spousta z vás ji zná z vlastní zkušenosti a aktivně na ni pracuje v dnešní době. Proto si jen řekneme, jaké novinky nám tento systém přinesl a něco málo o jeho historii.

Systém byl vydán 22. října 2009 a k dostání je 64bitová i 32bitová verze na rozdíl od serverové verze, která je pouze 64bitová. Jde o systém, který má za úkol nahradit xp a nt5 s miléniji. Je kombinací technologií Visty a upraveného jádra xp. Celý systém se zkoušel převážnou část testovací doby pod křídly xp.

Dne 13. července 2010 byla zveřejněna betaverze SP1, který je dnes již nedílnou součástí Systému. Ještě jedna vsuvka týkající se dnešního stavu, z důvodu zacelení bezpečnostních trhlin v systému již nejsou k dostání rozšiřující mini aplikace na plochu na oficiálních stránkách výrobce.

3.8.18.1.Přehled edic

- Home Premium
- Professional - verze pro firmy
- Ultimate - nejvyšší edice
- Další edice jsou: Starter - verze pro použití pro netbooky a slabší PC sestavy
- Home Basic - základní verze pro domácnosti
- Enterprise - verze pro firmy používající multilicence

3.8.18.2. Novinky

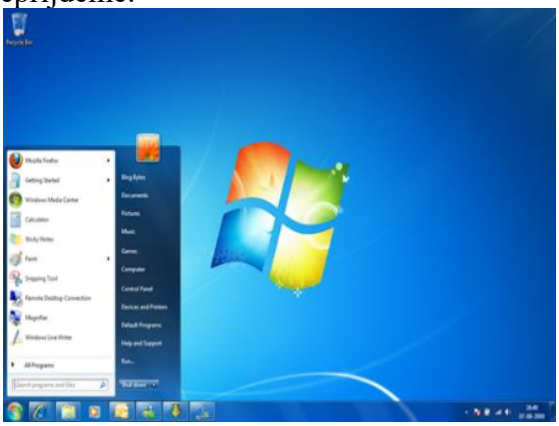
Windows 7 obsahuje řadu změn a vylepšení oproti předchozím verzím systému Windows, jako je použitelnější vyhledávání, rychlejší nastartování systému, lepší podpora více jádrových procesorů, podpora pro virtuální pevné disky, vylepšené a zmenšené jádro nebo vylepšené rozpoznávání řeči a rukopisu, které podporuje češtinu, ale řeč ne. Mezi nové funkce Milestone 1 také patří tzv. gadgety integrované přímo do Průzkumníka Windows. gadget pro Windows Media Center, schopnost vizuálního PINu, možnost spouštění položek jak z nabídky Start, tak z hlavního panelu a lepší mediální funkce. Kalkulačka umožňuje více řádkové vkládání dat, programovatelné statistické režimy a převodník jednotek.

Je integrován XPS Essentials Pack, dále Windows PowerShell Integrated Scripting Environment (ISE).

Do Ovládacích panelů bylo přidáno mnoho nových položek: Akcelerátory, ClearType Text Tuner, Displej kalibrace barev, Průvodce miniaplikacemi, Infračervený port, Uzdravení, Odstraňování závad, Pracovní Centra, Zaměřovací a další senzory, Manažer doporučení, Biometrická zařízení, Systémové ikony, Windows Solution Center, Zobrazení (Displeje).

Centrum zabezpečení systému Windows bylo přejmenováno na Windows Solution Center (Windows Health Center ("Windows Zdravotní středisko") v předchozích verzích systému Windows 7), které nyní zahrnuje i zabezpečení a údržbu počítače.

Dalšího výrazného vylepšení se dočkal režim Hibernace, který umožňuje uspat zařízení s opravdu nulovou spotřebou elektrické energie. Pokud zařízení v tomto režimu sebereme veškeré napájení, tak o nic nepřijdeme.



3.8.19. Windows 8 (NT6.2)

Windows 8 je operační systém z řady Windows NT, který je určen pro použití v desktopových i přenosných počítačích včetně tabletů. Windows 8 byl uveden do prodeje 26. října 2012. Ukončení základní podpory je naplánováno do 9.1.2018, dále budou poskytovány jen bezpečností aktualizace až do ukončení rozšířené podpory dne 10.1.2023.

Vydána byla 32bitová pro procesory x86, 64bitová verze pro procesory x86-64 a verze Windows RT pro procesory ARM, používané v mobilních zařízeních. Windows 8 je založen na uživatelském rozhraní Modern User Interface (původní název je Metro), které je dříve

známé ze systému Windows Phone. V systémových aplikacích se také rozšířila vysouvací nabídka Ribbon, která má nahradit klasické textové menu a byla použita už v některých aplikacích systému Windows 7, jako třeba Malování, a také v novějších verzích balíku Microsoft Office. Kvůli změnám v jádře systému může dojít k nekompatibilitě ovladačů. Aplikace napsané pro architekturu x86-64 nejsou kompatibilní s verzí pro architekturu ARM.

3.8.19.1. Minimální požadavky

- Procesor o taktu 1 GHz (x86 kompatibilní, s ARM Consumer Preview nepočítá).
- 1 GB operační paměti pro 32bitovou edici, 2 GB pro 64bitovou edici.
- 16 GB místa na disku pro 32bitovou edici nebo 20 GB pro 64bitovou edici.
- Grafická karta podporující DirectX 9.

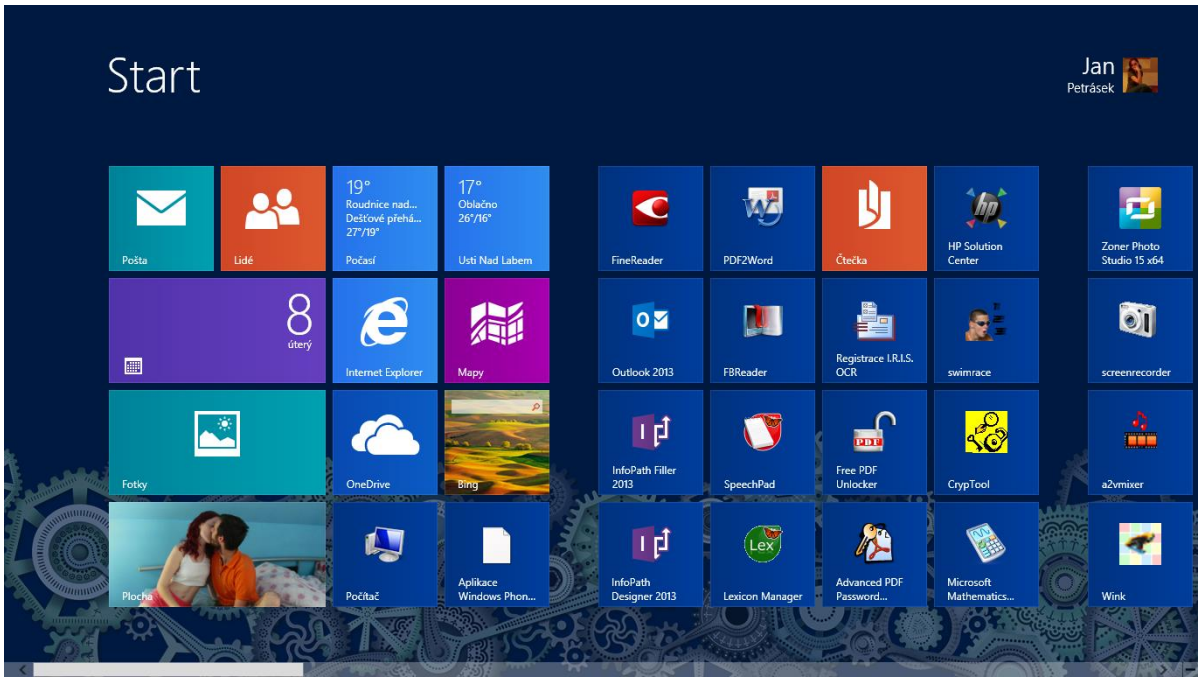
3.8.19.2. Novinky

3.8.19.2.1. Modern UI

Metro (nyní oficiálně Modern UI) je v informatice označení uživatelské rozhraní vytvořené firmou Microsoft, které se používá v aktuálních systémech Windows Phone a Windows 8. K jeho zásadám patří jednoduchost, zaměření na obsah a typografii. Označení Metro je ochranná známka německé firmy Metro AG, a proto není tento název firmou Microsoft používán. Pro vývojáře je používáno označení Modern UI. V současné době je rozhraní Microsoftem odkazováno jako rozhraní Windows 8.

První náznaky tohoto prostředí lze spatřit už ve Windows Media Center, kde text sloužil jako hlavní způsob navigace. V roce 2006 použil tyto zásady pro svůj vzhled Zune a do dnešní podoby uvedl Metro Windows Phone, ve kterém byly poprvé představeny „živé dlaždice“. Nově je Metro obsaženo ve Windows Phone, Xbox 360 a Windows 8.

Pro Windows Phone vytvořil Microsoft speciální písmo Segoe WP, ve Windows 8 je použito písmo Segoe UI Light. Windows 8 rovněž zahrnuje starší desktopové rozhraní jako dlaždici na obrazovce Start.



3.8.19.2.2. Přihlašovací obrazovka

Přihlašovací obrazovka se skládá z dvojice obrazovek, jedné, která zobrazuje základní informace (čas, datum, den, síť) a uživatelem zvolené informace.

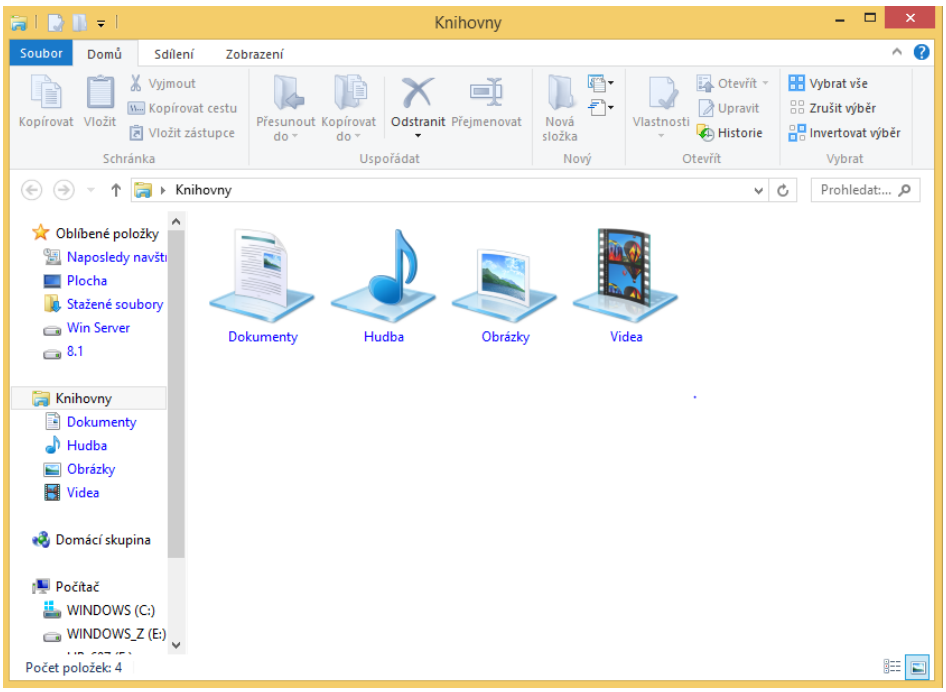


Následuje obrazovka s vlastním přihlášením, která je podobná té z Windows 7, pokud není užito obrázkové heslo.

3.8.19.2.3. Obrázkové heslo

Umožňuje přihlásit se do systému pomocí posloupnosti trojice bodů, které si uživatel zvolí na zadaném obrázku. Mezi standartním a obrázkovým heslem lze přepínat.

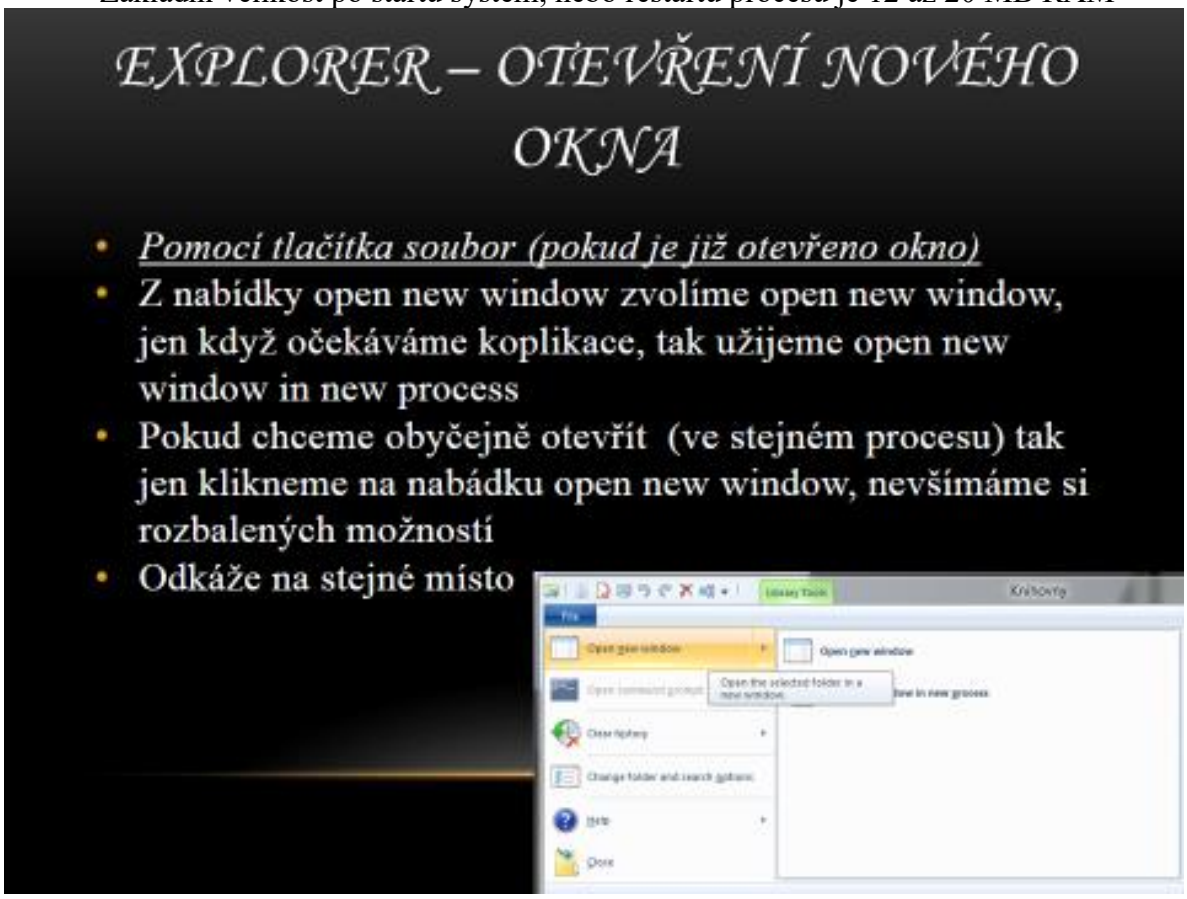
3.8.19.2.4. Průzkumník



Průzkumník zažil jednu z největších změn od Windows 95, tak si je pojďme ukázat.

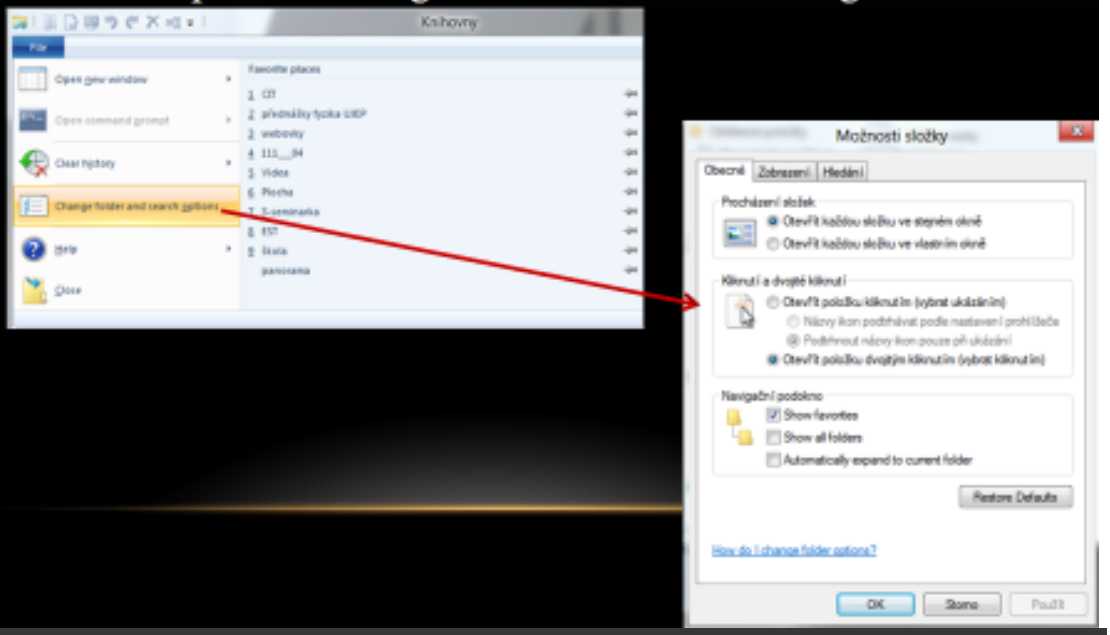
- Největší a na první pohled viditelnou novinkou je pás karet podobný aplikacím sady MS Office® 2010
- Kromě řízení, které je zdvojeno s pravým tlačítkem myši, přibyly i další ovládací prvky
- Vrací se legendární tlačítko soubor

- Je restartovatelným procesem, v případě problémů nemusí být ukončen, může být restartován
- Základní velikost po startu systém, nebo restartu procesu je 12 až 20 MB RAM



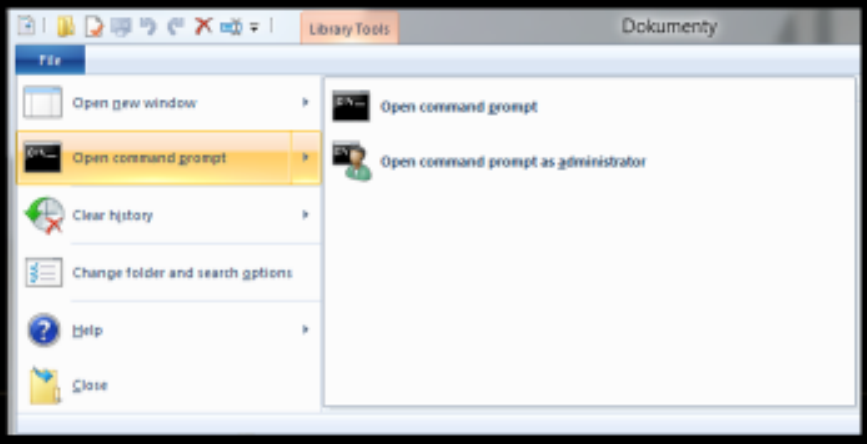
EXPLORER – MOŽNOSTI SLOŽKY

- Opět vcházíme z tlačítka soubor
- Zvolíme položku change floder and search setting



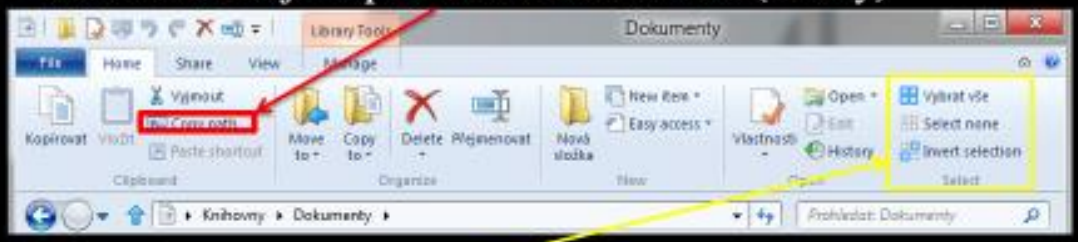
EXPLORER – MÍSTNÍ A UŽIVATELSKÉ CMD

- Opět vcházíme z tlačítka soubor (musíme mít zakliknutou složku, či disk, nebo výměnné paměťové médium, NE knihovnu)



EXPLORER – ZÁLOŽKA HOME

- Obsahuje nabídku skrytou v předchozích verzích v pravém tlačítku myši
- Novou funkcí je kopírovat adresu souboru (složky)



- Novinkou je též pomocník pro výběr položek, který nahrazuje kombinaci myši s klávesou a klávesovou zkratku ctl+A a nebo pravé tlačítko myši => vybrat vše

EXPLORER – MAZÁNÍ SOUBORŮ

- Klávesové zkratky delete a SHIFT+delete jsou nahrazeny ikonou záložky home (ale zůstávají stále platné)
- Pokud nerozbalíme nabídku a užijeme přímo ikony, tak je soubor, pokud není příliš velký, nebo nemá příliš dlouhé jméno, přesunut do koše (obdobně jako u delete)
- Pokud zvolíme z nabídky permanent delete, tak je soubor odstraněn rovnou, nepadá do koše
- Nelze odstranit systémové soubory a většina z nich nelze odstranit ani z windows.old



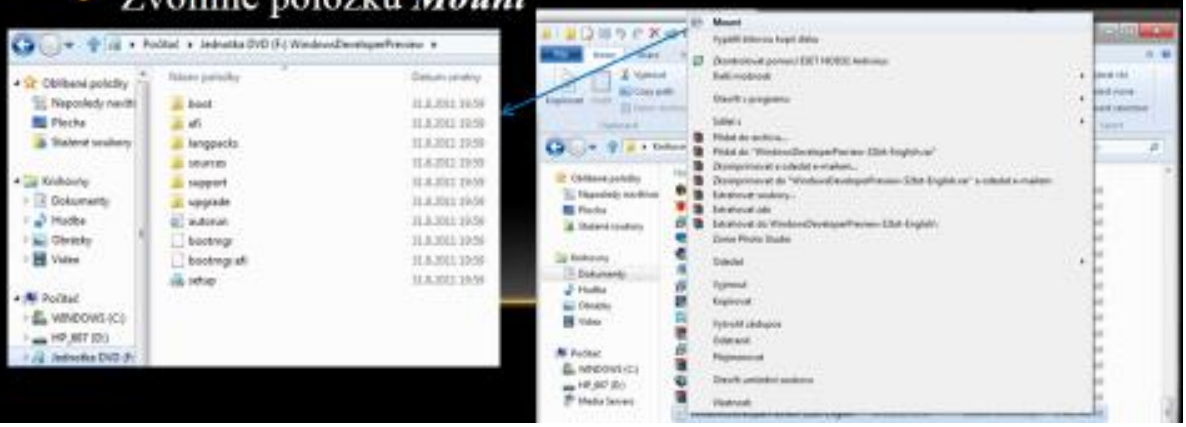
EXPLORER – VYPALOVÁNÍ, SDÍLENÍ, TISK, E-MAIL

- Tyto funkce najdeme na záložce share
- U vypalování máme možnost flash, nebo standartní
- Flash užívejte jen pro krátkodobě použitelné disky určené pro Windows 7 a 8
- Doporučuji vypalovat klasicky s přehrávačem a tudíž vypalování dokončit položkou zapsat na disk



EXPLORER – PRÁCE S ISO SOUBORY

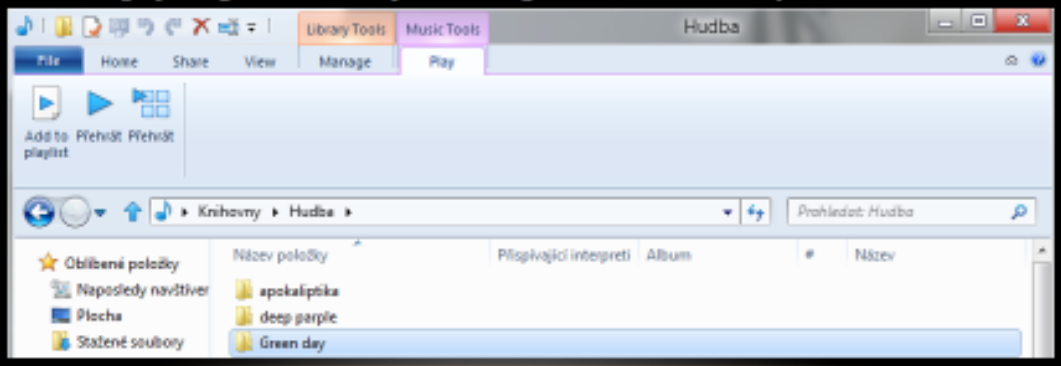
- V průzkumníku je integrovaná možnost rozbalení ISO souboru do řadiče reálné mechaniky, aniž by musela obsahovat disk s daty
- Na iso soubor klikneme pravým tlačítkem myši
- Zvolíme položku ***Mount***



Mimo jiné podporuje i soubory VHD a VHDX dále pak zip.

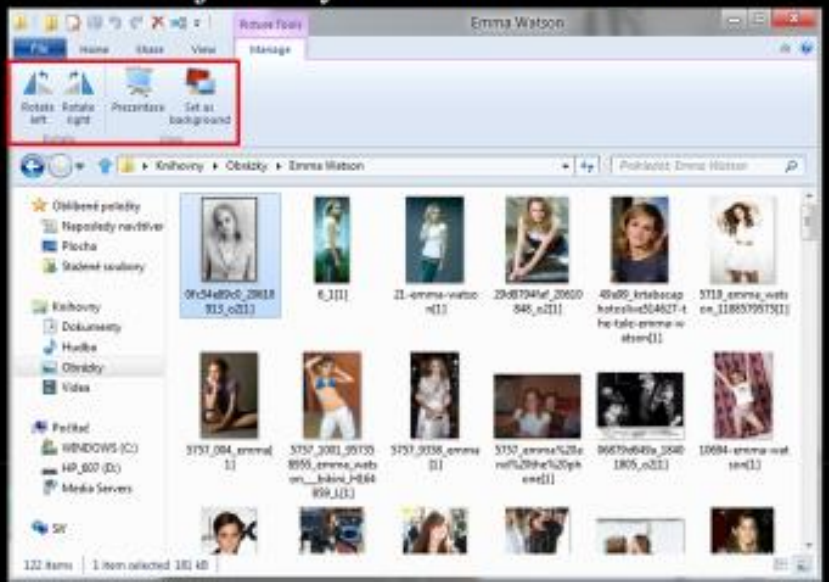
EXPLORER – HUDBA

- V knihovně hudba po označení složky se přidá záložka play
- Umožňuje přehrání obsahu složky výchozím přehrávačem (doporučen Windows Media Player)
- Funguje i pro videa, je dostupná i z knihovny videa

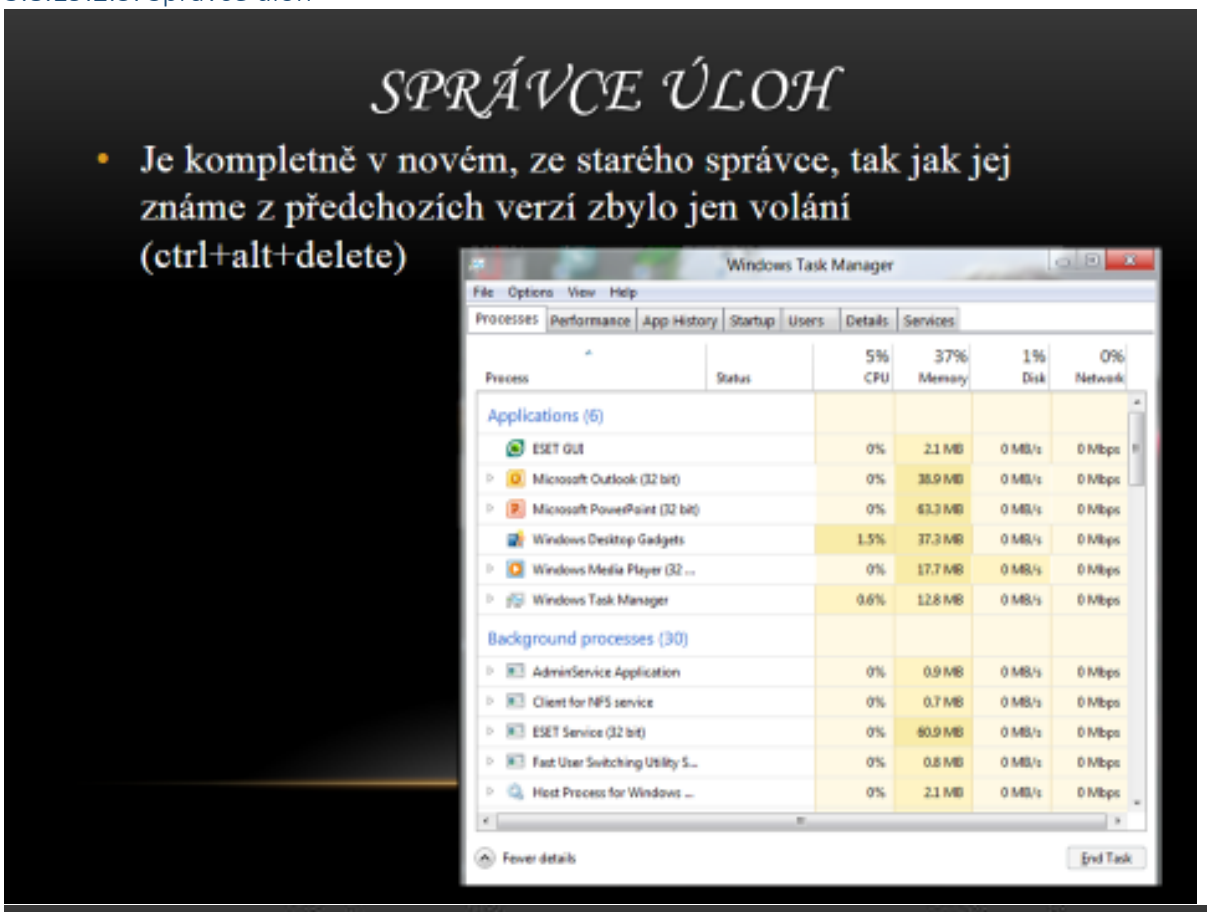


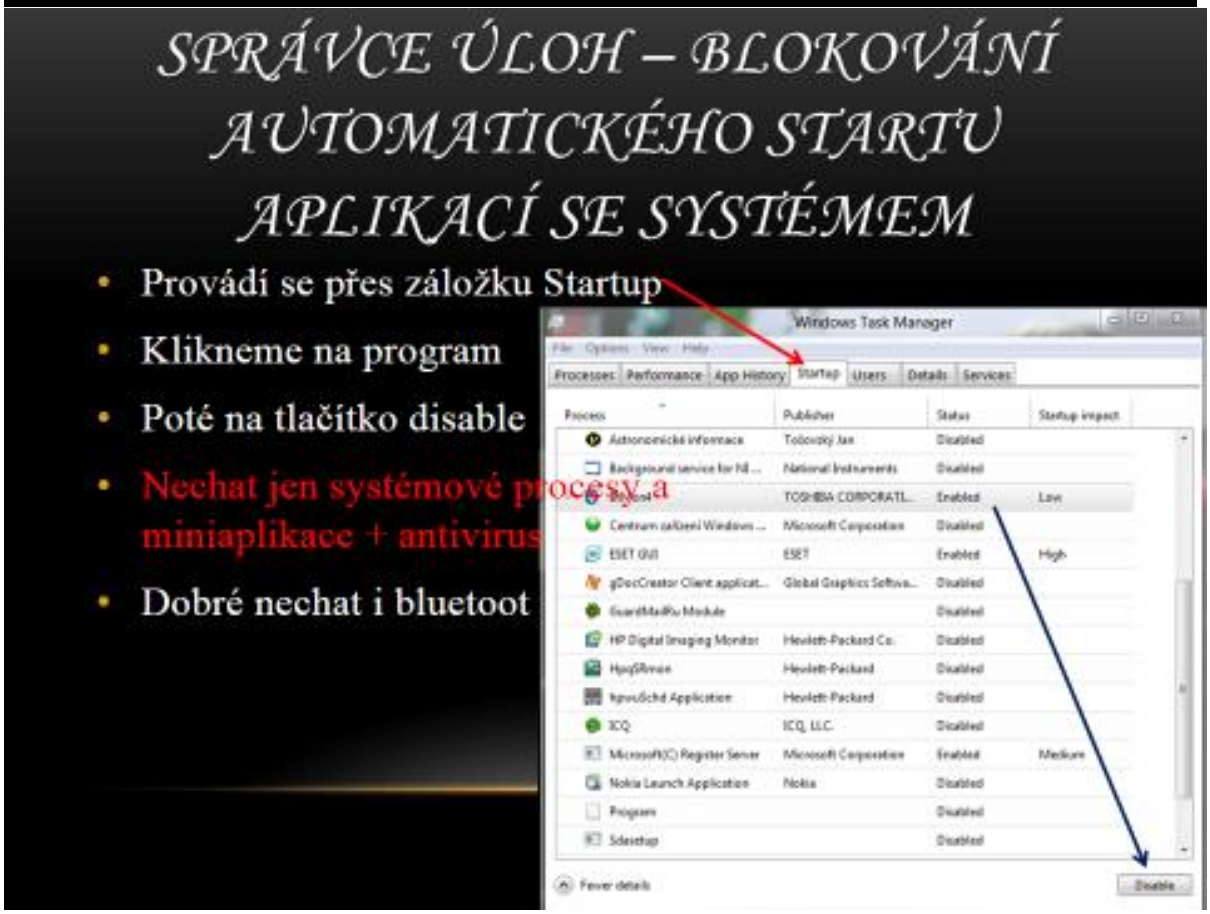
EXPLORER – OBRÁZKY

- Do samostatné karty se vrací naprosto geniální řešení základních úprav obrázků a práce s nimi z Windows XP
- K dokonalosti ještě chybí to náhledové okno



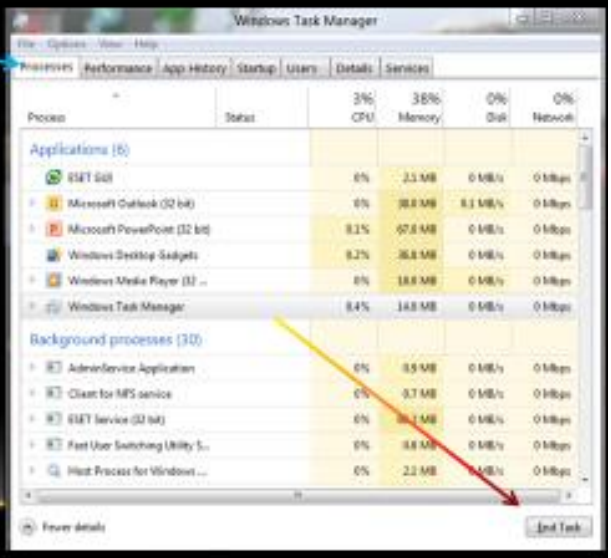
3.8.19.2.5. Správce úloh





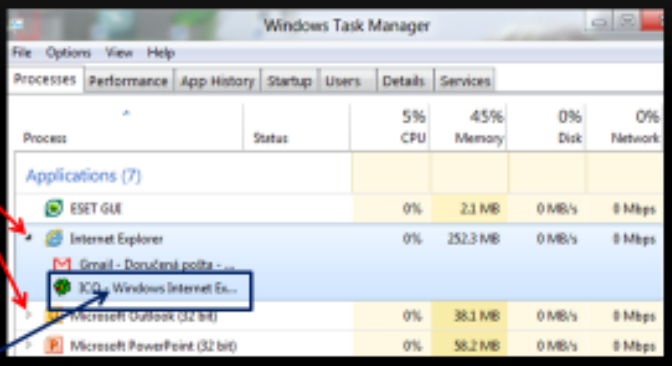
SPRÁVCE ÚLOH – UKONČOVÁNÍ APLIKACÍ

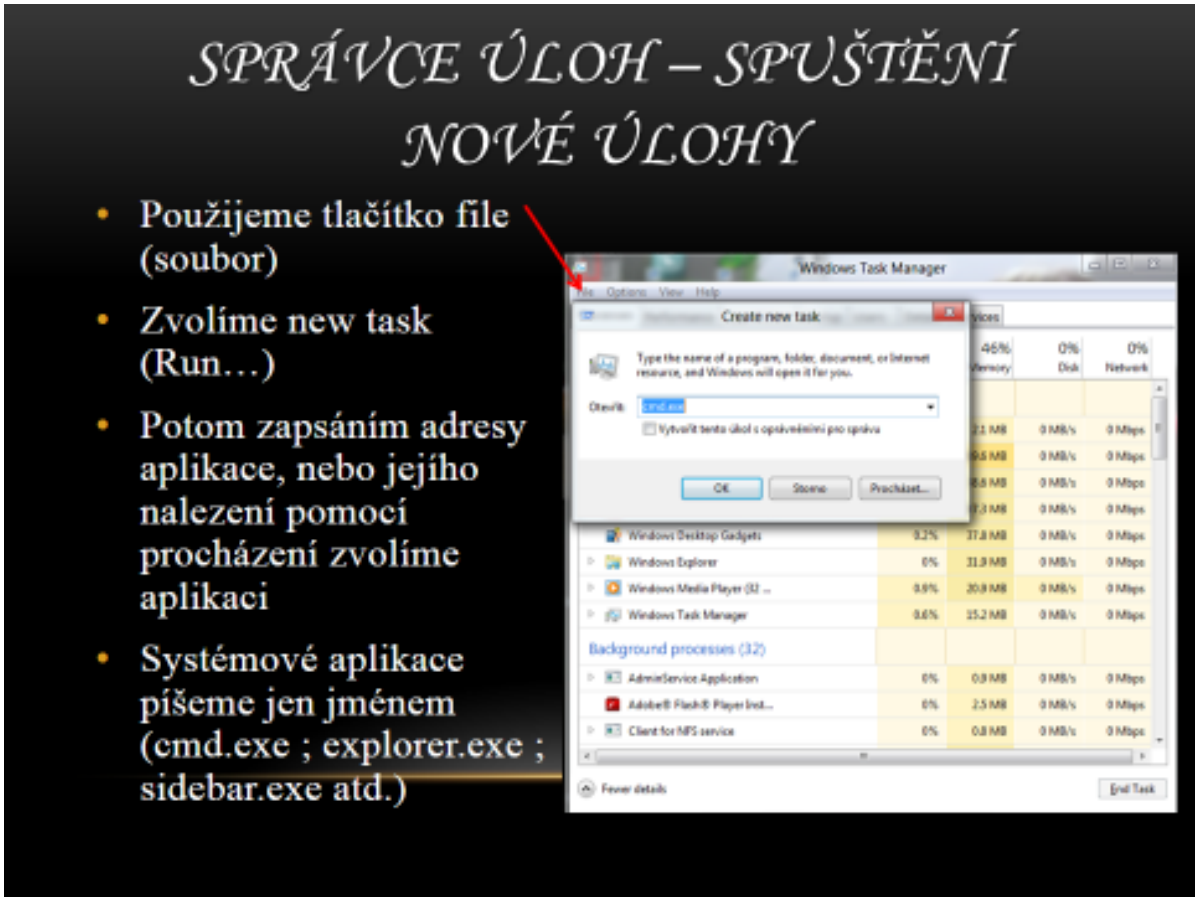
- Provádíme pomocí záložky Process
- Klineme na proces
- Následuje tlačítko End Task
- Nebo pravé tlačítko myši a End Task, či jiná editační nabídka
- Zmizela možnost ukončit strom procesu, ta je dostupná z detailů



SPRÁVCE ÚLOH – UKONČOVÁNÍ ČÁSTÍ ÚLOH

- Nejprve rozbalíme položky úlohy rozkliknutím šipky
- Označíme podproces
- Klikneme na tlačítko End Task





3.8.19.2.6. Modern UI App

Jde o aplikace, které podporují pouze Windows 8 a jsou laděné pro dotykové ovládání. Tyto apliace jsou k dostání ve Store a dostupné pouze pomocí startu, nelze je připnout na lištu nebo vytvořit zástupce na ploše. Tyto aplikace nelze minimalizovat a mohou běže dvě na jednou, jedna v kontrolním pásu o šíři 300 pixelů a druhá na zbytku obrazovky. Pokud metro aplikace není na ploše, tak hibernuje, tyto aplikace nepracují na pozadí.

3.8.19.2.6. Connected standby

Jde o nadstavbu režimu spánku určenou pro tablety. Je to obdoba režimu hybernace, ale ne úplně, procesor se může probouzet a je aktivní připojení k internetu. Zařízení v tomto režimu nesmí spotřebovat více jak 5 % kapacity baterie za 16 hodin a obnovit se do plného výkonu musí OS během 300 ms.

3.8.19.3. Klady

- Stabilita systému
- Možnosti oprav systému
- Rychlost systému
- Možnost ukončení části procesu
- Možnost restartování procesu
- Zlepšení vzájemné komunikace více webových prohlížečů
- Perfektní zpětná kompatibilita až po Windows 95
- Kvalitní univerzální ovladače
- Možnost ručního uspořádání startu
- Dobrá podpora pro práci s grafikou
- Pružnost systému

- Integrace Hyper-V
- Nízké HW nároky
- Obrázkové heslo
- Nativní podpora USB 3.0
- Nativní podpora formátů ISO a práce s VHD

3.8.19.4. Nevýhody

- Přílišné vybočení v sekci ovládání ze zaběhnutého standartu
- Nevytváření podsložek startu, ale zároveň zanášení startu uinstalama a manuály, či pomocnými částmi aplikací
- Přibytí nových klávesových zkratek, které jsou v praxi nepoužitelné, protože systém je možno řídit myší či dotykem
- Neumístění všech novinek do ovládacích panelů (myslím podrobné, nikoliv aero)
- Odstraněn režim jádra OS

3.8.20. Windows 8.1 (NT6.3)

Windows 8.1 (označován jako „Blue“) je první aktualizace operačního systému Windows 8. Microsoft se rozhodl nevydávat, pro uživatele známé, Service Packy, ale vydal přímo novou verzi, která patří do řady Windows NT. Windows 8.1 byl zpřístupněn 17. srpna 2013 ve Windows Store a 18. srpna pro všechny uživatele. V současné době má systém update 1 a další servisní balíček je na cestě.

3.8.20.1. Minimální požadavky

Procesor taktován na 1 GHz nebo rychlejší s podporou technologií PAE, NX a SSE2

RAM paměť: pro 32bitovou verzi je potřeba 1 GB operační paměti a pro 64bitovou verzi 2 GB operační paměti.

Místo na disku: 16 GB pro 32bitovou verzi, 20 GB pro 64bitovou verzi.

Grafické zařízení: podpora Microsoft DirectX 9 s ovladačem WDDM.

3.8.20.2. Novinky

- návrat tlačítka Start
- vylepšené hledání
- změna prostředí Windows Store
- více aplikací na obrazovce (až 4 aplikace)
- 3 typy velikostí dlaždic
- Aktivní dlaždice
- vytvoření obrázkového hesla s gesty
- prezentace fotografií na uzamikací obrazovce

3.8.20.3. Spouštění Windows

Windows 8.1 stejně jako předchozí verze Windows 8 umožňuje uživatelům výběr, jak bude systém vypnut a následně z jakého stavu bude spuštěn. První volba je klasické vypnutí systému, které má za následek, delší spouštění systému při dalším zapnutí. Druhá varianta je Hybrid vypnutí systému, které výrazně zrychlí start systému. Je to ve své podstatě kombinace mezi hibernací a režimem spánku.

Novinkou ve Windows 8.1 je spouštění na plochu. To znamená, že uživatel si může zvolit, zda se systém po zapnutí přepne na klasickou plochu, nebo do Start menu. To je oproti Windows 8 výrazná změna, jelikož spousta uživatelů stále dává přednost klasické ploše známé z předchozích verzí.

3.8.20.4. Tlačítko Start

Microsoft vrátil uživatelům tlačítko Start, ovšem zde nemá stejnou funkci jako v dřívějších OS. Zde tlačítko Start přepíná mezi prostředím Modern UI a klasickou plochou. Stále tedy uživatelé musí postrádat klasickou nabídku Start, integrovanou přímo v systému, pokud uživatel hledá nějakou náhradu, lze si doinstalovat aplikaci nahrazující nabídku Start.

Microsoft přidal rozšířenou nabídku při pravém kliknutí na tlačítko Start, které alespoň trochu vylepšuje ovládání systému. Důležitou položkou, která zde přibyla, je možnost vypnutí nebo restartování systému.

3.8.20.4. Přizpůsobení

Pro uživatele přibyla možnost nastavení průhledného pozadí v prostředí Start menu, tudíž má stejné pozadí na ploše i v prostředí dlaždic. Další možností nastavení je změna velikosti dlaždic, kde lze nastavit 4 rozměry – ¼, normální, široká a velká (2×2). Velkým přínosem, jsou změny v sekci nastavení, kde přibylo mnoho voleb seřazených do sekcí podobně jako v Ovládacích panelech. Další rozšíření nabízí uživatelům nastavit jako spořič displeje video, zobrazování fotek na zamčené obrazovce, nová témata pro zamykací obrazovku, rozdělení obrazovky pro více aplikací, automatické aktualizace aplikací v prostředí Modern UI atd.

3.8.20.5. Rozšíření

Za zmínku stojí vylepšené hledání, které nyní vyhledává nejen na pevném disku, ale nově i na disku SkyDrive a na Internetu. Dále je velkou výhodou odinstalace aplikací, kde lze aplikace odinstalovat pouze z aktuálního zařízení, nebo ze všech zařízení, které uživatel používá (je na nich přihlášen stejným jménem). To ale není u odinstalace vše, dále je možnost odinstalace více aplikací najednou, nemusí se tedy odinstalovávat každá aplikace samostatně. Mezi další rozšíření, které najdeme v systému Windows 8.1, patří vylepšení a přidání aplikací, jako například aplikace Budíku, kde je možné nastavit si upozorňování. Také přibyla možnost nastavit si v systému režim klidu (od kdy do kdy) a systém nebude zasílat žádná hlášení.

3.8.20.6. Update1

9. dubna 2014 byla uvolněna aktualizace pro Windows 8.1 označená jako Windows 8.1 Update. Má vylepšit používání s klávesnicí a myší – moderní aplikace se budou moci připínat na hlavní panel, hlavní panel se bude objevovat i v aplikacích pro moderní rozhraní a tyto aplikace bude možno zavírat křížkem, jak je tomu v desktopových aplikacích či minimalizovat. Dále přibyde možnost vypnout, hledat, restartovat a podobně do pravé horní části nabídky start. Celkově se zlepší i tak velmi dobrá rychlost a stabilita systému, přičemž zůstává zachována pružnost, jaká byla před updatem.

3.8.20.7. Další informace

Podrobnější informace o systému a videonávod pro začátečníky, který představuje možnosti systému a jeho novinky najdete na webu: http://elektrotechnika.vesele.info. K dostání jsou i iso otisky instalačních médií a aktualizace update.

Serverovou verzí je systém Windows Server 2012 R2, který má velmi nízké nároky a výkon, kloubí technologie serverového OS a systému Windows 8.1.

3.8.21. Windows 10

Má zcela novou filosofii ve srovnání s minulými verzemi, nyní se i větší úpravy provýádějí přímo v životním cyklu systému. Životnost jedné verze OS je plánována na 6 až 8 měsíců, kdy je vydána verze nová. První komerční verze měla koncové označení 240 a byla vydána 29. července 2015. Jako první OS byla v rámci Insider programu (náhrada původního Dewelopers programu) od začátku vývoje k dispozici v českém jazyce bez nutnosti instalovat jazykový balíček. Verze 240 byla oproti poslední vývojové verzi stabilní a dokonce stabilnější,

nežli verze vydaná na podzim téhož roku pod označením končícím 586. Nejpovedenjší vezí byl Aniversary update, vydaný s označením 1607 roku 2016 a jako poslední umožňoval bezplatný přechod z předchozích verzí Windows. Micorosoft se rozhodl majitelům Windows 7, 8 a 8.1 umožnit (následně i vnucoval) přechod na Windows 10 bezplatně.

Windows 10 je rozdělen do několika vývojových linií, ačkoliv jeho hlavní myšlenkou je sjednotit systém napříč platformami od Raspberry Pi, přes telefony, počítače až po rozšířenou realitu v podobě MS Hallolens. Systém je v zásadě vyvýjen ve 3 větvích, větvi pro IoT a průmyslová zařízení, standartní verzích (edice Home, Pro a Enterprise, Mobile a Mibile Enterprise) a longstable verzích stejných edic. Longstable edice obsahují o jednu edici navíc, jde o edici EDU, která poskytuje služby Enterprise edice studentům informatiky a licenční politikou je právně zabráněno jejímu použití mimo vzdělávací účely, ač licence studentům zůstává doživotně. Longstable edice mají podporu 20 let, pak je nutné provést aktualizaci na novější verzi, standardní edice zůstane v plné podpoře 16 měsíců (ob verzi nejdéle je nutné aktualizovat) a verzi, která je pouze v rozšířené podpoře nelze po naformátování diskové jednotky znovu aktivovat, je nutné ji aktualizovat na aktuální verzi, aby se systém úspěšně aktivoval (netýká se multilicenčních smluv).

Desítky s sebou přinesly řadu novinek, ale zároveň krok zpět oproti Windows 8.1, kdy si nemohu pomoci, ale 8.1 jsou stabilnější a rychlejší (především v mobilní verzi). Asi největšími novinkami jsou virtuální plochy, prostor a miniaplikace pro práci s dotykovým perem či grafickým tabletem, nástroje pro snadné sdílení, notifikační centrum a nový store. Nesmíme také opomenout webový prohlížeč Edge, který mimo jiné hned po instalaci systému zajišťuje nativní podporu pro práci s pdf soubory, další novinkou je možnost bezplatně instalovat mobilní verzi Office, což stačí na prohlížení a úpravu existujících dokumetů. Novinkou v nejvyšších edicích je také Virtual Scure Mode a mnoho dalšího i v oblasti Hyper-V. Osobně jako krok zpět krom stability hodnotím i uživatelské rozhraní, které je hybridem mezi Widnows 7 a 8.1. Určitým diktátem od MS mi přijde i integrace Skype do OS, protože integrovaný skype je stále více vývojovou verzí nežli finálním produktem. Nespornou výhodou je i potření hranice mezi Windows a Linux, kdy si vývojáři aplikací mohou aktivovat podporu Basha a spouštění linuxových aplikací. Ruku v ruce s rychlým, a ne vždy dotaženým vývojem, jde i prudký rozvoj Visual studia, které si naštěstí na rozdíl od OS zachovalo svou preciznost a začíná se z něj stávat i ve velné verzi nástroj na všechno kolem programování a ladění.

Bohužel nástup Windows 10 a ukončení podpory Windows Phone 8.1 a tím i některých fyzických zařízení způsobil velký odlive užiovatelů z mobilní verze systému a tím vedl k definitivnímu ukončení výroby telefonu Lumia a postupný útlum vývoje mobilní verze OS, ačkoliv podobnost s počítačem je zde velmi povedená včetně kompatibility MS Store aplikací.

Celkově, ač jsem se podílel na testování OS a i dnes je možné být v Insider Programu, je pro mě osobně Windows 10 zklamáním a patřím mezi ty, kteří jej opustili. Verze pro IoT je velmi zdařilá a je vidět, že zde se MS snaží ředstihnout Linuxovou konkurenci a její vývoj je opravdu k lepšímu, v telefonu se po posledních aktualizacích systém relativně ustálil, mírně zrychlil a snížil spostřebu energie, ale kvalit WP 8.1 nedosahuje. Hlavně průmyslový grafici by mě mohli teď kárat, protože OS je nativně vybaven podporou 3D tisku a nástroji postřebnými pro jeho přípravu, dokonce je nově vybaven i 3D malováním, ale problémy se stabilitou i nové rozhraní, které mi moc nevyhovuje jsem se vrátil ke svým virtuálnm Windows 8.1 a jako primární systém používám Windows server 2016, což jsou serverové desítky, ale i zde jsou věci , které mi více vyhovovali na předešlé verzi (Windows server 2012R2), především zde postrádám plnohodnotný Store, ale novinky v Hyper-V vyvažují zápory, celkově je serverová verze výrazně kvalitnější, nežli i Enterprise longstable edice desítek, kterou jsem používal a považuji ji za vrchol toho, co systém dokáže nabídnout.

Psát více o systému, který má dle všeho „patent na dlouhověkost“ nemá vzhledem k jeho rozvoji smysl, sledujete proto web MS nebo moje webové stránky.

3.8.22. Systém 6

Byl vydán společností Apple Computer v roce 1988. V systému 6 byl také použit nástroj, nazývaný Font/DA Mover na správu písem a Desk Tools (DA). Tento nástroj byl vytvořen, protože DAS nebyly nainstalovány jako aplikace, ale místo toho byly integrovány do systému softwarových souborů, podobně jako pro drivery. To znamenalo, že DAS nemohly být řízeny pomocí finderu. Font/DA Mover byl použit také pro instalaci písem do systému 6 pro použití v softwaru, například MacWrite, Apple Word na zpracování softwaru. Font/DA Mover byl poprvé vyvinut Bertem Sloanem v roce 1985 a krátce poté byl zahrnut do systému Macintosh software jako standardní.

Kooperativní multitasking realizoval Macintosh v březnu 1985 pomocí programu Switcher od Andy Hertzfelda, který umožnil uživateli spustit více aplikací a přepínat mezi nimi. Switcher nebyl zařazen do operačního systému, ale bylo možné ho zakoupit od Apple samostatně. Systém 6 měl několik významných omezení, které byly odstraněny jeho nástupcem, Systemem 7. Byla podporována 24bitová adresace paměti a omezenou kapacitu disku až 2 GB a 65536 souborů na jednotku, neumožňoval uživatelům měnit písmo, velikosti, nebo styly pro uživatelské rozhraní prvků jako jsou menu a záhlaví okna.

Cena se při uvedení pohybovala kolem 49 US dolarů.

3.8.23. Systém 7

Je stále ještě používán malým počtem uživatelů, kteří vlastní dnes již spíše archaický hardware té doby. Sdílení souborů. Zároveň s uživatelským rozhraním zlepšení pro instalaci AppleTalk, System 7 také obsahoval server pro jednoduché sdílení souborů dovolující kterémukoli počítači sdílet soubory do sítě AppleTalk.

Systém byl uveden 13. května 1991. Obsahoval funkce jako souběžné zpracování úloh, virtuální paměť, sdílení souborů, zlepšené uživatelské rozhraní, QuickTime a QuickDraw 3D.

Vlastnosti v porovnání se System Software 6, System 7 nabídnul: Vestavěný multitasking a odpadkový koš se stal běžným adresářem. Dále nabídl Aliasy - Alias je malý soubor, který reprezentuje jiný objekt souborového systému (dokument, aplikace, adresář, pevný disk, objekt na síti, vyjímatelné médium nebo tiskárna), ovládací panel se přesunul do zvláštního adresáře

Apple menu se přemístilo do menu Apple Menu Items, včetně Aliasů, Bublinová nápověda, nástroj systému podobný Kontextové nápovědě, AppleScript, a skriptovací jazyk na vytváření automatických úloh.

Typický alias je malý soubor, mezi 1 a 5 kB. Funguje jako přesměrování na objekt v systému, jako je. Ačkoli byl pro programátory aplikací pomocí AppleScript na implementaci složitý, pro uživatele to byl silný nástroj, který je dostupný dodnes jako součást Mac OS X. System 7 byl první verzí Mac OS, která se musela instalovat z více disket, neboť instalátor byl větší než jedna 1,44 MB disketa. Byl to také první operační systém od Apple, který byl distribuován na CD. CD neobsahovalo žádný další software. K počítačům Macintosh byly často přibalovány aplikace jako textový editor, šachy atd. System 7 také obsahoval aplikace využívající síťový protokol AppleTalk.

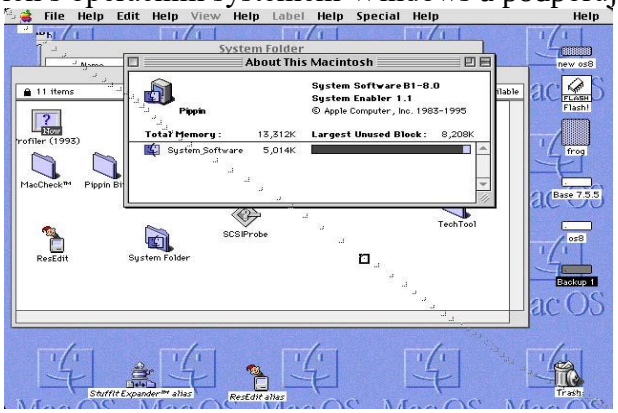


3.8.24. Systém 8

Apple vydala tento systém 26. července 1997. Mac OS 8 je jedno z nejúspěšnějších vydání software a přinesl řadu velmi důležitých změn, představil rozhraní Platinum a nativní vícevláknový (multi-threaded) Finder pro PowerPC. Vyšlo celkem 5 edic tohoto systému (8.0, 8.1, 8.5, 8.5.1, 8.6).

Další vlastnosti: Nastavení fontů a barev, Kontextová menu, Překrývající se okna, Ovládací panel pro nastavení fotografií na pozadí, Jednoduchý Finder, Vytvoření Nápovědy, rychlejších průvodců obsahujících HTML stránky s nápovědou, Podpora protokolu AFP, Zvýšení výkonu virtuální paměti, Rychlejší přestavění pracovní plochy.

Během prvních dvou týdnů se ho prodalo přes 1,2 milionu kopií. Mac OS 8.1 představil nový a mnohem účinnější systém souborů, známý jako HFS+. Mac OS 8.5 byl první verze Mac OS, která vyžadovala procesor PowerPC. Používal původní verzi QuickDraw a AppleScript, spolu s vyhledávacím nástrojem Sherlock. Mac OS 8.1 Vypuštěn 19. ledna 1998. Mac OS 8.1 představil nový systém souborů známý jako HFS+, Mac OS 8.1 také používal vylepšenou verzi PC Exchange, dovolující uživatelům vidět dlouhé názvy souborů (do 255 znaků), vytvořené na počítačích s operačním systémem Windows a podporujících FAT32.



3.8.25. MacOS X

Současná verze systému MacOS, pod názvem X pomalu uzavírá naše putování dějinami OS. Grafické rozhraní je přejímáno nebo napodobováno v iPodech, iPhonech, iPadech atd. Program zvaný RakDok umožňuje užití nástrojové lišty z Mac OS X ve Windows (NT5 a vyšší) jako pomocné nástrojové lišty.

První edice byla vydána 24. března 2001. Vznikl jako kombinace několika různých technologií, kdy základ systému se jmenuje Darwin (je složen z hybridního jádra unixového typu, XNU spolu s množstvím BSD, GNU a dalších open source nástrojů), nad jádrem je množina knihoven, služeb a technologií, které jsou přejaty většinou z NeXTSTEPu a předchozí verze

Grafické uživatelské rozhraní se jmenuje Aqua.

3.8.25.1. Snow Leopard

- 29. srpen 2009
- První plně 64bitový Mac OS X pro procesory Intel
- Většina systémových aplikací je 64bitová
- OpenCL – systémová podpora pro využití výkonu grafické karty v negrafických aplikacích
- Plně integrované vícedotykové ovládání
- Možnost zapisovaní dat do souborového systému NTFS
- Podpora MS Office 2007 a 2010
- Teoretická možnost rozšíření RAM až na 16GB



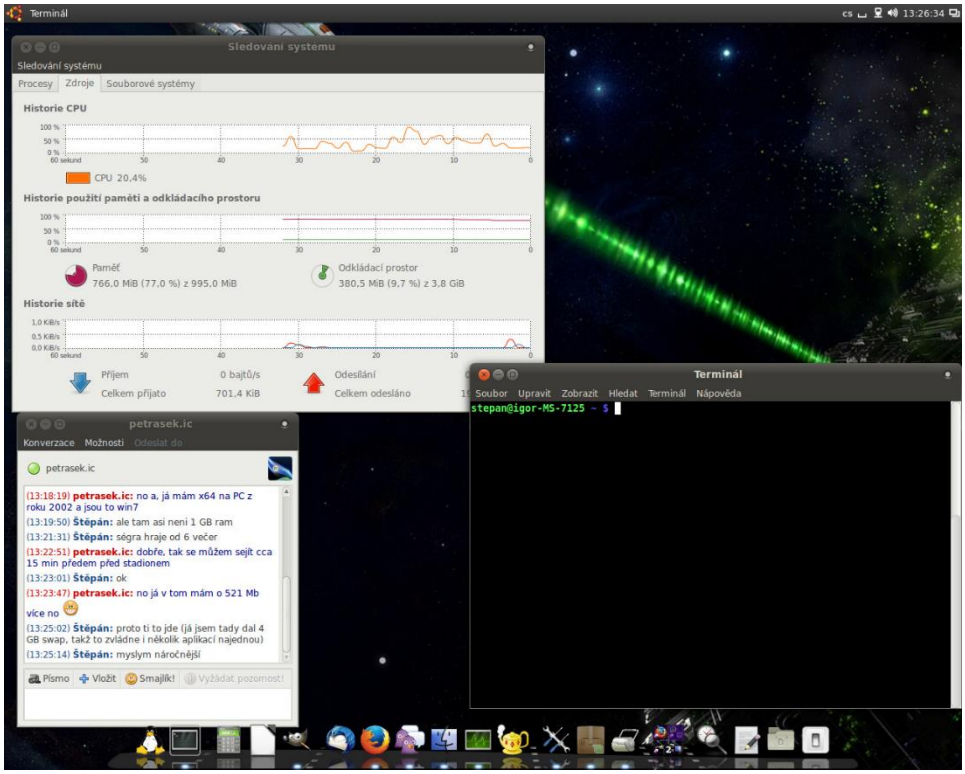
3.9. Linux

Tučňák není tak populární mezi domácími uživateli, ale jeho popularita roste hlavně mezi počítačovými piráty a je hojně rozšířen na webserverech.



Díky povaze Linuxu jsou veškeré síťové aktivity s Linuxem jednoduché a přirozené. Jediné v čem je Linux ještě lepší i mimo síť, je tzv. live CD, což není nic jiného, nežli systém, který vám umožní po totálním kolapsu OS zachránit vaše data, protože je spouštěn přímo z mechaniky CD/DVD+-R/RW a nahrán do paměti RAM. Díky hackrům v této podobě existují i Windows XP, nebo Vista, ale nikdy takovéto disky nevyšli oficiálně, na rozdíl od Linuxu. Linux je označení jádra unixového typu. Jde o internetové šířený OS, který je volně dostupný v licenci GPL. Jeho počátky lze vystopovat do konce osmdesátých let, kdy jistý finský student Linus Torvalds pracoval na univerzitě s Minixem a chtěl podobný operační systém domů. Linux je svou povahou síťový operační systém, jeho hlavní úlohou je sloužit jako OS na serverech. Linux se díky své povaze a svému šíření vyvíjí hodně rychle, což s sebou nese velké problémy s kompatibilitou, ani různé distribuce stejné verze nejsou spolu zcela kompatibilní. Linux jako takový nemá ucelenou jednotnou koncepci vývoje. Je dobrou volbou pro mladé hravé lidi, kteří nechtějí hekovat, a nemají peníze na drahé OS. Dobrý je též jako záložní OS.

Pro představu o systému zde uvádím několik snímků, které mám od jednoho stoupence tohoto systému, osobně tento systém moc rád nemám, jsem zastáncem systémů Windows 8 a 8.1, ale tím nechci říct, abyste tučňáka neměli rádi.



Jak jsem již naznačil, tak se Liunux vyvýjí v jednotlivých distribucích a za každou distribucí stojí jiná skupina vývojářů, ať z řad užovatelů, tak z řad firem. Snímky,které jsou výše pocházejí z distribude Fedora ve verzi 18 s grafickým prostředím GNOME. Většinu distribucí lze instalovat buď s GNOME neb KDE užovatelským rozhraním, kdy KDE je podobnější Windows, GNOME je podobnější OSX od Apple. Všechny distribuce umožňují online instalaci, některé i stažení iso souboru pro instalaci bez přístupu k internetu.

Distribuce dělíme na 2 velké skupiny, binární a překládané, kdy binární si se instalují z již zkompilovaných souborů obdobně jako Windows, zatím co překládané mají zkompilovaný jen instalční program, který zbytek systému stahuje (nebo načítá z definovaného uložiště) v podobě zdrojového kódu a překládá přímo v průběhu instalace, což zaručuje o něco lepší kompatibilitu s HW, ale je to cesta určená pouze pro pokročilé uživatele, příkladem je Arch linux. Celkově kombatibilita s HW na Linuxu problematičnětjěší, nežli pod jinými OS, typicky problémové jsou grafické karty Nvidia a síťové karty Realtech.

Základními serverovými distribucemi jsou Debian, Suse Enterprise, Rethead Enterprise Linux, na domácích stanicích je to pak Fedora (vhodná pro ty, co to s Linuxem myslí vážně), Mint, Ubuntu (nedoporučuju, stává se z toho uzavřená platforma podobná Apple) a Debian.

Kategorií, která je úplně samostatná je biolinux, což je speciální distribuce určená pro bioinformatiku, která má k tomu v sobě již instalované nástroje a s další sadou nástrojů v mírně předkonfigurovaném stavu je k dispozici MyPro, které je virtuálním strojem pro Virtual Box a Hyper-V.

Další speciální edicí je Kali Linux, který má v sobě nástroje pro tesování bezpečnosti počítačových sítí a webových aplikací. Jde o distribuci, která je velmi oblíbena hackery, protože má v sobě veškeré základní vybavení potřebné pro haciking a umí pracovat plnohonotně z USB disku jako Live OS, což zjednodušuje jeho použití kdekoliv a kdykoliv.

Většina komerčně využívaných distribucí je připravena pro to, aby běželi jako virtuální server nebouživatelské stanice jak na Hyper-V, tak na Vmware ve velkých datových centrech a zároveň Lunux umožňuje s velmi nízkou režijí virtualizovat další Linux s využitím jádra fyzicky instalovaného linuxu, který hostuje ten virtuální. Linux je možné si postavit i na míru na webu: http://www.instalinux.com/cgi-bin/coe_bootimage.cgi

3.10.Mobilní OS

Jde o systémy, které jsou určeny pro mobilní telefony (budeme se jimi zabývat později) a tablety. Nejčastějšími představiteli je Android od společnosti Google, následovaný Windows Phone od Microsoftu a iOS od Apple. V minulosti to byly také Windows Mobile a Symbian.

Tyto operační systémy jsou obvykle silně spjaty s hardwarovou stránkou zařízení. Systém je většinou nahrán na zvláštní interní a standardně nepřepisovatelné paměti. Různými způsoby lze dosáhnout přepsání této paměti (tzv. flashnutí) a uživatel může původní systém přemazat. Jednoduché telefony používají jednoduché tovární OS šité na míru většinou slabšímu hardware.

3.10.1. Android

Android je rozsáhlá open source platforma, která vznikla zejména pro mobilní zařízení (chytré telefony, PDA, navigace, tablety). Zahrnuje v sobě operační systém (založený na jádru Linux), middleware, uživatelské rozhraní a aplikace. Vyvíjí ho konsorcium Open Handset Alliance, jehož cílem je progresivní rozvoj mobilních technologií, které budou mít výrazně nižší náklady na vývoj a distribuci, a zároveň spotřebitelům přinese inovativní uživatelsky přívětivé prostředí. Při vývoji systému byla brána v úvahu omezení, kterými disponují klasické mobilní zařízení jako výdrž baterie, menší výkonnost a málo dostupné paměti. Zároveň bylo jádro Androidu navrženo pro běh na různém hardwaru. Systém tak může být použit bez ohledu na použitý chipset, velikost či rozlišení obrazovky.

Samotná platforma Android dává k dispozici nejen operační systém s uživatelským prostředím pro koncové uživatele, ale i kompletní řešení nasazení operačního systému (specifikace ovladačů aj.) pro mobilní operátory a výrobce zařízení a v neposlední řadě pro vývojáře aplikací poskytuje efektivní nástroje pro jejich vývoj – Software Development Kit. Platforma Android měla v srpnu 2013 bezmála 80% podíl na trhu s chytrými telefony.

Architektura operačního systému Android je rozdělena do 5 vrstev. Každá vrstva má svůj účel a nemusí být přímo oddělena od ostatních vrstev. Nejnižší vrstva architektury je jádro operačního systému, které tvoří abstraktní vrstvu mezi používaným hardwarem a zbytkem softwaru ve vyšších vrstvách. Jádro systému Android je postaveno na jádře systému Linux. V prvních verzích Androidu šlo o verze Linuxu 2.6.x, pozdější verze využívají novější linuxová jádra. Je využito mnoha vlastností Linuxu, jako podpora správy paměti, správa sítí, zabudované ovladače nebo správy procesů, například souběžného běhu aplikací, které běží jako samostatné procesy s oprávněním stanoveným systémem. Tato vlastnost přispívá ke stabilitě a také ochraně systému. Naopak systém nepodporuje grafické uživatelské rozhraní X Window System a ani úplnou sadu GNU knihoven. Důvodem použití jádra Linux byla také vlastnost poměrně snadného sestavení na různých zařízeních a tím zaručená přenositelnost. Další vrstvou jsou knihovny, které jsou napsány v C nebo C++ kódu a využívají je různé komponenty systému. Tyto funkce jsou vývojářům poskytnuty prostřednictvím Android Application Framework.

Android není zcela otevřený systém, ale otevřená je jeho velká část: od modulů Linuxového jádra, knihoven, aplikačního programové rozhraní až po základní aplikace. To dává výrobcům možnost přizpůsobit systém nasazenému hardwaru a také lépe integrovat své aplikace. Android je zároveň šířen pod „business-friendly“ licencí (Apache/MIT), což umožňuje třetím stranám využívat systém pro nejrůznější účely. Ale některé části jsou uzavřenou technologii firmy Google, přesto Google tento systém prezentuje jako open source. Android také obsahuje několik patentovaných technologii, které si musí výrobci telefonů licencovat. Díky vydání produktu jako open source a nezávislosti na použitém hardwaru může být Android spuštěn na velkém počtu přístrojů. Tato skutečnost zapříčinila vznik velkého a ekonomicky úspěšného ekosystému tvořeného aplikacemi, vývojáři a uživateli.

3.10.2. IOS

iOS je mobilní operační systém vytvořený společností Apple Inc. Původně byl určen pouze pro mobilní telefony iPhone, později se však začal používat i na dalších mobilních zařízeních této firmy, jako jsou iPod Touch, iPad a nejnověji Apple TV.

iOS je odlehčenou verzí operačního systému Mac OS X, používaného v počítačích společnosti Apple. Jedná se tedy o systém UNIXového typu. Jelikož je určen pro mobilní zařízení, neobsahuje veškerou funkcionalitu OS X, na druhou stranu ale přidává podporu dotykového ovládání. Systém se dělí na čtyři základní vrstvy, které zajišťují základní funkčnost a poskytují vývojářům API a frameworky potřebné k vývoji aplikací.

Až do verze iOS 4.0 nebylo možné psát aplikace, které by byly schopné běhu na pozadí. Stisk tlačítka Home vyvolal ukončení aplikace. iOS 4.0 sice nepřinesl možnost plnohodnotného běhu na pozadí, avšak poskytuje služby, které umožňují provádění některých činností, i když aplikace neběží na popředí.

V iOS je možné spouště aplikace napsané v jazyku C nebo pokročilejším Objective-C. Dlouhou dobu bylo možné vyvíjet pouze v aplikaci XCode, což je vývojové prostředí od firmy Apple (nabízené zdarma). Toto prostředí je však dostupné pouze pro operační systém Mac OS X, takže vývoj např. ve Windows či Linuxu není možný. Tento problém se pokusilo řešit několik projektů, které se snažily kompilovat programy napsané v jiných jazycích do nativního kódu Objective-C. Asi největším počinem v této oblasti je krok společnosti Adobe, která v nové verzi svého nástroje pro vývoj aplikací Flash umožňuje kompilovat právě do programu určeného pro iOS. Tento (a podobné) nástroje však byly zakázány v licenčním ujednání, ale po velké nevoli ze strany vývojářů byly opět povoleny.

3.10.3. Symbian

Symbian OS je operační systém, který byl navržen pro využití v mobilních zařízeních (tzv. „chytrých telefonech“ neboli smartphonech). Doplňují ho knihovny, grafické uživatelské rozhraní a referenční implementace nástrojů, které vytvořila firma Symbian Ltd. Symbian OS je následovníkem systému EPOC používaného v kapesních počítačích Psion a běží výhradně na procesorech ARM. V současné době se jedná o operační systém reálného času. V období únor 2010 až březen 2011 používal licenci EPL a byl tedy otevřený software. Předtím a potom používal uzavřenou licenci. Do systému může uživatel přidávat nativní aplikace, které však závisí na verzi použitého operačního systému. Symbian je dnes používán především v mobilních telefonech značky Nokia. 11. února 2011 Nokia oznámila ústup od dalšího vývoje Symbianu a přechod k platformě Windows Phone 7 a 8. Koncem roku 2012 také Nokia ukončila provoz stránky http://symbian.nokia.com.

V roce 1998 se spojily firmy Motorola, Ericsson, Nokia a Psion, aby vytvořili sdružení Symbian, které prosazuje používání operačního systému EPOC navrženého specificky pro mobilní zařízení. Připojení společnosti iMatsushita znamenalo, že Symbian představuje 85 % světového objemu prodeje chytrých mobilních telefonů. Symbian je od prosince 2008 vlastněn firmou Nokia. Ta na svém barcelonském kongresu oznámila vykoupení zbývajících 52 % akcií, takže nyní vlastní více než 99.9 % akcií. (Předchozí zastoupení 47,9 %, Ericsson 15,6 %, Sony Ericsson 13,1 %, Panasonic 10,5 %, Siemens AG 8,4 % a Samsung 4,5 %.) Vznikla také organizace Symbian Foundation, jejíž členové (Nokia, Motorola, Ericsson, LG, Samsung a další) chtějí vytvořit jednotnou a otevřenou platformu pro mobilní telefony. Kód systému byl pozvolně uvolňován pod licencí Eclipse Public License (EPL), který byl dokončen 4. února 2010. Některé části však dříve, například v říjnu 2009 byl otevřen EKA2 mikrokernel Symbianu. Při uvedení aktualizace Belle Nokia de facto značka Symbian zaniká, protože se jej Nokia rozhodla komunikovat pouze jako Nokia Belle. Už kolem roku 2011 se ale začal Symbian mezi ostatními systémy (iOS a Android) propadat. To nakonec způsobilo, že Nokia, která tomuto systému

plně důvěřovala, začala mít velké ztráty. To jí dokonce donutilo prodat její sídlo ve Finsku. V současné době je Nokia oblíbená hlavně v Indii.

3.10.4. Windows mobile

Windows Mobile je operační systém firmy Microsoft, který je založen na Windows CE. Je určen pro mobilní zařízení (PDA, smartphony a zařízení portable media center). Používá vzhled odvozený od klasických Microsoft Windows a malou podmnožinu jejich Win32 API, avšak obsahuje zcela jiné hybridní jádro. Nástupcem systému Windows Mobile je systém Windows Phone.

Sám jsem několik let používal a dnes používám spíše svátečně jedno MDA od hp, které pracuje na WM 6.1 Professional. tento systém je zcela bezúdržbový, pohodový a stabilní a rychlý, jedinou „nevýhodou“ je, že člověk při jeho obsluze musí před sebou vidět počítač, nikoli telefon. Ovládání WM 6.1 je dost podobné ovládání Windows XP.

Windows Mobile má ve většině verzí tyto rysy:

- Obrazovka Dnes ukazuje datum, informace o vlastníkovi, nadcházející události, e-mailové zprávy a úkoly. Uživatelé mohou upravit, které informace se mají na obrazovce zobrazovat. Na obrazovce se nachází rovněž stavové ikony Bluetooth, Wi-Fi atd. Programy nainstalované v zařízení mohou přidat na obrazovku další položky. Obrázek na pozadí lze změnit přímo v zařízení nebo lze změnit celé téma.
- Horní pruh ukazuje čas, hlasitost a stav připojení. Hlavní součástí horního pruhu je tlačítko Start, které je podobné tomu z desktopové verze Windows. Start menu zahrnuje seznam nedávno spuštěných programů, devět upravitelných položek menu a odkazy na seznam programů, nastavení, vyhledávání a nápovědu.
- Microsoft Office Mobile je sada mobilních verzí aplikací Microsoft Office zahrnutých ve Windows Mobile. Zahrnuje Word Mobile, Excel Mobile a PowerPoint Mobile (od verze Windows Mobile 5.0). Mobilní verze mají mnoho rysů desktopových verzí, ale některé funkce byly značně redukovány.
- Outlook Mobile zahrnuje úkoly, kalendář, kontakty a poštu, které lze synchronizovat s aplikací Outlook nebo s přímo s Exchange Serverem.
- Windows Media Player pro Windows Mobile podporuje formáty .WMA, .WMV, .MP3 a .AVI.
- Klient pro PPTP VPN.

3.10.4.1. 6.x

Windows Mobile 6, známý dříve pod kódovým označením Crossbow, je nejnovější verze tohoto mobilního operačního systému, která byla představena 12. února 2007 na 3GSM World Congress 2007. Existují celkem tři verze systému: Windows Mobile 6 Standard pro smartphony, Windows Mobile 6 Professional pro PDA s integrovaným telefonem (Pocket PC Phone Edition) a Windows Mobile 6 Classic pro PDA bez telefonního modulu.

Windows Mobile 6 je úzce propojen s Windows Live and Exchange 2007. Windows Mobile 6 Standard poprvé nabídl Orange SPV E650, zatímco Windows Mobile 6 Professional se nejdříve stal součástí O2 Xda Terra.

Přehled nových vlastností Windows Mobile 6:

- Podpora rozlišení 800x480 (WVGA).
- Schopnost upravovat dokumenty Office na smartphonech.
- Schopnost automatických aktualizací systému.
- Zlepšení přístupu přes vzdálenou plochu.
- Podpora VoIP.
- Integrace Windows Live.

- Šifrování dat na výměnné paměťové kartě.
- Smartfilter pro rychlejší vyhledávání.
- Podpora prohlížení HTML e-mailů v Outlook Mobile.
- Schopnost hledat v Exchange Server Address Book.
- Podpora AJAX, JavaScript a XML DOM v Windows Internet Explorer Mobile.

3.10.5. Windows Phone

Windows Phone je obchodní název mobilního operačního systému firmy Microsoft. Jedná se o nástupce systému Windows Mobile, se kterým však není zpětně kompatibilní. Windows Phone byl poprvé vydán 21. října 2010. V současné době jsou jako Windows Phone označovány systémy Windows Phone 7 a Windows Phone 8, které však nejsou vzájemně kompatibilní a dokonce interně obsahují různé platformy (Windows CE a nově pak přizpůsobené jádro z řady Windows NT). V současné době se i na starší modely s WP 8 nasazuje nový WP 8.1, který si osobně nemohu pro jeho rychlost, nenáročnost a možnosti vynachválit.

Windows Phone nabízí nové uživatelské rozhraní založené na Microsoft Windows Phone konstrukčního systému s kódovým názvem Metro. Domovská obrazovka s názvem "Úvodní obrazovka" se skládá z dlaždic. Dlaždice jsou odkazy na aplikace, vlastnosti, funkce a jednotlivé položky (například kontakty, webové stránky, aplikace nebo mediální položky). Uživatelé mohou přidávat, měnit pořadí, nebo odstranit dlaždice. Dlaždice jsou dynamické a aktualizují se v reálném čase, např. taška pro e-mailového klienta zobrazí počet nepřečtených zpráv nebo Deska může zobrazovat aktualizaci počasí. (Viz Windows 8 a 8.1)

Internet Explorer ve Windows Phone umožňuje uživateli udržovat seznam oblíbených webových stránek a dlaždic, které odkazují na webové stránky na úvodní obrazovku. Prohlížeč podporuje až 6 záložek, které lze načíst všechny současně. Mezi další vlastnosti patří multitouch gesta, hladké zvětšení / zmenšení animací, možnost ukládat snímky, které jsou na webových stránkách, sdílení webových stránek prostřednictvím e-mailu a podporu pro on-line vyhledávání, které umožňuje uživateli vyhledávat slova nebo fráze na webové stránce napsáním. Společnost Microsoft oznámila plány pravidelně aktualizovat Windows Phone webový prohlížeč nezávisle na aktualizaci systému Windows Phone.

Systém Windows Phone umožňuje komunikaci přes "vlákna", propojení se se svými kontakty pomocí Windows Live Messenger a Facebook Chat, stejně jako tradiční textové zprávy. Textová zpráva může být složena pomocí hlasového ovládání. Rozpoznávání hlasu umožňuje, která řeč má být převedena na textové zprávy, dále umožňuje textovou zprávu převést do řeči, které lze číst nahlas.

Windows Phone podporuje Hotmail, Exchange, Yahoo! Mail a Gmail nativně a podporuje řadu dalších služeb přes POP a IMAP protokoly. Původní typy účtů, kontaktů a kalendář mohou být spolu synchronizovány. Uživatelé mohou prohledávat své e-maily prostřednictvím vyhledávání v tomto tématu. E-maily jsou zobrazeny klasicky a více e-mailové schránky došlé pošty lze kombinovat.

Store nahrazuje Windows Marketplace ve verzi Windows Phone 8. Store nabízí jen aplikace a hry. O digitální distribuci hudby se na platformě Windows Phone 8 stará aplikace od společnosti Microsoft Xbox Music.. Pro uživatele telefonů Lumia je také k dispozici Nokia MixRadio, tato aplikace není k dispozici v České republice.

Microsoft Office Mobile poskytuje interoperabilitu mezi Windows Phone a stolní verzí Microsoft Office. Word Mobile, Excel Mobile, PowerPoint Mobile, OneNote Mobile a SharePoint Workspace Mobile dovolí většinu formátů Microsoft Office souborů, je třeba sledovat a upravovat přímo v zařízení Windows Phone. Microsoft Office soubory z OneDrive a Office 365, stejně jako soubory uložené lokálně na telefonu, lze přistupovat pomocí Office Hub.

Office soubory jsou seřazeny podle dlaždic: dokumenty Word (modrá taška), excelovské tabulky (zelená taška), PowerPoint prezentace (červená taška), a OneNote dokumenty (fialová taška).

3.10.5.1. WP 8.1

- Přidáno notifikační centrum
- Hlasová asistentka Cortana
- Vlastní pozadí dlaždic, možnost výběru vlastní, nebo již integrované fotografie
- Lepší správa zvuku
- Přidána Swype klávesnice, která (zatím) nepodporuje český slovník
- Možnost přidání třetího sloupce, aniž by proto byla úhlopříčka displeje uzpůsobena
- Ovládací tlačítka (šipka zpět, tlačítko Windows a tlačítko hledat) mohou být zobrazeny na displeji
- Kalendář: jeho vylepšení přináší lepší zobrazení událostí a integraci předpovědi počasí
- Aplikace mohou být instalovány na paměťovou kartu
- Internet Explorer 11 přinesl plnou podporu HTML5 videí
- možnost stahovat libovolné soubory do složek
- správce hesel
- možnost soukromého prohlížení
- synchronizaci se záložkami s Windows 8
- možnost otevření souborů před stažením
- podpora gest
- Nové prostředí fotoaparátu
- Optimalizace pro větší displeje
- Lepší podpora služeb od společnosti Google (Gmail, YouTube...)
- Data Sense (Inteligentní data), WiFi Sense (Inteligetní WiFi), Storage Sense (Inteligentní úložiště) - vylepšení vzhledu a možností u jednotlivých položek
- Spořič baterie - přidány funkce, jako zobrazení aplikací, které nejvíce vybíjejí baterii aj.
- Hudba, Video, FM rádio a kalendář jsou nyní samostatnými aplikacemi, které mohou být aktualizovány ze Store
- Store byl upraven do lepší a přehlednější podoby
- V SMS zprávách lze nyní vyhledávat
- Přibyla funkce s možností synchronizace motivu, nastavení aplikací, Internet Exploreru a hesel s dalšími zařízeními s Windows 8
- Podpora DualSIM



3.11.Přehled alternativních OS

Jde o systémy, které vyvíjely původně velké firmy, nebo university, čí parta popřípadě jeden programátor. Většinou jsou dobré jako záložní systémy, nebo druhé systémy pro mladé lidi, kteří rádi experimentují a také se tak trochu u svých PC nudí.

Uvádím výčtem některé z nich: Sky OS, FreeBSD, BeOS5, AROS - Amiga Research OS, AtheOS, Caldera DR-DOS, Cefarix, DS-OS, ELKS, FreeDOS, MenuetOS, Minix, NewDeal – nadstavbaOS, Oberon, PTS-DOS…

3.12.Závěrem

Doufám, že se vám dle mého nudná teoretická prohlídka operačních systémů, které jsou hudbou dní minulých, nebo i současných alespoň trošku líbila. Po dnešku je jasné, že Win8.1 nám nespadl jen tak z nebe, ale předcházela tomu velmi dlouhá a trnitá cesta vývoje. Jako takové ozvláštnění této kapitoly bych mohl doporučit virtualizaci a zkusit si některé archaické, ale i současné systémy na vlastní pěst, třeba objevíte nové zajímavé dimenze světa počítačů.

4. Microsoft Office 2013

Jde o vlajkovou loď sad Office vyvíjených od roku 1987, která vyšla ve verzích: home and student, profesional, profesional plus a pro vysokoškoláky. Slouží k práci s textem, fotografiemi, e-mailem a databázemi.

Přehled produktů obsažených v jednotlivých verzích:

- Home: Word, Excel, PowerPoint a Onenote, Picture manager, uplode centr
- Pro podnikatele: to samé co Home + Outlook
- Professional: to samé co Home + Publicisher, Access, Outlook

V této kapitole si ukážeme základy práce s trojicí produktů a to: Word, PowerPoint a Excel. Poradím vám, jak donutit tyto produkty pracovat takovým způsobem, aby editace a vlastní napsání diplomové či seminární práce nebylo pro vás nic těžkého. Řekl bych, že s výjimkou webových prohlížečů a multimediálních přehrávačů patří právě sada Office k nejpoužívanějším aplikacím.

Ještě, nežli se pustíme do díla, a že doje i na vás, již nebudu mluvit jen já, tak si pojďme v krátkém výčtu představit novinky našeho Office.

- Možnost ukončení a minimalizace, popřípadě stornování načítání souboru již ve spouštěcím okně
- Aktivní práce se soubory pdf ve dvou stupních komprese a možnosti převodu pdf do formátu doc, docx, xps, html, rtf a dalších
- Okamžité odeslání prostřednictvím MS Outlook ve formátu pdf, bez nutnosti uložení
- Možnost integrace dalších aplikací (např. PC Translátor)
- Spolupráce s webovou službou WorksSpace
- Bez instalační sada pro uživatele hotmail, či windowslive a propojení s plnou sadou
- Možnost instalace zkušební verze přímo z hotmailu
- Možnost odesílání online dokumentů pomocí hotmail
- Podpora formátu xps
- Detekce virově poškozených souborů
- Chráněný režim pro čtení dokumentů, které jsou stažené z webu
- Podpora videa i ve Wordu
- Tvorba jednoduchých aplikací pro Office
- Vylepšená podpora open office
- Propojení návrhů a šablon v různých programech sady (šablonu z PowerPointu mohu užít ve Wordu a podobně)

Také bych vám neměl tajit nároky a licencování této aplikace. Požadavky sady jsou:

- ✓ Operační systém Windows 8.1, 8, 7, nebo OSX Lion, či Windows Server 2008 R2 nebo Windows Server 2012
- ✓ NET Framework 3.5 nebo vyšší
- ✓ CPU: 1GHz s podporou instrukcí SSE2
- ✓ HDD: 3GB
- ✓ RAM: 1024 (x86) / 2048 (x64)
- ✓ Monitor s rozlišením 1366 x 768 nebo vyšším
- ✓ Grafická karta s Direct X 10 nebo vyšším
- ✓ Internetový prohlížeč: Microsoft Internet Explorer 8, 9, 10 nebo 11; Mozilla Firefox 10.x nebo vyšší; Apple Safari 5 nebo Google Chrome 17.x.

Asi by se slušelo říct, že pokud nemáte na to, abyste si zakoupili Office v obchodě, tak máte dvě možnosti, tou první je užití trial verze, která lze stáhnout z hotmailu, nebo využívat přímo online verzi, která bohužel má omezené možnosti zpracování.[1]

4.1. Word

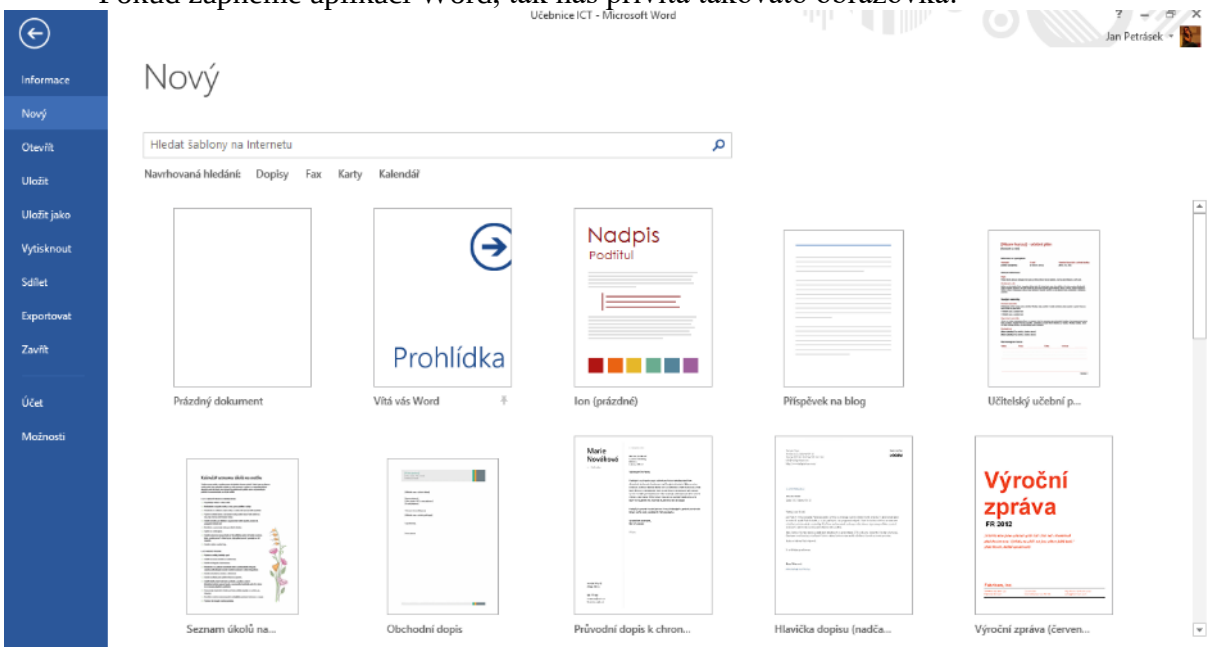
Jak již název programu, a to jsem angličtinář jako poleno, napovídá, jedná se o textový editor. V případě naší verze jak si ukážeme, to není zcela pravda, Word sice znamená slovo, ale Word 2013 umí i video, má velmi výkonné nástroje na úpravu obrázků a zvládá toho ještě o hodně více.

Asi by někoho z vás mohlo napadnout, proč to nejlehčí a nejužitečnější řadím až do čtvrté kapitoly, moje odpověď je prostá, bez znalosti předchozích kapitol by nebylo možné pochopit základní věci jako například požadavky aplikace, tedy by nebylo možné ji zcela správně instalovat, protože Word je pouze jeden z modulů sady Office.

4.1.1. Krátké představení

Hned na začátek jsem vám dal nepsaný úkol, nainstalovat sadu Office, zatím žádné další úkoly nečekejme, ukážeme si nejprve, kde, co v nové verzi je, protože je tam spousta změn hlavně oproti sadám 2003 a 2007, ale drobné změny jsou i proti sadě 2010.

Pokud zapneme aplikaci Word, tak nás přivítá takováto obrazovka:

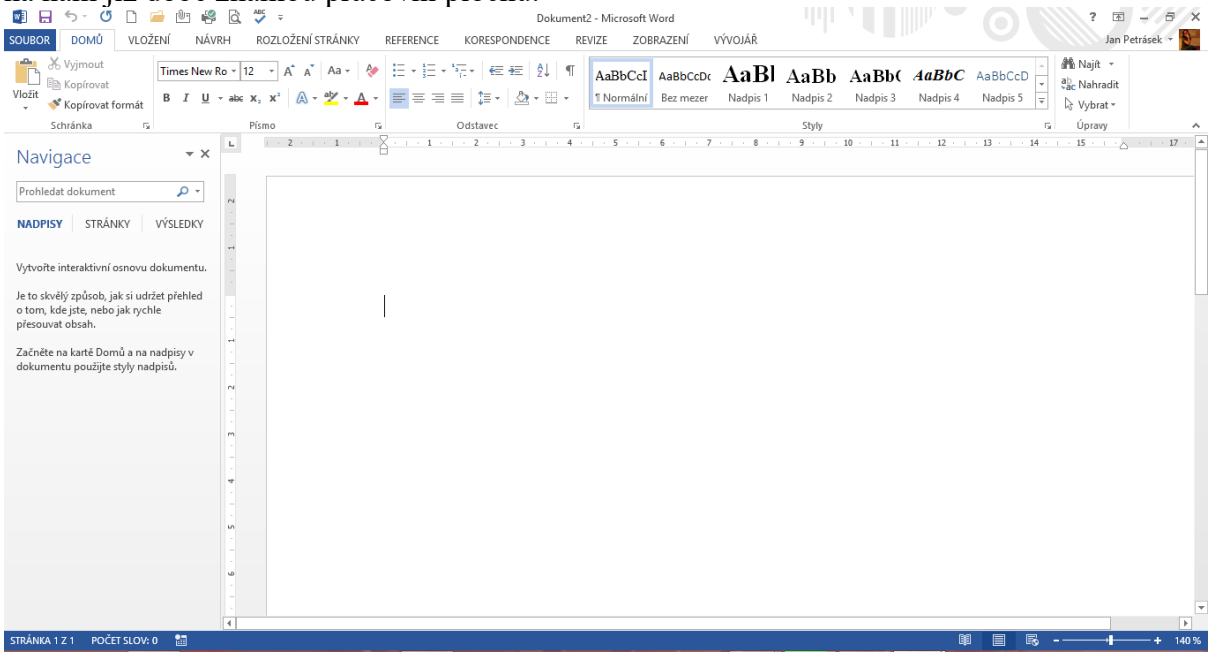


Kde je pro nás zásadních několik věcí:

1. V pravém horním rohu přihlášení k účtu Microsoft, které umožňuje práci s online uložištěm OneDrive
2. Hlavní část strany, kde se nám načtou základní šablony dokumentů
3. Vyhledávání pro nalezení šablony, pokud pro oči nevidíme, nebo pokud je šablona online
4. Poslední položka v menu na levé straně s názvem možnosti, která umožňuje provádět nastavení programu dle našich potřeb

[1] Pokud užíváme online app, tak se nemusíme starat o aktualizace, pokud jsme instalovali produkt na svůj počítač, tak je nutné restartovat systém (ač to instalátor nevyžaduje) a následně instalovat aktualizace, aby se naistaloval service pack.

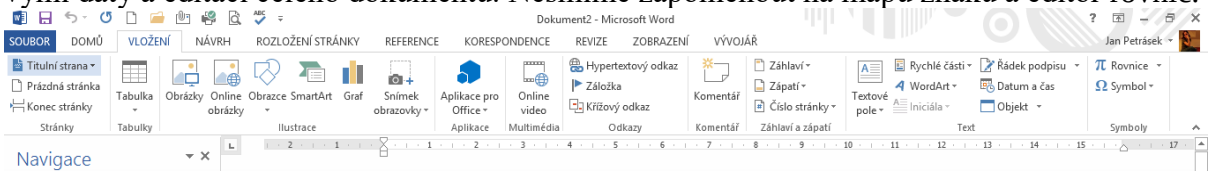
Pokud klikneme na prázdný dokument, což je hned první šablona, tak nás Word přenese na nám již době známou pracovní plochu.



Máme otevřenou záložku domů, kde nalezneme všechno pro editaci textu. Mám k dispozici (řazeno zleva):

- Schránka (kopírovat, kopírovat formát, vyjmout, vložit (rozbalovací možnost)
- Písmo (fonty, podtržení, velikost, ztučnění, kurzíva, přeškrtnutí, horní idex, dolní index, barva písma, zvýraznění, vymazání formátování, zvětšení/zmenšení písma, textové efekty)
- Odstavec (odrážky, číslování, zarovnání, zmenšení/zvětšení odsazení, řádkování, stínování, seřazení, zobrazení netisknutelných znaků, ohraničení)
- Styly (budeme se jim věnovat samostatně)
- Úpravy (najít, nahradit, vybrat)

Pokud přejdeme na kartu vložení, tak máme k dispozici položky pro práci s netextovými daty a editaci celého dokumentu. Nesmíme zapomenout na mapu znaků a editor rovnic.



Pojďme si tedy tuto lištu představit podrobně (řazeno zleva):

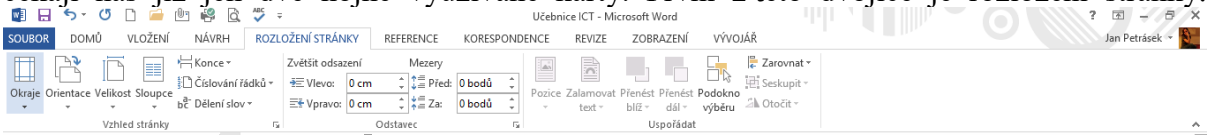
- Stránky (vložení či vytvoření titulní strany, vložení prázdné strany a konce strany)
- Tabulky (nástroj k vytvoření tabulky)
- Ilustrace (obrázky a obrázky online, smartArt (obrazce), graf, snímek obrazovky)
- Aplikace (aplikace pro MS Office)
- Multimedia (video ve formátu flash)
- Odkazy (hypertextové odkazy, záložky, křížové odkazy)
- Komentář
- Záhlaví a zápatí (záhlaví, zápatí, čísla stránek)
- Text (textové pole, rychlé části, iniciály, grafická písma, řádek podpisu, datum a čas, objekt, text ze souboru (v nabídce objekt))
- Symboly (symbol (mapa znaků), rovnice (editor rovnic a předdefinované rovnice)

Další karta nám připomene spíše aplikaci PowerPoint, jde o kartu Návrh.



Zde si můžeme vybrat z předdefinovaných stylů formátování dokumentů a jejich efektů, najdeme zde i položky ohraničení a barva strany. Tuto kartu si zamilujete, pokud publikujete dokumenty v pdf nebo xps na webu, nebo využijete k vytvoření webové stránky právě Word. Najdeme zde položku vodoznak, která nám zajistí, že je vidět, odkud dokument je a není možné, aby jej někdo jako celek vytiskl a vydával za svůj, či publikoval dále.

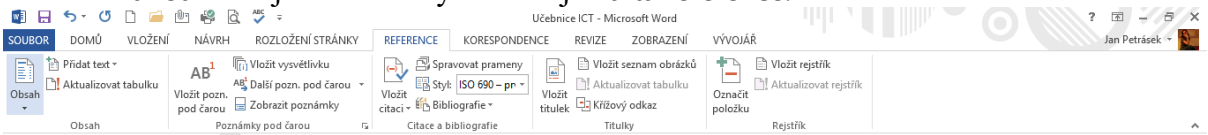
Nebojte se, že by naše cesta základním rozložením Wordu byla ještě nějak zdlouhavá, čekají nás již jen dvě hojně využívané karty. První z této dvojice je rozložení stránky.



Zde nalezneme důležité nastavení dokumentu především k tisku. Pojďme si tedy představit její části:

- Vzhled stránky (okraje, orientace strany, velikost (formát papíru), sloupce (rozdělení obsahu oddílu do sloupců podobně jako v novinách), konce (konec strany, sloupce, obtékání textu, konce oddílu), číslování řádků (hodí se pokud píšeme nějaký protokol obsahující kód v programovacím jazyce), dělení slov)
- Odstavec (podrobné nastavení odsazení a mezer před a za řádkem)
- Uspořádat (pracuje s netextovými daty)

Druhou z dvojice slibovaných karet je karta reference.



Pokud se na tuto katru lépe podíváme, tak zde nalezneme:

- Obsah (tvorba a úprava obsahu)
- Poznámky pod čarou (tvorba poznámek a vysvětlivek)
- Citace a bibliografie (nastavení normy a vložení citací, správa pramenů)
- Titulky
- Rejstřík (tvorba rejstříku, označení položek)

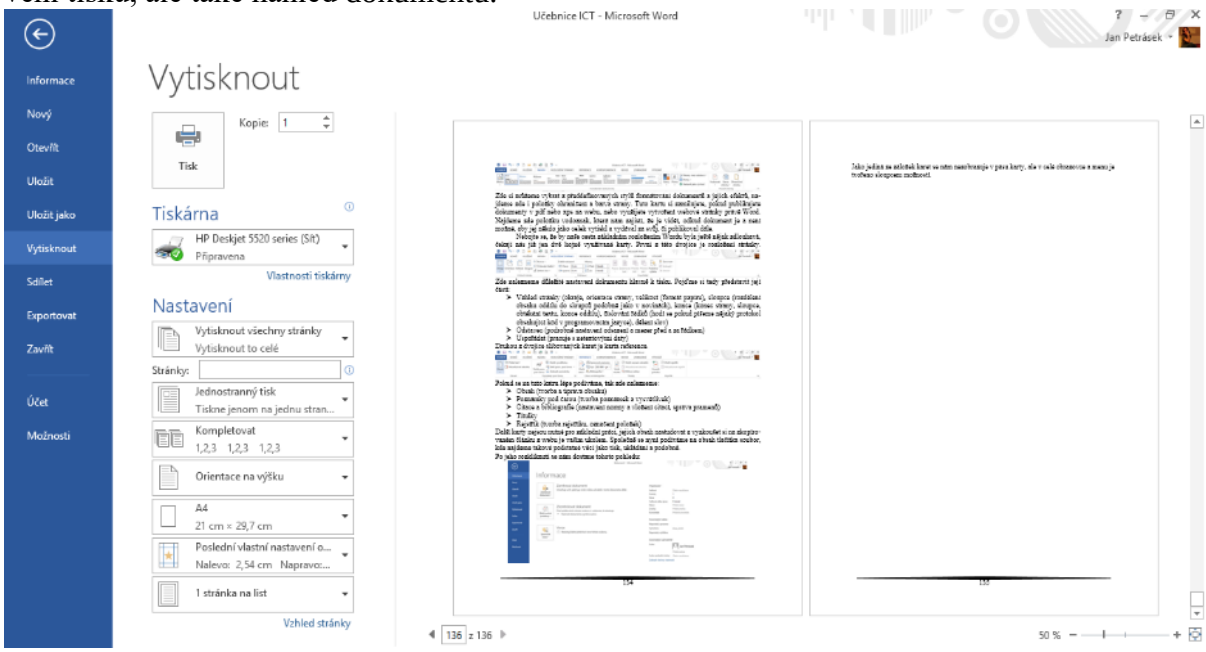
Další karty nejsou nutné pro základní práci, nastudovat jejich obsah a vyzkoušet si jej na zkopírovaném článku z webu je vaším úkolem. Společně se nyní podíváme na obsah tlačítka soubor, kde najdeme takové podstatné věci jako tisk, ukládání a podobně.

Po jeho rozkliknuti se nám dostane tohoto pohledu:

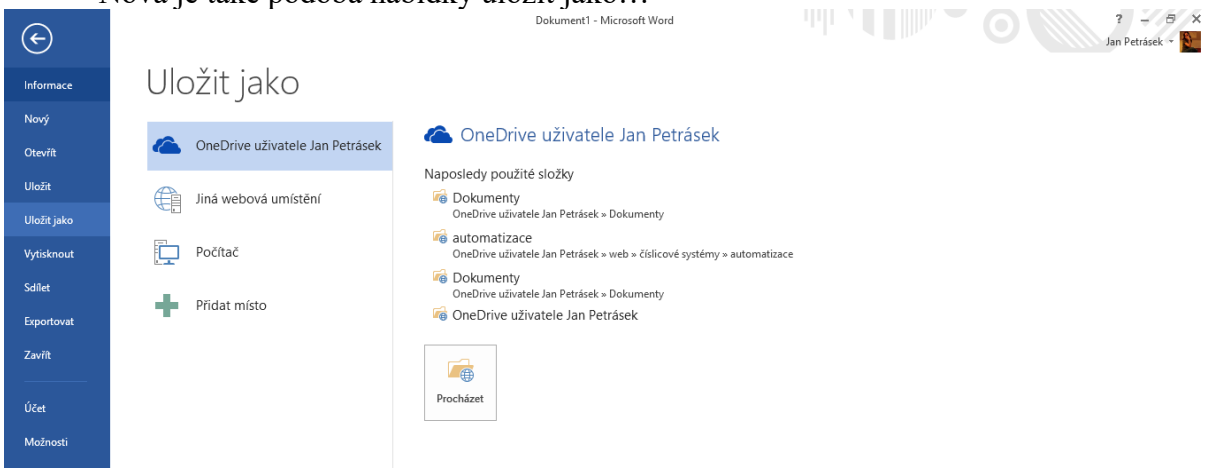


Jako jediná ze záložek karet se nám nezobrazuje v pásu karty, ale v celé obrazovce a menu je tvořeno sloupcem možností.

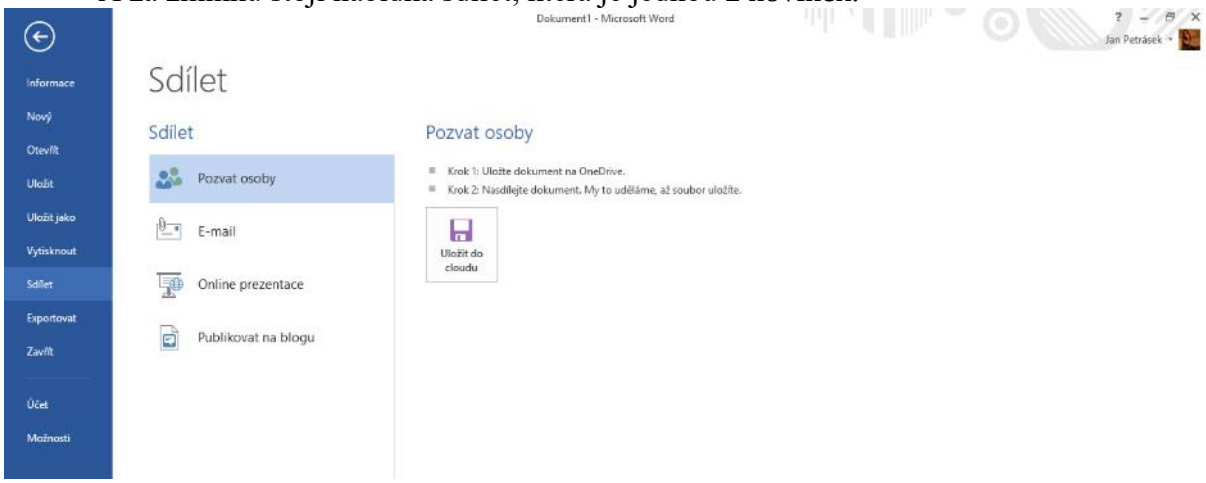
Jednou z nejdůležitějších nabídek pro nás je tisk. Nabídka tisku skrývá nejen nastavení tisku, ale také náhled dokumentu.



Nová je také podoba nabídky uložit jako…



A za zmínku stojí nabídka sdílet, která je jednou z novinek.



Nabídka Možnosti nás přesune do okna **nastavení** Wordu a nabídka účet slouží pro přihlášení k účtu outlook.com, který zde obstarává integraci emailu a přihlášení ke službě on-drive.

Kulatá šipka nahoře v kroužku je návrat do editace dokumentu.

Tímto bych zakončil teoretickou procházku programu MS Word, je nyní na každém z nás, aby se k programu vrátil, jednotlivé možnosti si vyzkoušel a případně užil nápovědu, nebo se podíval dále do této knihy.

4.1.2. Formální úprava dokumentů

Nicméně u teorie ještě zůstaneme. Nežli se pustíme do nastavování a vypracovávání prvních úkolů, které nám pomohou si aplikaci osvojit, tak bychom si nejprve měli říci něco o normách, které platí pro tvorbu dokumentů. Některé z nich má Word nastaveny implicitně, jiné si musíme nastavit sami. Přemýšlel jsem, zda vás zatěžovat názvy norem a dospěl jsem k názoru, že se jedná o naprostou zbytečnost, takže si jen řekneme, co je potřeba dodržovat, co vlastně z těch všech norem vyplývá. Pokud nebude přímo v zadání úkolu řečeno jinak, tak vždy dodržujeme tyto zásady. Tak si je tedy pojďme představit.

4.1.2.1. části práce (seminární, diplomová apod.)

- Titulní strana
- Prohlášení – povinný text na samostatné stránce, musí být vlastnoručně podepsán.
- Poděkování – volitelná položka, krátké poděkování osobám či organizacím, které se na výsledku práce podílely (vedoucí práce, konzultanti, firma), a obvykle se nečísluje strana.
- Obsah – Formální úprava je stejná jako u ostatních kapitol. Skládá se z titulku „obsah“ a seznamu hesel, obsahujících název příslušných kapitol, podkapitol a odpovídající číslo stránky. Příslušnost čísel k heslům může být zvýrazněna vodícími znaky, například tečkami. Na konci obsahu se uvádí výčet všech příloh. Obsah má být stručný a přehledný. Je automaticky generován na základě stylů.
- Úvod – první kapitola obsahuje zadání práce, cíl práce, plánovaný postup řešení a případně hypotézu výsledků. Objeví se zde charakteristika obsahu dalších kapitol. Také jsou vyjmenovány přílohy a popsána jejich funkce.
- Stať – Teoretická část – vlastní text práce, rozčleněný na kapitoly, podkapitoly atd., přehled dosavadního zkoumání daného problému, utřídění poznatků, srovnání pojetí různých autorů, vymezení základních pojmů.
- Stať – Praktická část – vlastní výzkum, případně historicko-srovnávací či genetický popis, dotazník, grafy, tabulky, nákresy, fotografie, statistika, interview atd. – v této části dokládáme či aplikujeme poznatky z teoretické části v praxi. Tato část by měla rovněž obsahovat diskusi výsledků zkoumání.
- Závěr – zde jsou shrnuty dosažené výsledky a je uvedeno jejich případné využití, práce je zhodnocena z hlediska stanovených cílů, jsou uvedeny podněty pro další studium a výzkum řešené problematiky.
- Poznámky (pokud nebyly uváděny pod čarou nebo na konci kapitol)
- Odkazy (pokud nebyly uváděny v poznámkách nebo v textu práce)
- Seznam pramenů (pokud bylo použito větší množství pramenů)
- Seznam literatury – píše se na samostatnou stránku a má formální úpravu kapitol. Za titulkem následuje seznam veškerých pramenů, které byly použity k vypracování práce. Hesla v seznamu jsou řazena abecedně podle příjmení autora. Bývají opatřena návěštím (obvykle pořadovým číslem), používaným v odkazech na publikace. Bibliografické údaje jsou uváděny podle norem pro bibliografickou citaci ČSN ISO 690-1 a

690-2. Nezapomeňte! Správná bibliografická citace vyžaduje celou řadu údajů. Tyto údaje je třeba pečlivě shromažďovat v průběhu celého zpracování seminární práce, například zavedením záznamů o prostudované literatuře.

- Seznam zkratek (pokud byly použity nestandardní zkratky)
- Seznam značek, symbolů aj. (pokud byly použity nestandardní značky)
- Seznam obrázků (Doporučené je zařazovat obrázky formátů JPG, JPEG, nedoporučuje se bmp a nepřípustný je animovaný gif.), grafů, schémat atd. (pokud byly zařazeny do textu)
- Přílohy – nezapočítávají se do rozsahu práce, mohou být svázány v dokumentu, mají stejný formát jako text práce, – připojeny volně, například barevná grafika, soubor fotografií, diapozitivů, – umístěny do kapsy na vnitřní straně zadní desky vazby, například diskety, CD-ROM, – svázány do samostatného svazku (při velkém rozsahu)
    - výčet všech příloh je nutno uvést v obsahu a zmínit se o nich v úvodu.

4.1.2.2. Sazba textu

Dokument je tištěn na listech velikosti A4 po jedné nebo po obou stranách. Všechny listy jsou z pravidla z téhož druhu papíru. Okraje stránek je výhodné nastavit před započetím vlastního psaní. Ušetříte si tím spoustu komplikací se změnou velikostí tabulek, obrázků, grafů.

4.1.2.2.1. Okraje stránky

horní okraj 3,0 cm

dolní okraj 5,0 cm

levý okraj 3,5 cm

pravý okraj 4,0 cm

4.1.2.2.2. Číslování stránek

Číslujeme všechny platné stránky arabskými čísly (Římské číslice jsou v normách vyhrazeny pro podkapitoly) počínaje první fyzickou stránkou seminární práce. Na některých stránkách se číslo nezobrazuje, např. titulní strana, poděkování, prohlášení. Přílohy nečíslujeme nebo mohou být číslovány zvlášť. Stránky číslujeme arabskými čísly obvykle dole uprostřed.

4.1.2.2.3. Volba typu písma

Textové editory nabízejí neuvěřitelné možnosti v používání písma. Zhruba lze rozlišit písma (fonty (Normovánými základním fontem je Times New Roman, Arial)):

- patková (Times…)
- bezpatková (Arial …)
- kaligrafická a volně psaná (Gigi, Bradley …)

Řezy písma: obyčejné, tučné, kurzíva, tučná kurzíva.

Lze sázet písmo: obyčejné, rozšířené, zúžené.

Lze volit velikosti písma: Times 10 bodů... Times 12 bodů (Je základní normovanou velikostí pro běžný text stati, nikoliv pro nadpisy!!)….Times 16 bodů.

Platí, že méně bývá více. Nekombinujete v jedné práci více písem, nestřídejte často efektní řezy písma.

Doporučuji

Nadpisy, tituly kapitol: 22/24 bodů (nadpis 1), 18/20 bodů (nadpis2), 16/18 (nadpis 3), 14/16 bodů (nadpis 2)

Vlastní text práce Times 12 bodů

Poznámky Times 10 bodů

Zvýraznění části textu doporučujeme tučně, kurzívou, tučnou kurzívou, proložením (nikoliv vkládáním mezer mezi písmena). Nepoužíváme zvýraznění textu podtržením (Běžné je toto zvýraznění pro hypertextový odkaz, jiný druh textu jím nezvýrazňujeme).

4.1.2.2.4. Řádkování

Lze volit jednoduché řádkování nebo 1,5 řádku. Při jednoduchém řádkování se mezi odstavci řádkuje dvakrát, při 1,5 řádkování se mezi odstavci řádkuje jedenkrát. Nadpisy se od předcházejícího textu oddělují dvěma řádky, od následujícího textu jedním prázdným řádkem.

4.1.2.2.5. Dělení slov

Dělení slov se řídí pravopisnými a estetickými pravidly. Není nutné, při pracích menšího rozsahu, jinak dle zmiňovaných pravidel. Z estetických pravidel dodržujte zásadu, že rozdělením slova nesmí vzniknout vulgarimus (spisova-tele, své-vole, ná-držka), nerozdělujte poslední slovo na stránce, nedělte slova v nadpisech, první část rozděleného slova má mít alespoň dvě písmena (a-nabáze – špatně, ana-báze, anabá-ze – dobře).

4.1.2.2.6. Mezery

Chybějící nebo přebývající mezery patří mezi nejvíce viditelné prohřešky. Aby nedocházelo k zalomení řádku na nevhodném místě, používejte nezalomitelnou mezeru (K použití nezalomitelné mezery použijte u MS Word Ctrl+Shift+mezera.):

- za neslabičnými předložkami
- u číselných údajů opticky rozdělených po tisících (1 250 000)
- u spojení zkratek a číselných údajů (360 km/h, 20 Kč)
- u data (2.10.1998)
- u titulů a zkratek (MUDr. Horák, Portál, s.r.o.)
- u výrazů jako Karel IV.

POZOR!

- Mezi slovy je jen jedna mezera!
- Za čárkou a tečkou ve větě je vždy jedna mezera!
- Před tečkou a čárkou není mezera! (Není ani před středníkem, otazníkem, vykřičníkem a další větnou interpunkcí)
- Ke kontrole textu použijte znak ¶, umístěný na horní liště.

4.1.2.2.7. Interpunkční a další znaménka

- Tečka, čárka, dvojtečka, středník, otazník, vykřičník tři tečky – se píše se těsně za slovo (Bez mezery!!!) a za nimi následuje mezera.
- Uvozovky – se vždy píší ke slovu a vně se píše mezera. Dává-li se do uvozovek celá věta, sází se tečka uvnitř uvozovek. V celé práci použijte jeden typ uvozovek.

- Závorky – se přisazují těsně ke slovu (Bez mezery, mezera je nepřípustná ve všech případech!) a vně se píše mezera. Není přípustné používat místo závorek lomítek /text/.
- Pomlčka a spojovník – se nerozlišují a je to typografická chyba!
- Spojovník „-„ je kratší, používá se s dělení slov a jako spojovací znaménko. Kolem spojovníku se nesázejí mezery (ping-pong).
- Pomlčka „ –“ je delší a vkládají se kolem ní mezery. Naznačuje přestávku v řeči, má význam čárky. Nemá svoji klávesu (MS Word Ctrl+Num). V novějších verzích editoru MS Word se znak „-“ mezi mezerami automaticky změní v „ – “ .Pokud pomlčka nahrazuje předložku či spojku (výrazy „a“, „až“, „od do“,), sází se z obou stran bez mezery (např. v letech 1837-1838)
- Zkratky – dílčí prvky složených zkratek se ukončují tečkou a vzájemně se oddělují mezerou (např. S. K. Neumann, Veselí n. Mor., prof. PhDr. Jan Skočdopole, CSc.)

4.1.2.3. Bibliografické citace

Bibliografické citace informačních zdrojů v tištěné formě se řídí ČSN ISO 690-1 z roku 1996. Bibliografické citace elektronických informačních zdrojů určuje ČSN ISO 690-2, platná od ledna 2000.

4.1.2.3.1. Struktura bibliografických citací

Bibliografické citace jsou souhrn údajů o citované publikaci nebo její části a umožňující její identifikaci. Základ citace tvoří prvky, které je nezbytné do citace začlenit – pokud jsou identifikovatelné.

Pro tvorbu citace lze užít i OCR software pro digitalizaci potřebného odstavce, ale pozor, na dodržení normy, text je třeba náležitě upravit a ohraničit.

Povinné údaje citace monografické publikace jako celku:

Primární odpovědnost za dílo – příjmení a jméno toho, kdo dílo vytvořil nebo poskytl k zveřejnění (autor, editor, redaktor). Tiskneme příjmení velkými písmeny, jméno malými písmeny (BAUMAN, Zygmunt) Název díla – uvádíme celý. Je možné zkrátit, pokud nezměníme smysl názvu. Můžeme uvést také podnázev, pokud je to užitečné z hlediska zpřesnění nebo identifikace zdroje. Tiskneme kurzívou (např.: *Úvahy o postmoderní době*), nebo dle ISO 690-2 můžeme standartním písmem.

Vydání – zapisujeme arabskými číslicemi ve zkrácené podobě (2.vyd.,dotisk, 5.autoriz. přepracované vyd.)

Nakladatelské údaje – zapisujeme v pořadí: místo, nakladatelství, datum.

- místo vydání (Praha, Plzeň)
- nakladatelství – zapisujeme nakladatelství nebo jméno nakladatele. Není-li nakladatel udán, uvede se zkratka „b.n.“ (Sociologické nakladatelství).
- datum – zapisujeme vždy arabskou číslicí (2006)

Standardní číslo – je přiděleno popisné jednotce v souladu s platnými normami. ISBN pro monografické publikace, ISSN pro seriálové publikace. U starších titulů nebývá uvedeno.

Zápis na řádku

BAUMANN, Zygmunt. *Úvahy o postmoderní době*. 2.vyd.,dotisk. Praha: Sociologické nakladatelství (SLON), 2006. ISBN 80-86429-11-3.

PLAŇAVA, Ivo. *Průvodce mezilidskou komunikací*. 1.vyd., Praha: Grada Publishing, a. s., 2005. ISBN 80-247-0858-2.

Povinné údaje bibliografické citace seriálové publikace:

1. Název publikace – tiskneme kurzívou (Dějiny a současnost)
2. Podnázev – uvádíme pokud chceme zpřesnit zdroj (kulturně historická revue)
3. Odpovědnost – ten, kdo dílo vytvořil. Může to být osoba, korporace aj.
4. Vydání – uvádíme rok, odkdy se publikace vydává (1983-).
5. Místo vydání, nakladatelství – (např. Praha : NLN, s.r.o.)
6. Rok vydání – uvádíme rok, který jsme použili (např.2007)
7. Standardní číslo – např. ISSN 0700-0758

Zápis citace na řádku:

Dějiny a současnost: kulturně historická revue.o.s. pro podporu historické literatury a časopisu Dějiny a současnost, 1983-. Praha : NLN 2007. ISSN 0700-0758.

Povinné údaje bibliografická citace článku v seriálové publikaci

1. Primární odpovědnost za článek – KOUCKÁ, Pavla
2. Název článku – Myšlenková nákaza
3. Název zdrojového dokumentu – Psychologie dnes (zápis kurzívou)
4. Místo vydání, nakladatel – Praha : Portál
5. Rok vydání – 2006
6. Ročník – 12
7. Číslo svazku – 12
8. Strana – s.11 – 15

Zápis citace na řádku

KOUCKÁ, Pavla. Myšlenková nákaza. Psychologie dnes. Praha : Portál, 2006. Roč. 12, č. 12, s.11 – 15.

Bibliografická citace elektronické monografie, databáze a počítačového programu:

KOPPLOVÁ, B. SEKERA, M. Mediální studia na přelomu tisíciletí a role médií v dějinách českých zemí [CD-ROM]. Praha : Fontis, 2003.

Chemická laboratoř. Chemie 1. [CD-ROM]. Praha : Langmaster International, 2003-2006.

Bibliografická citace elektronické seriálové publikace, časopisu, diskusní skupiny:

I Dnes : elektronický deník [on line]. Dostupné z URL : http://idnes.cz.

Jak správně běhat [on line]. Quido. I Dnes.cz. Dostupné z URL : http://zdraví.idnes.cz/deset-duvodu-proc-vstat-z-pohovky-a-zacit-behat.

FORMAN, Zdeněk. Kýchání a jiné [on line].Dostupné z URL : http://chovani.unas.cz.

4.1.2.4. Odkazy

Odkaz je stručná forma citace vsunutá do pokračujícího textu nebo připojená jako poznámka. Odkaz musí obsahovat údaje dostatečné pro zajištění jednoznačnosti. Lze odkazovat metodou číselných citací, průběžnými poznámkami nebo uváděním prvního prvku a data.

Metoda číselných citací

Text a odkazy

Teorie neviditelných kolégií byla prozkoumána v přírodních vědách (2). O jejich nepřítomnosti se zmiňuje Stieg (13,s.556). Může to být, jak tvrdí Burchard (8)….

Citace (v seznamu citací jsou publikace očíslovány)

2. CRANE,D. Invisiblle colleges. Chicago : Univ.of Chicago Press,1972.

8. BURCHARD, JE.How humanists use a library.Intrex:report of a planing konference on information transfer experiments, Sept.3, 1965.Cambridge, Mass.:M.I.T.Press, 1965, s. 219.

13. STIEG,MF.The information needs of historians.College and Research Libraries, Nov. 1981, vol.42,no.6,c.549-560.

Metoda uvádění prvního prvku a data

Text a odkazy

Teorie neviditelných kolégií byla prozkoumána v přírodních vědách (Crane,1972). O jejich nepřítomnosti se zmiňuje Stieg (1981,s.556). Může to být, jak tvrdí Burchard (1965, s.219)

Citace (v seznamu citací nejsou publikace číslovány)

CRANE,D. Invisiblle colleges. Chicago : Univ.of Chicago Press,1972.

BURCHARD,JE.How humanists use a library.Intrex:report of a planing konference on information transfer experiments, Sept.3, 1965.Cambridge, Mass.:M.I.T.Press, 1965, s. 219.

STIEG,MF.The information needs of historians.College and Research Libraries, Nov. 1981, vol.42,no.6,c.549-560.

4.1.2.5. Použitá literatura pro vytvoření tohoto návodu

FILKA, Jaroslav.Metodika tvorby diplomové práce. Praktická pomůcka pro studenty vysokých škol.1.vyd. Brno : Knihař ve spolupráci s CMTF UP v Olomouci, 2002.ISBN 80-86292- 05-3.

KAPOUNOVÁ, Jana. Formální úprava diplomové práce. 1. vyd., dotisk. Ostrava : Ostravská Univerzita,1998. ISBN 80-7042-141-X.

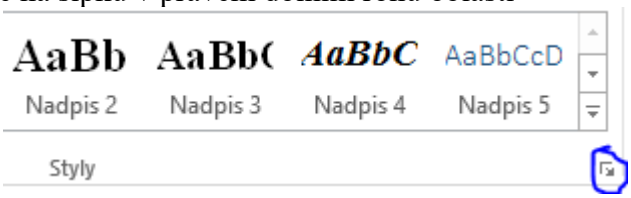
ČESKÝ NORMALIZAČNÍ INSTITUT.ČSN 01 6910.Úprava písemností psaných strojem nebo zpracovávaných textovými editory. 1. vyd. Praha : Český normalizační institut,1997.36s.

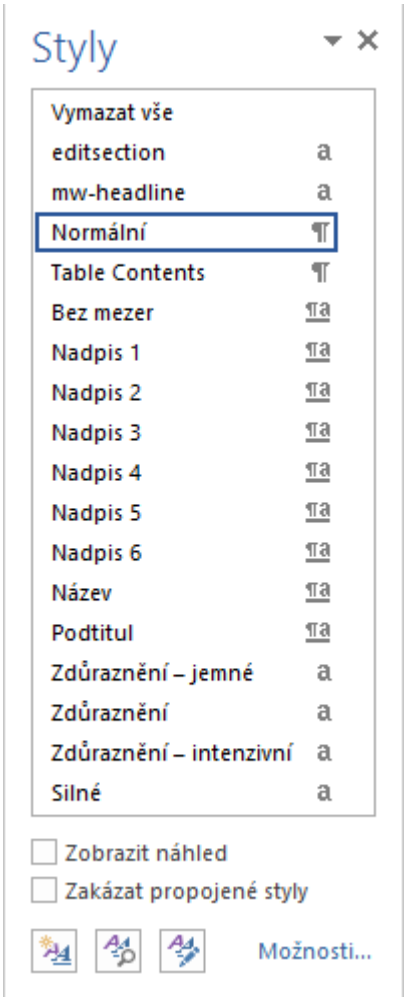
ČESKÝ NORMALIZAČNÍ INSTITUT. ČSN ISO 690 (010197) .Dokumentace – Bibliografické Citace : obsah, forma ,struktura. 2.vyd. Praha : Český normalizační institut,1996.

4.1.3. Nastavení stylů

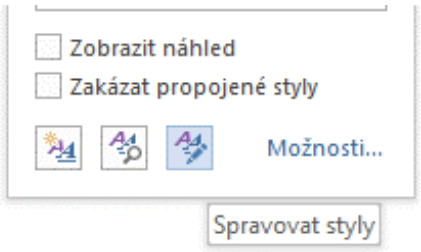
Každá z nás má nějaký styl oblékání, poslouchá nějaký styl muziky, to všechno patří k životnímu stylu každého z nás. Stejně jako my máme životní styl, tak dokumenty mají taky jistý styl svého vzhledu (tedy především ty formální). Proto, abychom si šetřili práci a byli schopni vytvořit styl, který je požadován, tak nám hodně napomůže nastavit si dílčí styly textu. Tak pojďme na to:

- ✓ karta domů
- ✓ oblast styly
- ✓ klineme na šipku v pravém dolním rohu oblasti



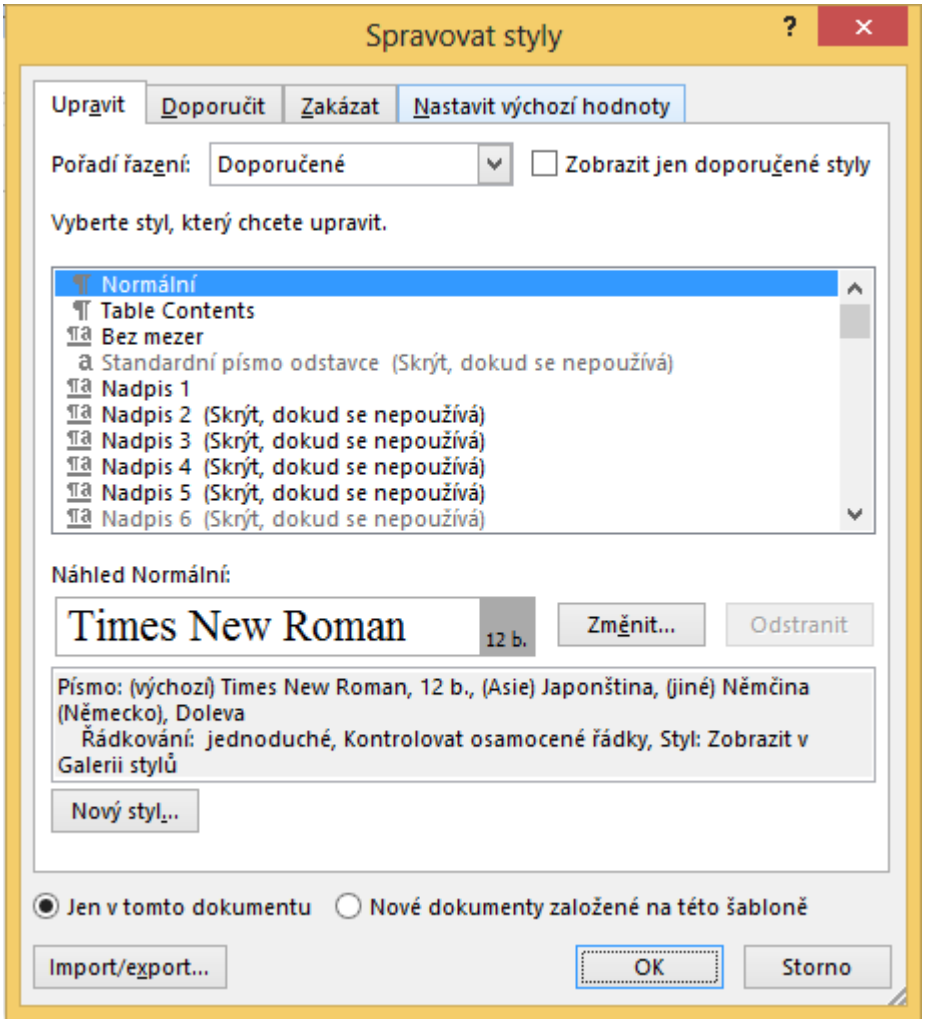


✓

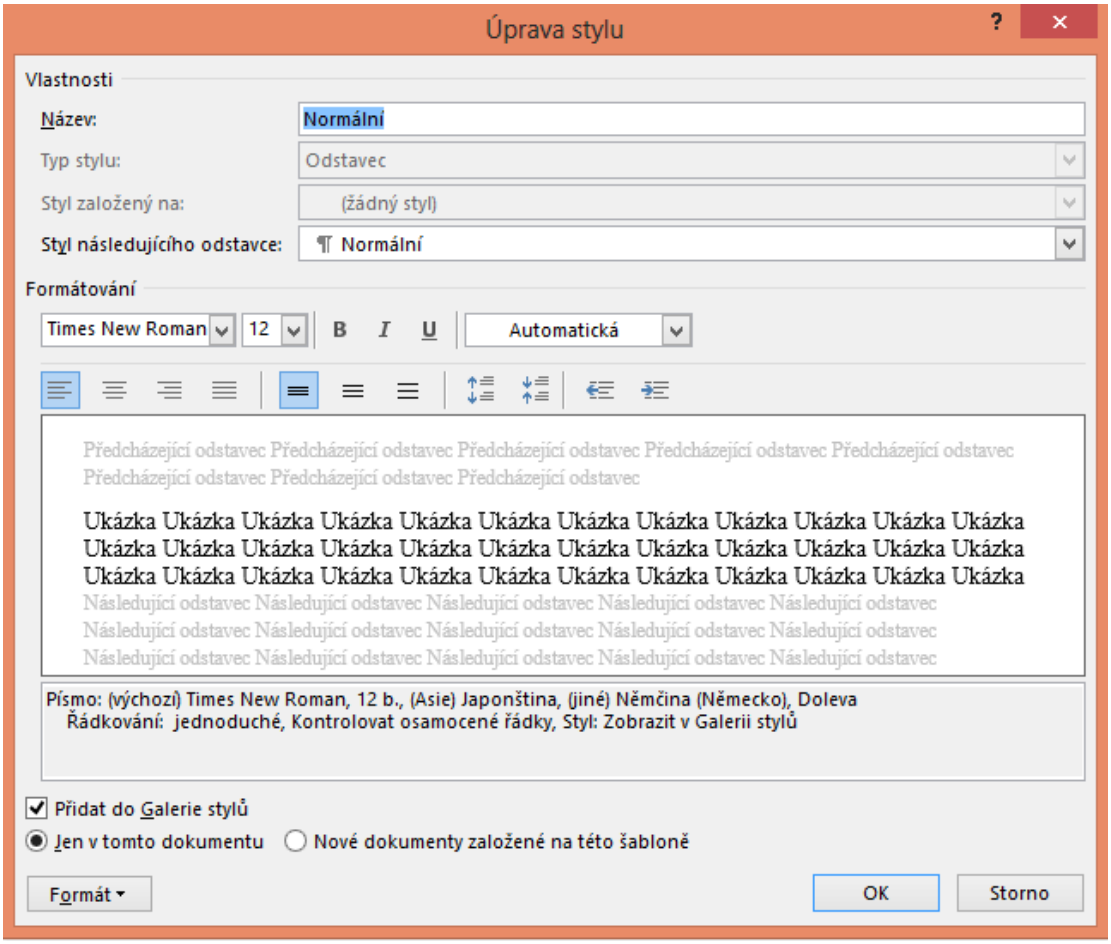


✓

✓ Otevřeme kontextové menu spravovat styly, tak jak je to naznačeno na předchozím obrázku



✓
✓ Vybereme styl a klikneme na **změnit**

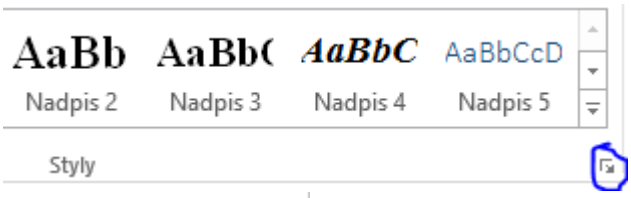


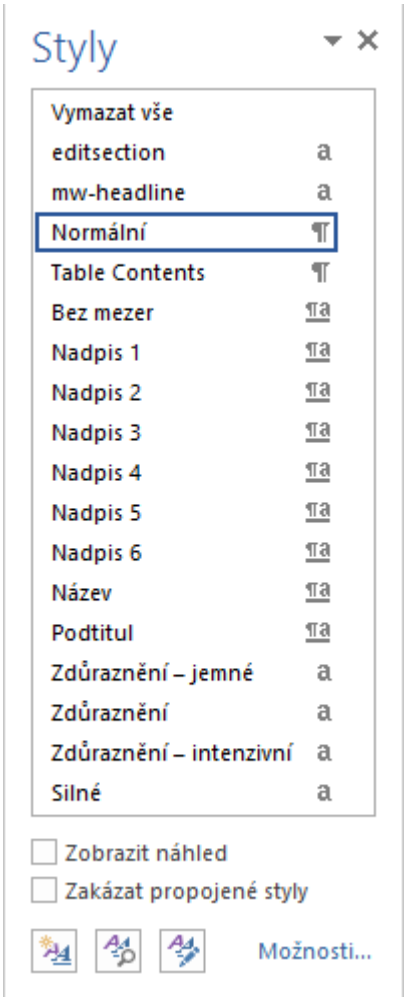
- ✓
- ✓ Vše vyplníme dle normy, mezery před a za najdeme v nabídce **formát** a její položce **odstavec**
- ✓ Normální je výchozí text, kterým se píše stať, nadpis jedna má nejvyšší prioritu a s rostoucím číslem se snižuje priorita
- ✓ Pokud chceme mít dle normy trvale nastavený Word, tak zaškrtneme položku nové dokumenty založené na této šabloně
- ✓ Pokud se jedná o styl normální, nadpis 1 až 6, citát, název, podtitul a název knihy, tak zkontrolujeme, zda je zaškrtnuto přidat do galerie stylů
- ✓ Vše potvrdíme **ok**

Pokud potřebujeme takto nastálo nastavený styl upravit pro jeden dokument, tak nám může pomoci galerie stylů, kde stačí na daný styl kliknout pravým tlačítkem a zvolit položku změnit. Pokud nemá být změna platná i pro další později vytvořené dokumenty, tak zaškrtneme jen v tomto dokumentu.

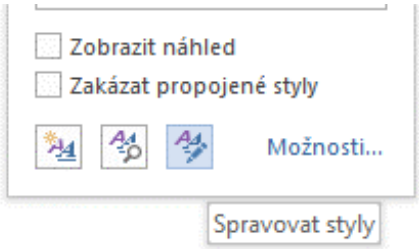
Každý z nás je naštvaný, když se něco pokazí a musí reinstalovat počítač, samozřejmě že musí tím pádem reinstalovat i sadu MS Office. Člověk si pak jen říká, „kurde“ to zase mám hledat tu normu a všechno to znova nastavovat? A mě se nechce… Potěším všechny, kteří se dostali do této situace, stačí mít někde zálohovaný dokument, který má styly nastavené dle normy, pak se dají styly importovat z tohoto dokumentu do šablony. Sám jsem to dělal už ani nevím kolikrát, tak pojďme na to.

- ➢ karta domů
- ➢ oblast styly
- ➢ klineme na šipku v pravém dolním rohu oblasti

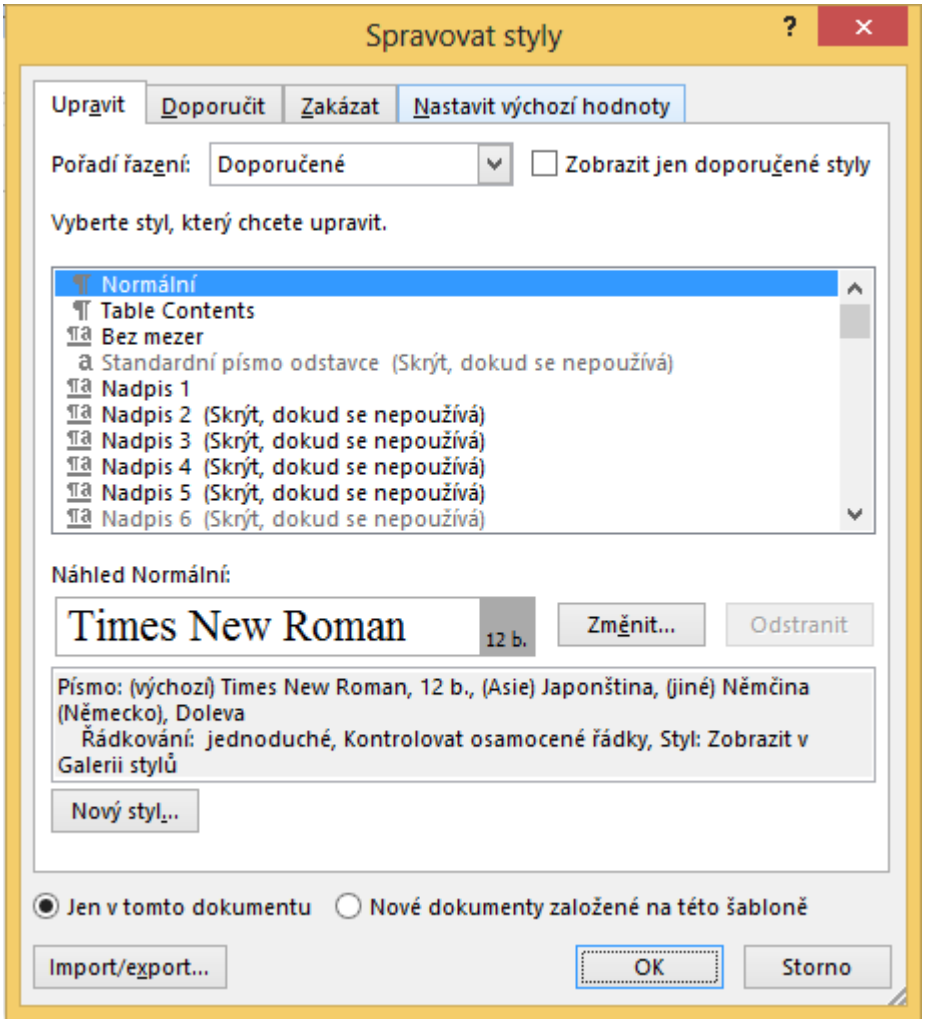




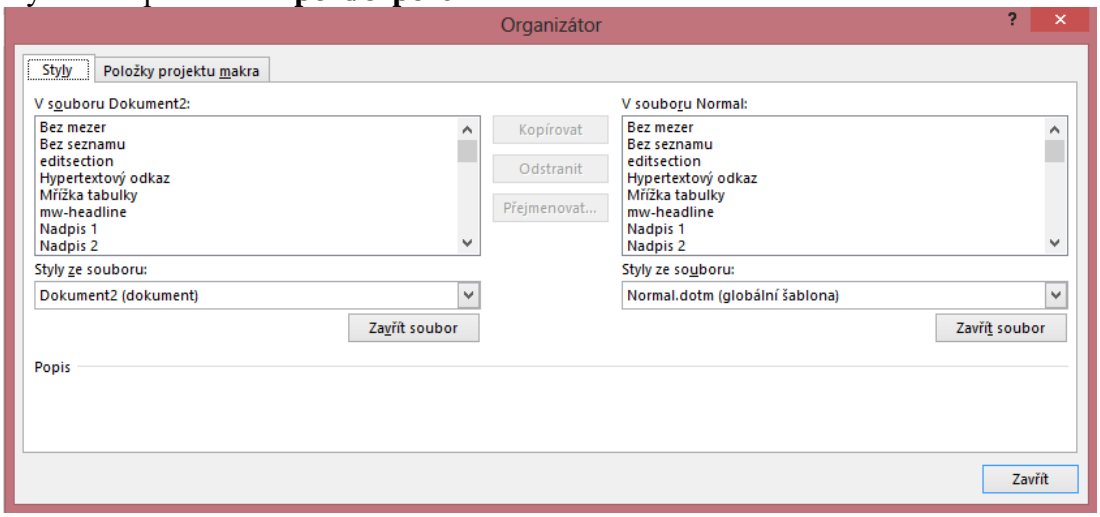
-



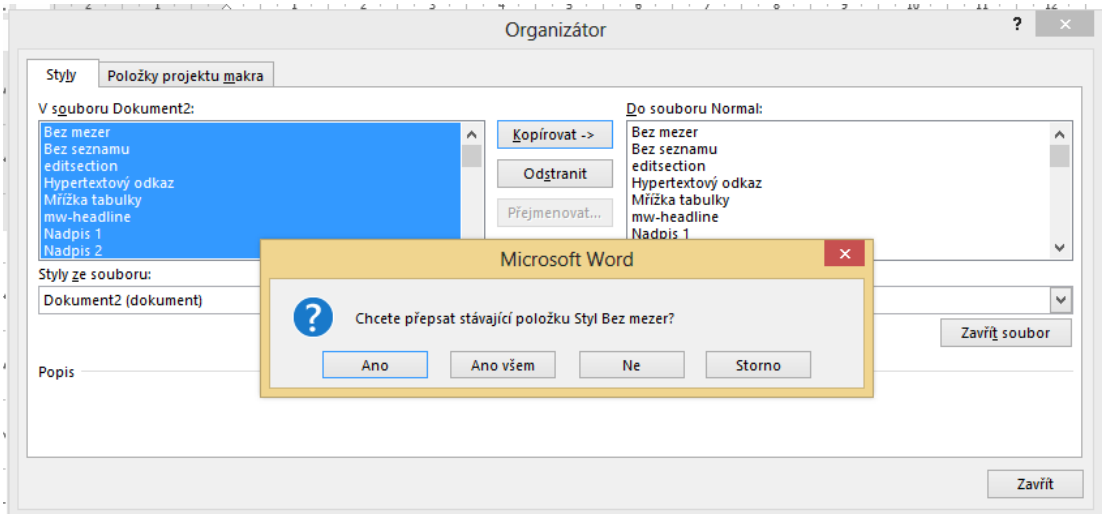
-
- Otevřeme kontextové menu spravovat styly, tak jak je to naznačeno na předchozím obrázku



- Vybereme položku **Import/export**…



- V levé části vybereme pomocí rozkliknutím nebo jako následek dalších možností po volbě zavřít soubor náš soubor se styly
- Klikneme na první styl a stáhneme posuvník až úplně dolů, stiskeneme klávesu shift a držíme ji stištěnou, následně klineme na poslední styl
- Pokud máme celý sloupec označený, tak zvolíme **kopírovat**



-
- Zvolíme **ano všem**
- Po dokončení stačí pouze u**zavřít** a máme vše nastaveno

4.1.3.1. Tvorba obsahu

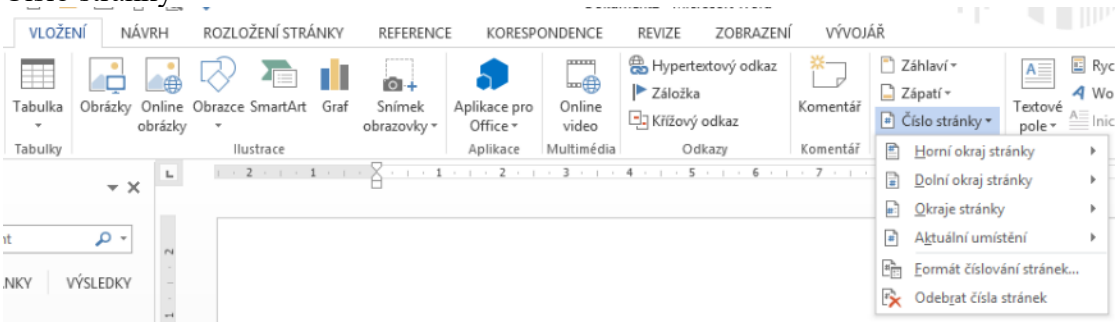
Pokud máme vytvořený text a při jeho editaci jsme využili styly (bez nich to nefunguje), tak se můžeme pustit do tvorby obsahu, aby všichni věděli, kde co najdou. Představte si, že bych do této publikace nedal obsah (uznávám, i ten při svém rozsahu může lést na nervy, nežli si na něj zvyknete, je podrobnější, nežli bývá dobrým zvykem), to by jste asi tuto publikaci nečetli a hodně špatně by se vám v ní hledalo, co nečetli, nestáhli by jste si ji, nebo nezakoupili v tištěné podobě.

K tomu, aby se nám podařilo vytvořit obsah však musíme nejprve udělat něco jiného, a to očíslovat strany, protože bez číslovaných strany by obsah byl jen plýtváním papírem.

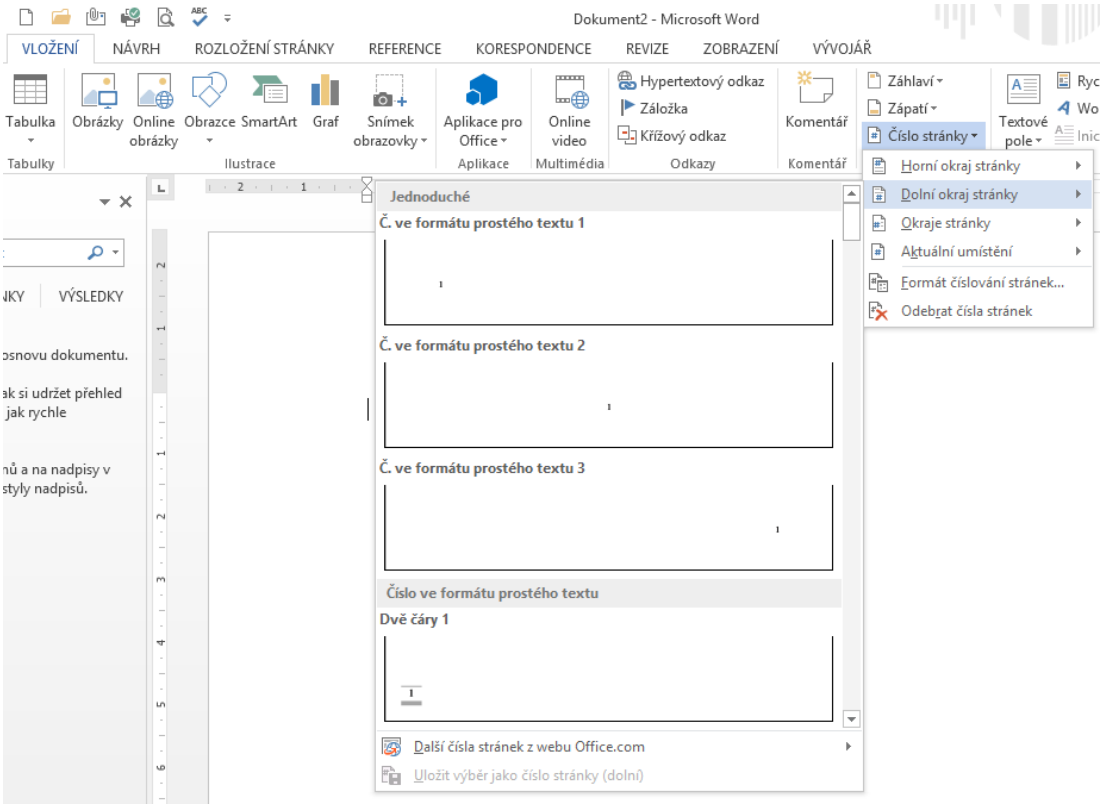
4.1.3.1.1. Číslování stránek

Není v tom žádná věda, jak se může zdát, Word všechno udělá za nás. Tak pojďme na to:

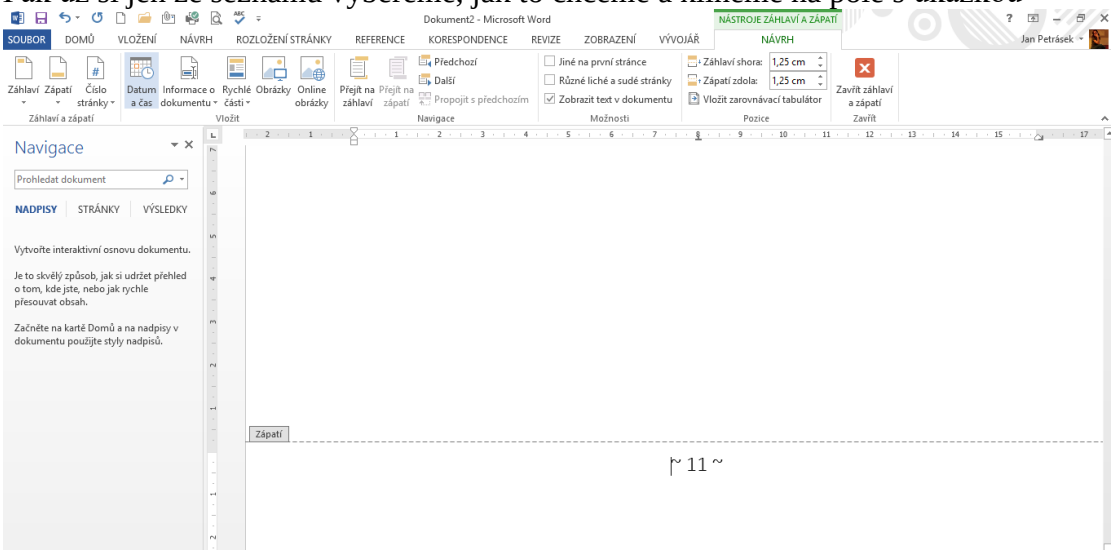
- ✓ Karta vložení
- ✓ Číslo stránky



- ✓
- ✓ Vybereme kam má být číslo strany uloženo (hodní/dolní okraj atd.)



✓ Pak už si jen ze seznamu vybereme, jak to chceme a klineme na pole s ukázkou



✓ Následně zaštěkneme na kartě návrh možnost jiné na první straně (pokud je první strana dokumentu titulní stranou)

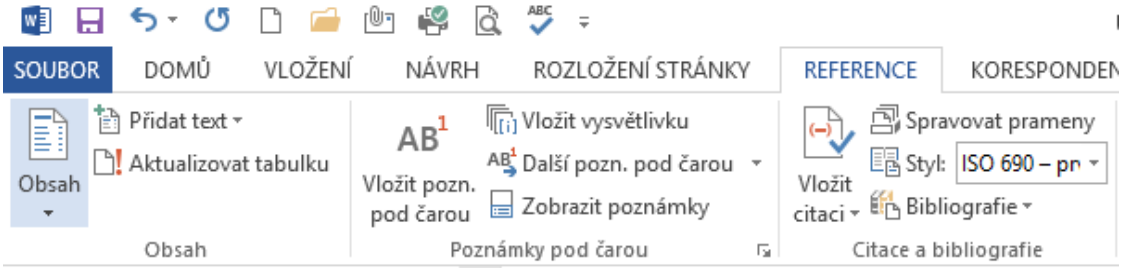
4.1.3.1.1. Úkol pro vás čtenáře

Doteď jsem psal výhradně já, nyní nadešel čas na avizovaný úkol. Vytvořte si prázdný dokument a pětkrát stisknete ctrl+enter, následně dvakrát enter a pak ještě šestkrát ctrl+enter. Když to budete mít, tak je vaším úkolem, aby Word očísloval jen poslední šestici stran a to počínaje číslem 2.

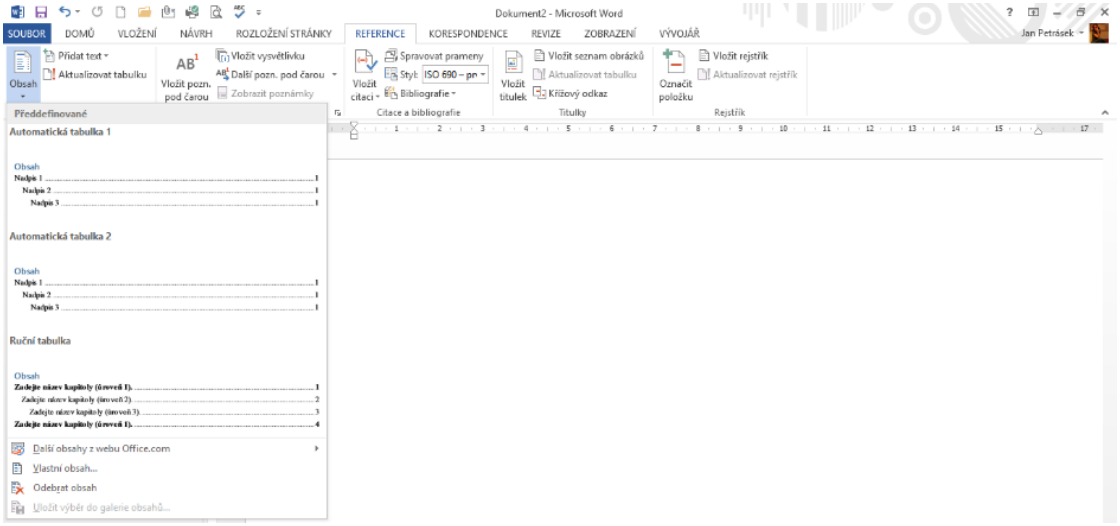
4.1.3.1.2. Vlastní obsah

Nyní máme vše, co potřebujeme pro tvorbu obsahu. Máme nastylovaný dokument, který má očíslované strany, takže v tvorbě obsahu nám již nic nebrání.

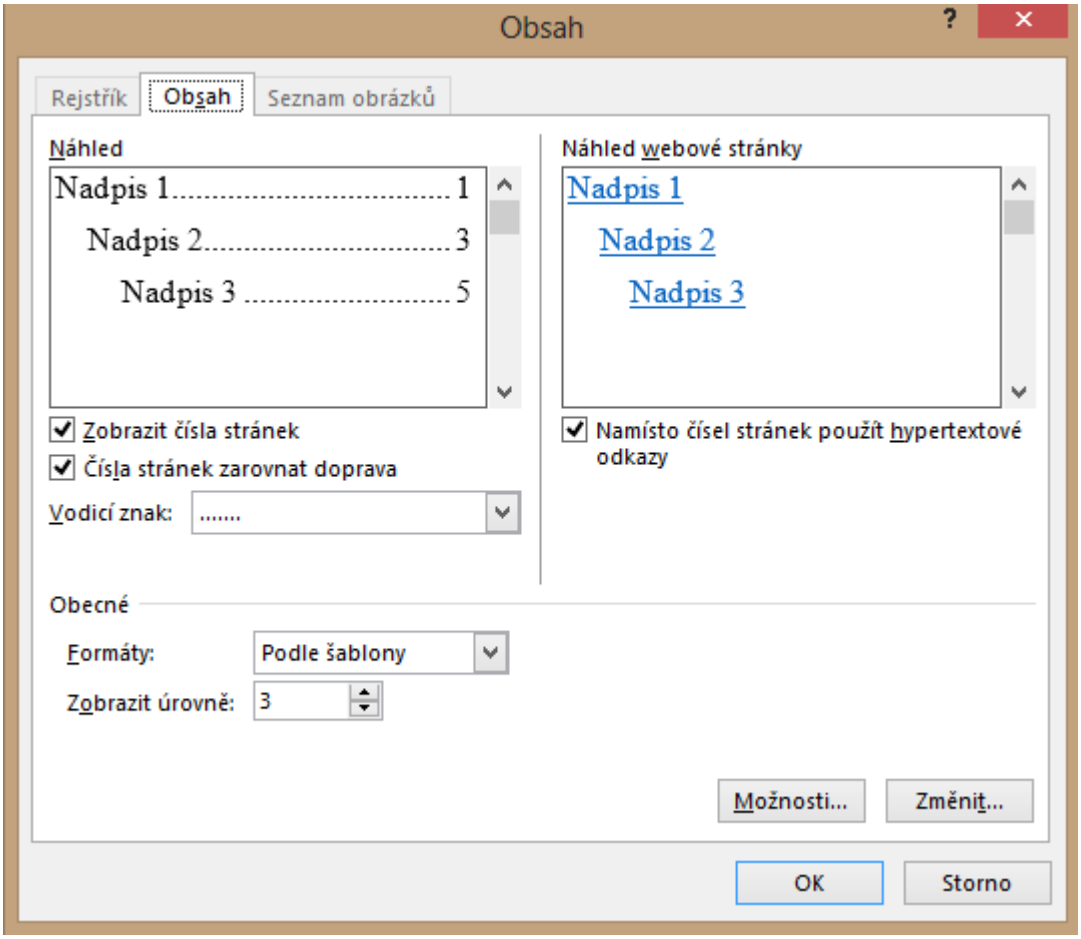
- Přejdeme na kartu reference



- Rozklikneme nabídku obsah



- Vybereme některý z navržených obsahů, pokud se nám nelíbí žádný, tak zvolíme vlastní obsah



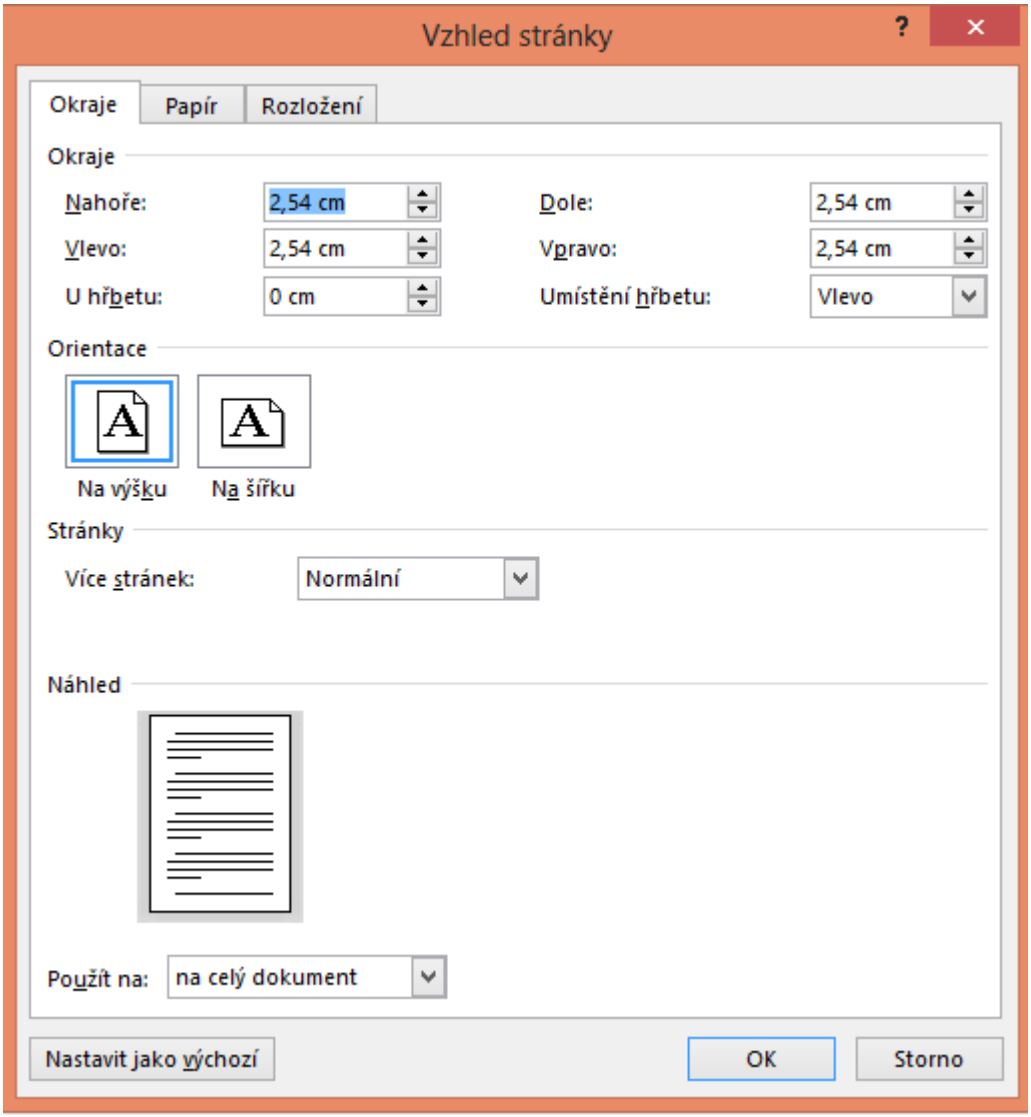
- Zde si zvolíme formát obsahu a zobrazované úrovně nadpisů
- Pokud si přejeme změnit, tak provedeme změnu vodicích znaků a stvrdíme ok
- ✓ Obsah je na světě
- ✓ Při provádění dalších změn v dokumentu si vždy zvolíme možnost aktualizovat pole, buď ve volbě pravého tlačítka myši po kliknutí do obsahu, nebo na kartě reference zvolíme aktualizovat

4.1.4. Nastavení okrajů

Jistě jste si všimli, že v normě jsou dané určité okraje. Tato kniha má ale jiné, na všech stranách 2,5 cm, je to z důvodu, že je počítáno s jejím čtením v elektronické podobě, tak aby byly dobře čitelné snímky obrazovek, byl zvolen menší okraj. Určitě se vám všem, stejně jako mě, než jsem si pořídil tiskárnu s bezokrajovým tiskem, už někdy stalo, že když jste si chtěli něco vytisknout, pokud jste v dokumentu měli příliš velký obrázek, tak vás tiskárna poctila hlášením, že máte položky i mimo tisknutelnou plochu, zda si přejete pokračovat. Toto hlášení vzniklo, protože jste měli obrázek, který přesáhl okraje nastavené ve Wordu, potažmo minimální okraje, které má ve svém nastavení tiskárna.

Pojďme si nyní ukázat, jak se okraje ve Wordu dají upravovat. Není to žádná věda, postupujme společně dle následujících instrukcí:

- Přejdeme na katru rozložení stránky
- Úplně vlevo si vybereme položku okraje
- Abychom mohli zadat okraj, který splňuje normu, tak musíme zvolit poslední položku, tedy vlastní okraje

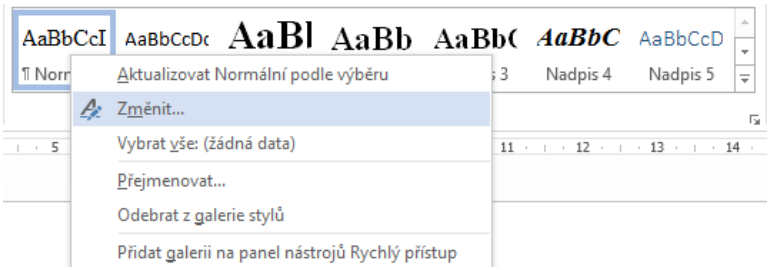


- V tomto dialogovém okně si můžeme vybrat kromě úpravy okrajů i orientaci stránky a formát papíru (v základu máme formát A4)
- Nastavíme požadované okraje a necháme umístění hřbetu vlevo
- Položka u hřbetu zůstane nevyplněná, tu užijeme pouze pokud bychom dělali atypickou vazbu s užitím hřbetu jinde, nám správnou hodnotu udává levý okraj
- Vše dokončíme potvrzením OK

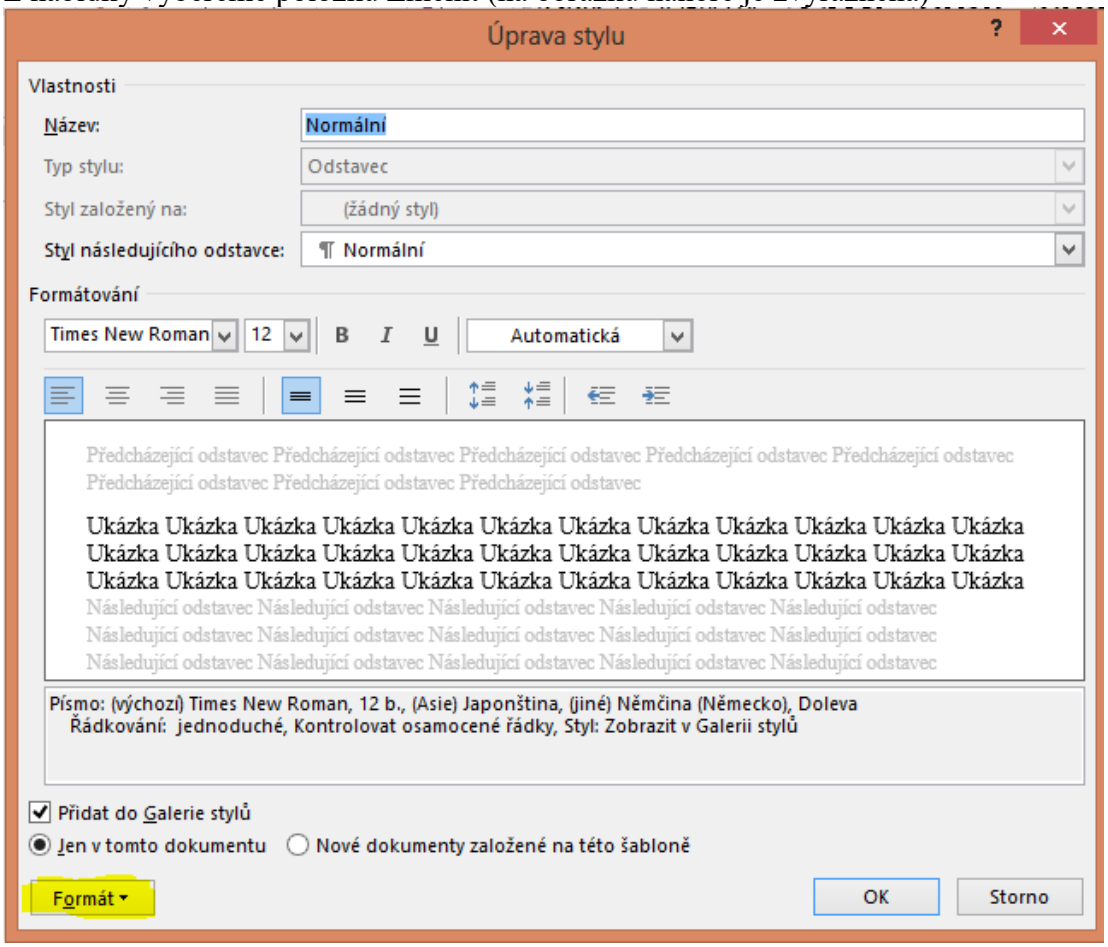
4.1.5. Jak si šetřit práci s tvorbou odstavců?

Každý to zná, nedodržuji to zde úplně striktně, jak jste si asi již mohli všimnout, někde mi to nejde do kupy s odrážkami či číselnými seznamy, jinde jsem jen líný. Každý odstavec začíná odsazením prvního řádku, to nás všechny učily již naše češtinářky na základní škole, ale jak odstranit tu lenost, aby člověk na to nemusel myslet a odsazovat klávesou tabulátoru? No, ono je to jednodušší, nežli by se mohlo zdát, pokud máme „souvislý“ text a uděláme to pomocí klávesy jednou, tak stačí zakončit odstavec klávesou enter a Word doplní toto odsazení i u dalšího odstavce. Pokud to chceme mít opravdu všude poctivě a nechceme na to myslet, tak se musíme pustit do úpravy stylu. Upravovat budeme styl Normální, tak pojďme na to.

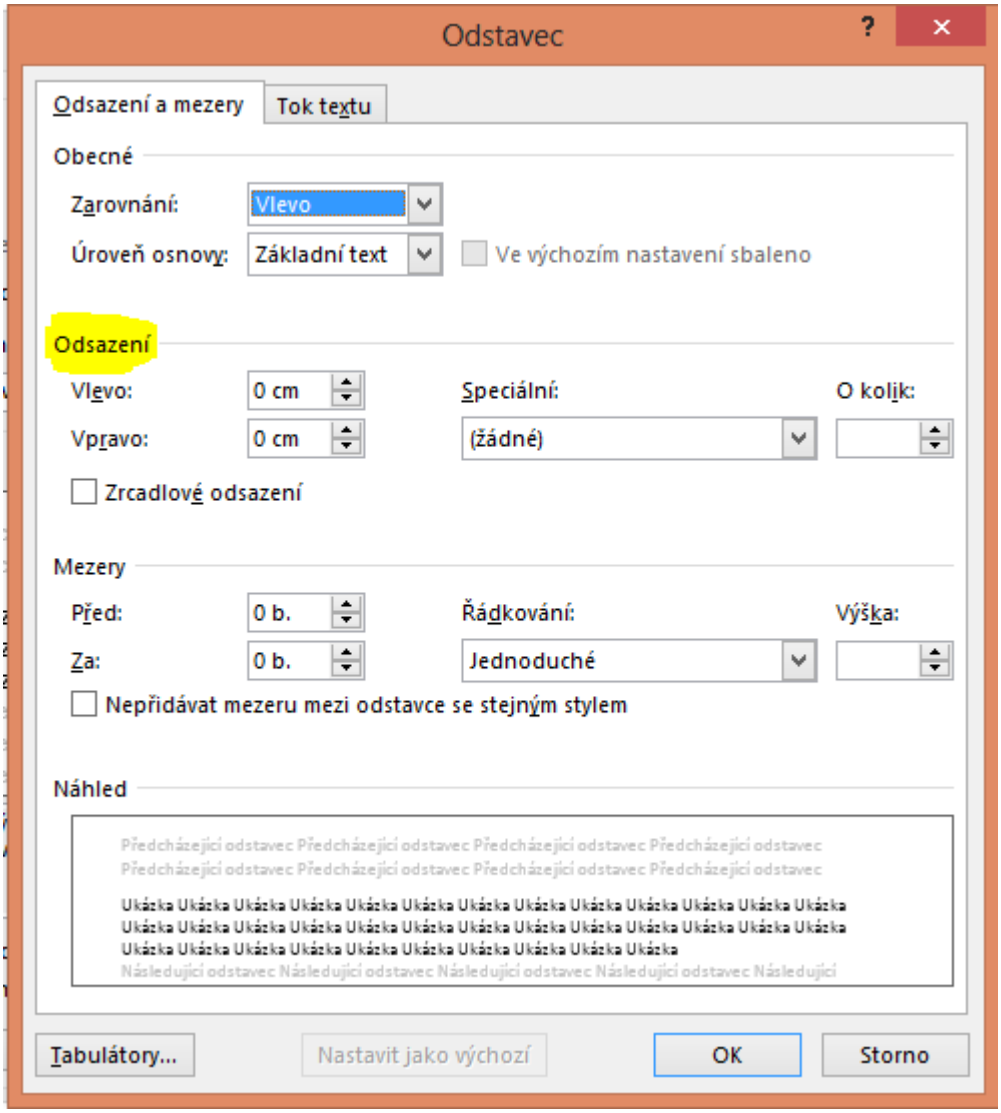
- Na kartě domů si v oblasti styly klikneme pravým tlačítkem myši (pokud užíváme dotyk, tak dlouze podržíme) na styl normální



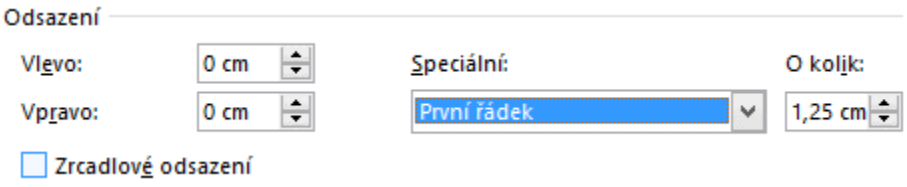
- Z nabídky vybereme položku Změnit (na obrázku nahoře je zvýrazněna)



- Otevřeme nabídku formát a zvolíme položku odstavec (dostaneme následující nabídku, kde budeme pracovat se zvýrazněnou částí)



- Rozklikneme nabídku položky speciální a zvolíme první řádek



- Pokud máme tak, jak je na výše uvedeném obrázku, tak vše postupně potvrdíme **OK**

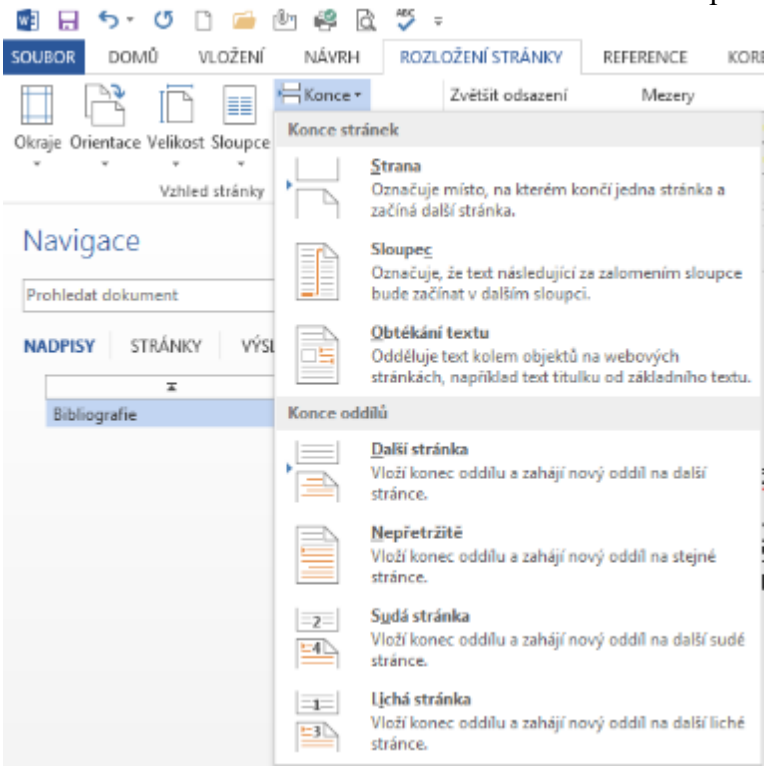
4.1.5.1. Úkol 2 pro Vás

Nyní se zase stanu exekutorem vaší pozornosti při čtení, máme tu další úkol. Pomocí stylů a všeho, co jsme si ukázali, se pokuste zkopírovat si delší článek z wikipedie nebo od jinud z webu a upravit jej podle normy, tedy nadefinovat nadpisy, očíslovat stránky, nastavit okraje, vložit obsah apod. Na závěr si cvičně udělejte citaci zdroje. Tento úkol je plně ve vaší kompetenci, tentokrát nebudu (pro rozmanitost správných řešení) správně řešení uvádět, již jsem jej uvedl v předcházejícím textu. Přeji hodně zdaru. ☺

4.1.6. Konce

Význam slova konec je velmi rozmanitý, konec může mít lano, přednáška, vztah, ale třeba i stránka, odstavec, sloupec, či průběh. Obyčejně by se dalo říct, že konec dává možnost vzniknout něčemu novému, nějakému začátku. Držme se nyní spíše těch konců, které souvisejí s editací textu. Konce nám umožňují rozdělit číslování stran, změnit záhlaví či zápatí, aby nebylo stejné v celém dokumentu, správně (nikoliv „prasácky“ enterem) zakončit odstavec či jiný blok textu, zakonči stranu, aby nedocházelo k rozhození dokumentu při jeho další editaci či převodu do jiného formátu. Umožňují nám vkládat prázdné řádky a vůbec.

Konce, jejich možnosti a typy najdeme na kartě rozložení stránky v sekci vzhled stránky. Pokud si tuto nabídku rozklineme tak se nám dostane přibližně tohoto pohledu:



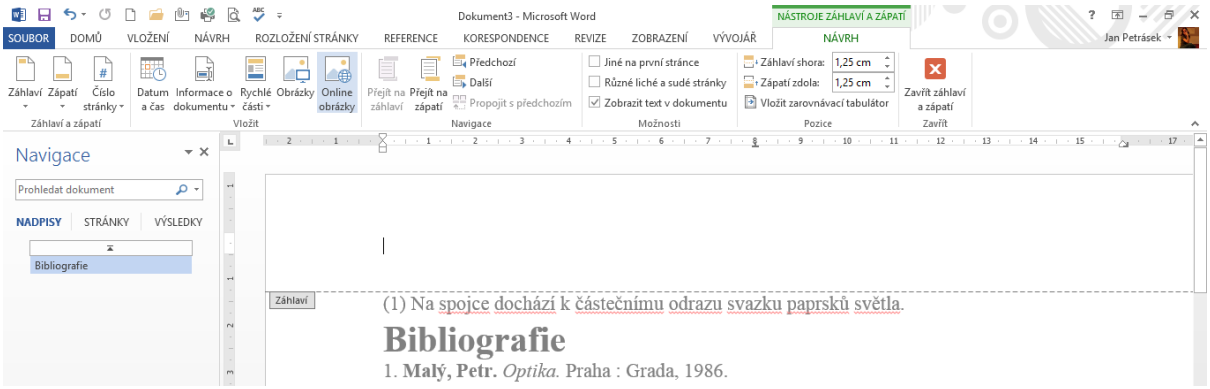
Nepřetržité konce slouží k rozdělení oddílů, myslím si, že je zbytečné se více o tomto problému rozepisovat, protože vše podstatné nám Word prozradí v nápovědě.

4.1.7. Záhlaví a zápatí

Určitě jsme se setkali s tím, že máme do záhlaví uvést svoje jméno třídu a datum, nebo cokoliv jiného. Jenže jak to udělat? Opět je to na rozdíl od starých verzí velmi jednoduché, přejdeme na kartu vložení a zvolíme záhlaví či zápatí dle potřeby. Po rozkliknuti nabídky si může vybrat z šablon pro ty nejužívanější, pokud chceme vytvořit vlastní, tak zvolíme položku upravit.



Následně se nám stane toto:



A můžeme volně vytvářet záhlaví či zápatí, dle naší potřeby. Je možné kdykoli přecházet na libovolnou kartu dle naší potřeby, jen pokud budeme chtít užít některý z editačních prvků, je nutné přejít opět na kartu návrh.

Záhlaví je typicky samostatné pro každý oddíl, takže pokud chceme mít pro více oddílů stejné záhlaví, tak musíme zapnout tuto položku. Úpravy záhlaví či zápatí ukončujeme buď kliknutím na červený křížek v kartě návrh, nebo klikem (popřípadě dvojklikem) do textu pod záhlavím.

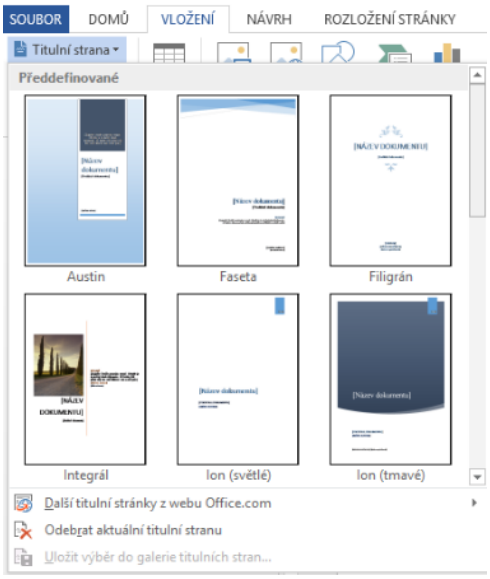
4.1.7.1. Úkol 3 pro vás čtenáře

Tentokrát vás nenechám dlouho zahálet, je mi jasné, že minulou práci jste si uložili, takže tento dokument si opět otevřete a dejte do záhlaví vaše jméno, datum vzniku dokumentu, místo vzniku dokumentu a jméno webového prohlížeče, který jste užili při hledání článku. Opět nebudu uvádět řešení, protože máte zcela volnou ruku s konkrétní podobou záhlaví. Jak postupovat jsme si ukázali výše a dobrým pomocníkem, který není nikdy zakázán je nápověda ☺.

4.1.8. Titulní strana

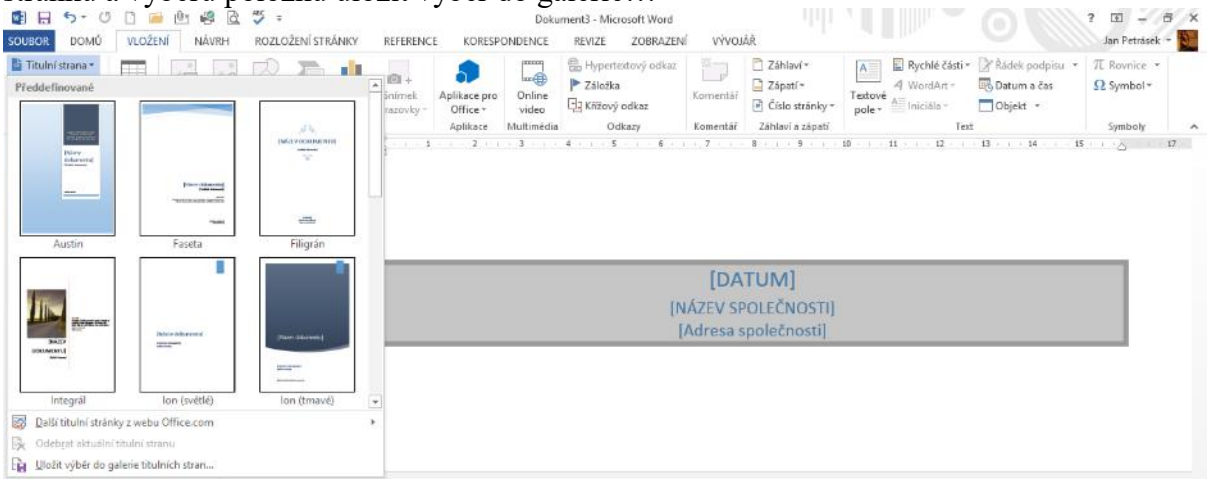
Známe to všichni, učitel nebo šéf si vymyslí, jak má vypadat titulní strana, nebo že máme vypracovat nějakou esej, či tak na jisté téma. Všechny tyto dokumenty potřebují titulní stránku, ta se nám nejlépe vytváří úpravou z již existující titulní stránky, nebo můžeme dokonce některou z nich užít (jako já na tuto knihu). Titulní stránku, kterou si vytvoříme při studiu třeba na seminární práce, můžeme užít opakovaně, pokud si ji uložíme, vše si ukážeme pěkně popořádku.

Nejprve si řekneme, kde vlastně takovou titulní stranu nalezneme, je to na kartě vložení v oblasti stránky. Pokud si otevřeme nabídku titulní strana, tak uvidíme následující:



Povšimněme si, že zde chybí položka vlastní. Vlastní stránku leze vytvořit úpravou některé z předefinovaných, nebo vytvořením na čistém blanku a označení celé stránky.

Pokud si označím stránku, kterou chci uložit jako titulní, tak poté klepnu opět na titulní stránku a vyberu položku uložit výběr do galerie…

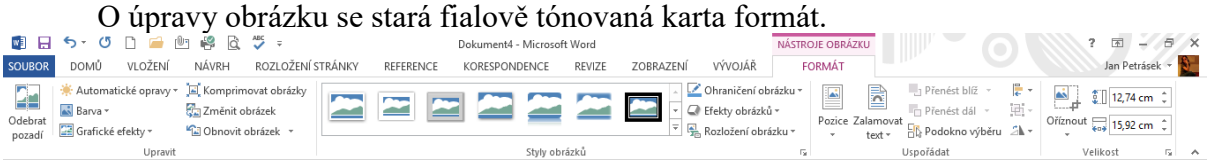


Pokud si vyberu některou ze stránek galerie, tak ji stačí již jen vyplnit ☺ a nadbytečné smazat, tak aby naše titulní strana vyhovila požadavkům. Šablona nám nastaví písmo, ale vše je možné měnit jako by to byl obyčejný text, jen v tomto případě stejně jako na drobné úpravy nepoužíváme styly.

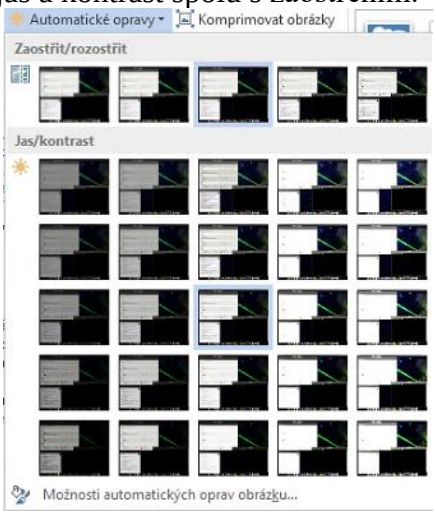
4.1.9. Obrázky

Nedílnou součástí většiny našich dokumentů jsou obrázky, které doplňují text. Může jít o fotografie, nákresy, snímky obrazovky a vše, co nás napadne. Nový Word nám umožňuje i zkopírované položky z jiných aplikací vkládat jako obrázky, pokud obsahují jiná data nežli jen tex. Vložit obrázek je jedna věc, druhá je jej upravit, natočit, oříznout, zesvětlit, zvýraznit, změnit jeho velikost, zasadit jej na požadované místo v textu apod.

O úpravy obrázku se stará fialově tónovaná karta formát.



V nabídce máme spoustu předdefinovaných úprav, ale ty se nehodí pro formální dokumenty. Pro formální dokumenty můžeme v případě potřeby užít automatické opravy, kde najdeme jas a kontrast spolu s zaostřením.



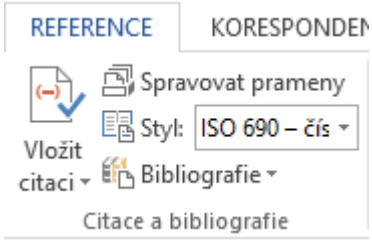
Další možnost, kterou pro úpravu obrázků v oficiálních dokumentech máme je oříznutí či změna velikosti. Zásadní vlastností je pozice, ta udává, kde bude na stránce obrázek umístěn a zda bude za textem, před textem či v textu, který jej bude obtékat.

Pokud zpracováváme více, nebo rozměrově větší obrázky, tak je dobré obrázky komprimovat, aby se zmenšil dokument jako celek, co se týká zabrané paměti a byl tedy snáze přenositelný po síti.

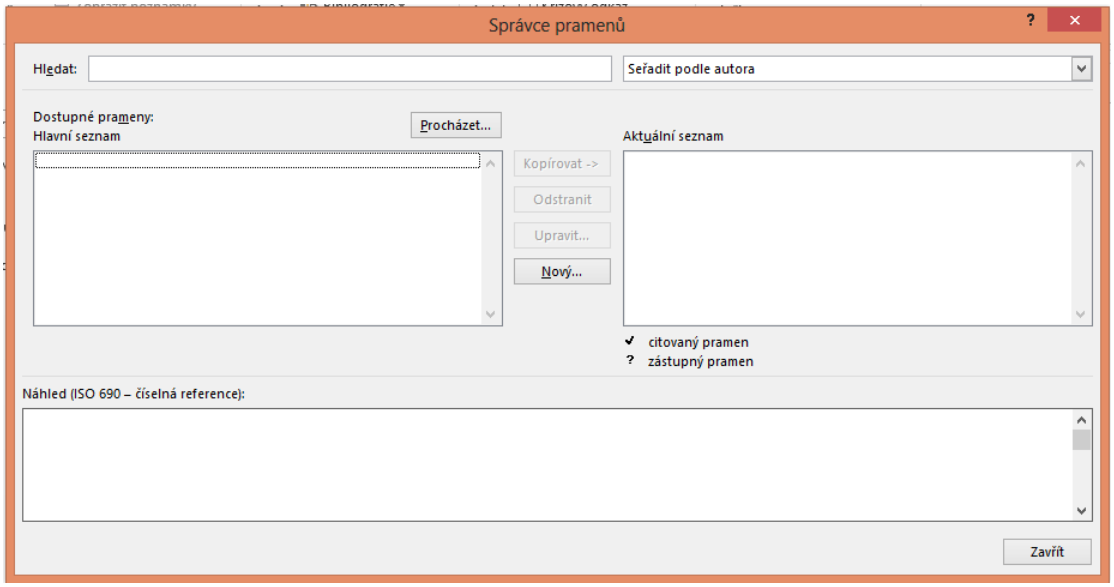
4.1.10. Citace

Pokud chceme citovat nějakou knihu či práci, je nutné napsat citaci a také uvést zdroje. Právě na uvedení zdrojů má Word dost silné nástroje. Pro vlastní text citace je předdefinován samostatný styl, jinak text musí být uzavřen v uvozovkách a kurzívou. Pojďme si nyní představit nástroj na bibliografii.

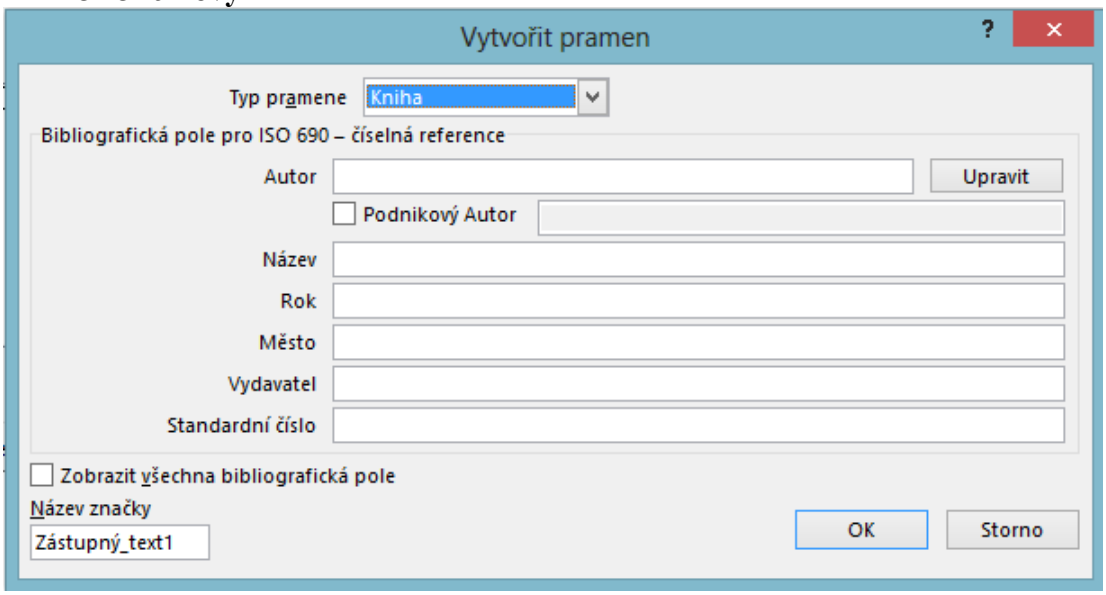
- Přejděme na kartu reference



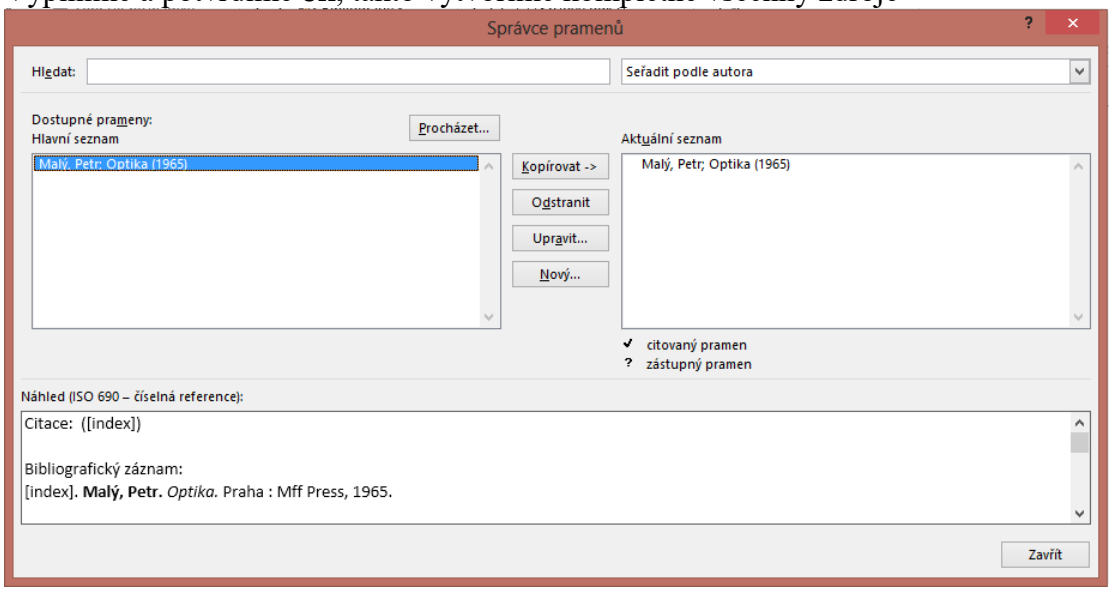
- Zvolíme **Spravovat Prameny**



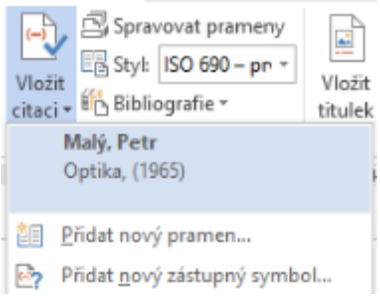
- Klikneme na **nový**



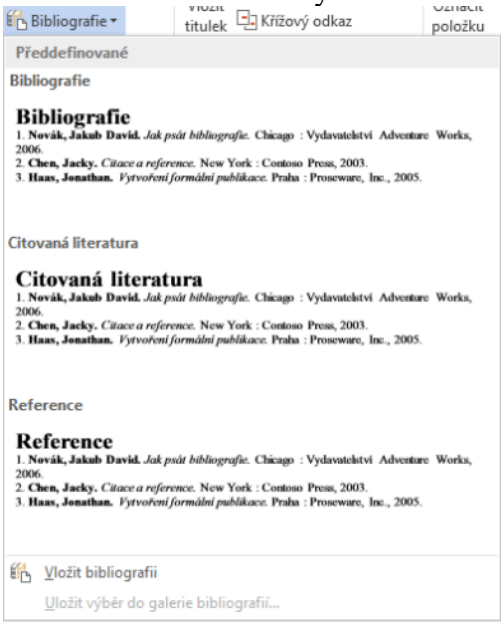
- Vyplníme a potvrdíme **ok**, takto vytvoříme kompletně všechny zdroje



- Dokončíme volbou **zavřít**
- Pro vložení citace si rozbalíme **vložit citaci** a klikneme na příslušný zdroj



- Následně zapíšeme citovanou pasáž
- (1) „*Na rozhraní prostředí dochází k lomu paprsku*.“
- Pro vložení seznamu zdrojů zvolíme položku **bibliografie** a Vybereme jeden ze seznamů citované literatury



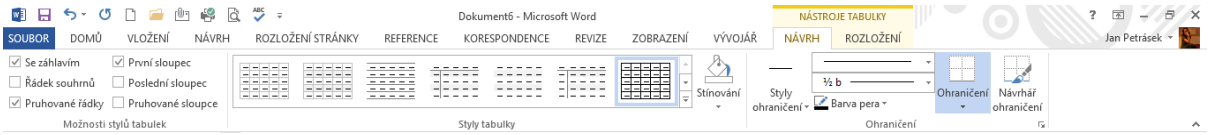
- Pokud chceme vložit zdroj citace na jeden řádek přímo k citaci, tak užijeme položku vložit bibliografii

4.1.11. Tabulka

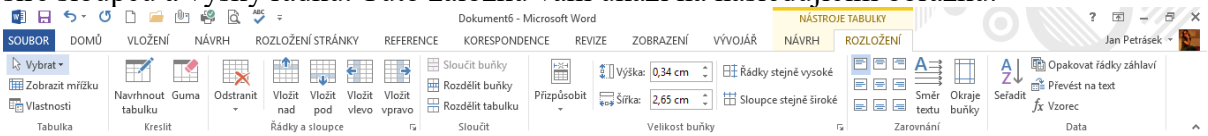
Každý se někdy potřebuje vyznat v jednoduchých heslovitých věcech a vidět jak spolu souvisí, nebo popřípadě i čísla. Tabulka je často užívaným šikovným pomocníkem, pokud budete touto knihou listovat, tak zde nějaké tabulky také naleznete. Tabulka nám za jistých okolností může pomoci i s formátováním dokumentu, pokud děláme třeba strukturovaný životopis (a zapomeneme, že Word a Publisher mají pro tyto účely šablonu), tak máme dvě možnosti, buď pracovat s textovými poli, nebo užít právě tabulku, do které napíšeme celý dokument a pak jen vypneme ohraničení, takže nebude vidět. Neměl bych vám takovéto věci říkat, protože jsme v učebnici a toto je chybný postup, který rozhodně nezkoušejte na učitele informatiky, ale v praxi, pokud to zasíláme jako pdf nebo tiskneme tak si takto práci šetřit můžeme, tam to nikdo kontrolovat nebude.

Tabulku vytvoříme na kartě vložení, kde je přímo prvek tabulka. Máme dvě základní možnosti, označením čtverečků (buněk) v nabídce prvku vytvoříme tabulku, nebo použijeme možnost vložit tabulku. Pokud chceme ve Wordu prezentovat data vytvořená v Excelu, tak můžeme tabulku z Excelu jen zkopírovat, nebo můžeme použít položku tabulka Excelu.

Když vložíme tabulku do dokumentu, tak se nám automaticky otevře karta pro práci s tabulkou.



Tato karta má dvě záložky, návrh, kde najdeme položky hraničení a stínování, rychlé formátování podle předdefinované šablony, a rozložení, kde je vše co se týká obsahu tabulky (zarovnání, seřazení a spol.) a také nástroje pro úpravu tabulky (přidání nebo odebrání sloupců / řádků). Na záložce rozložení je možné najít i nástroje pro úpravu rozměrů buňky a sjednocení šíře sloupců a výšky řádků. Tuto záložku vám ukáži na následujícím obrázku.



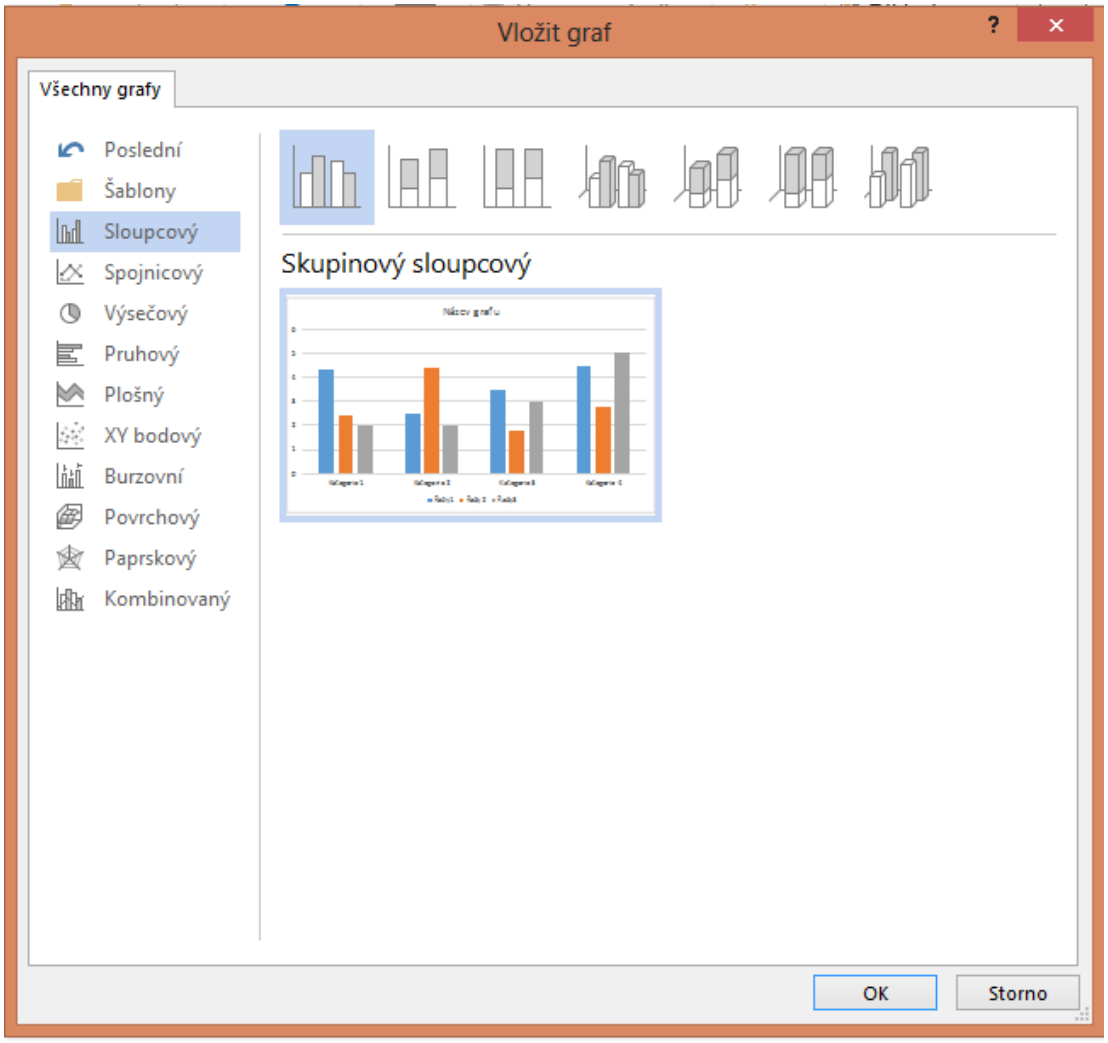
4.1.11.1. Úkol pro čtenáře

Jak jste zjistili, tak nemalou část Wordu vám ponechávám jako samostudium, řeknu vám, kde co je a k čemu se to dá užít. Přijde mi zbytečné se s tím rozepisovat, jako to dělal jeden můj kolega pro verzi 2007, kde jen nad Wordem strávil cca 3 měsíce napsal o něm knihu s více jak 300 stranami. Vzhledem k tomu, že nový Word vám po najetí na prvek napíše, co to je a co to umí, popřípadě dá náhled nějakého vzorového příkladu, tak si mohu dovolit vám toho nechat ne málo na samostudium. Samostudium hraje v informatice velmi důležitou roli, možná důležitější nežli v jiných oborech. Většinu špičkových programátorů nikdo neučil, jak to mají dělat, tak například Bill Gates, ten během týdne vymyslel vlastní programovací jazyk (Basic) a napsal program, který uměl kód Basicu převést do strojového kódu. Nepsal to sám, dělali na tom ve dvou, ale nikdo je to neučil, naučili se to sami, metodou pokus omyl. Neříkám, že jsem nějaká špička, ale mě také nikdo neučil, všechny praktické znalosti jsem získal sám metodou pokus omyl, neměl jsem se nikdy koho zeptat, a když potom již ano, tak jsem to nepotřeboval. Na škole mi dali teoretický základ a znalost historie, ale praxi již ne, tu jsem uměl, když jsem přišel na střední školu. Je dobré mít nějakou pěknou knihu, která je dobrým pomocníkem, ale musíte hlavně pracovat sami, kniha má poradit, prozradit základní princip, navést na správnou cestu a prozradit některé fígle, na které člověk přijde až po letech praxe, ale musí vám dát prostor k samostudiu, tak tady je ten prostor a další úkol.

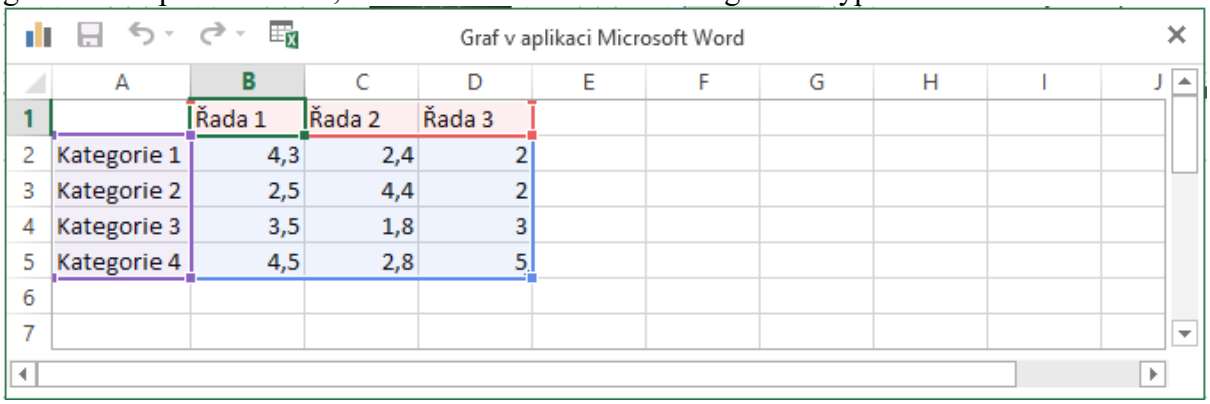
Vytvořte si tabulku, kde budete mít spotřebu vašeho auta, množství natankovaného paliva, najeté kilometry a spotřebu, popřípadě data tankování, udělejte si tento přehled, budeme s ním pracovat později dále po drobné modifikaci.

4.1.12. Grafy

Pokud by mi někdo řekl tak před 10 lety, že vytvořil graf ve Wordu, považoval bych jej za cvoka, protože na grafy a výpočty je v balíku Office předurčena aplikace Excel. Nicméně jak jsme si řekli již v úvodu, tak nová sada Office si zakládá na propojení programů, přenos šablon a podobně, takže není divu, že aplikace Word dokáže vytvořit grafy, nebo upravit ty, které jsou zkopírované z aplikace Excel, takže již odpadá pro většinu základních úprav nutnost jít znovu do Excelu, tam graf upravit, a nahradit původní instanci ve Wordu. Je jasné, že ač se jedná o velký skok kupředu, tak Wodr má stále mírně omezené možnosti práce s grafy. Pokud nevyužíváme vložení grafu z Excelu, tak můžeme graf vytvořit pomocí karty vložení, kde v sekci ilustrace nalezneme položku graf. Pokud na tuto položku klineme, tak se nám zobrazí následující dialog, který můžeme znát z Excelu, pokud jsme již něco zkoušeli.



Zde si vybereme typ grafu a potvrdíme ok. Následně se nám vloží graf a otevře dialogové okno aplikace Excel, která nám umožní zadat data grafu a vypadá následovně.

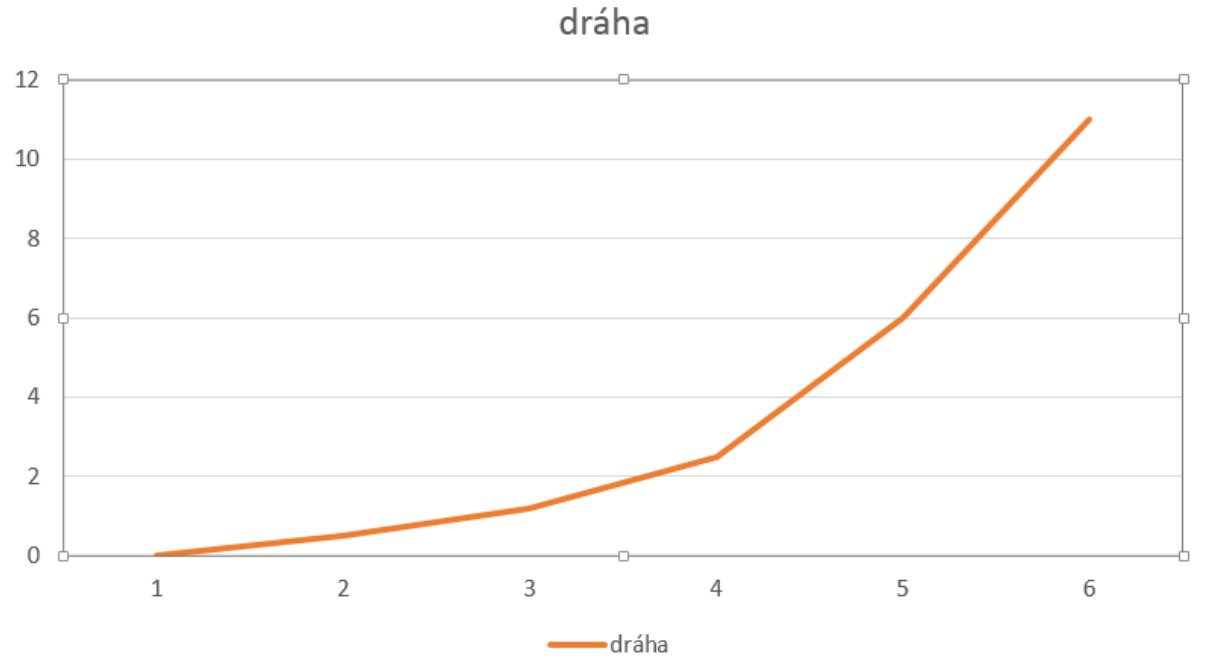


	A	B	C	D	E	F	G	H	I	J
1		Řada 1	Řada 2	Řada 3						
2	Kategorie 1	4,3	2,4	2						
3	Kategorie 2	2,5	4,4	2						
4	Kategorie 3	3,5	1,8	3						
5	Kategorie 4	4,5	2,8	5						
6										
7										

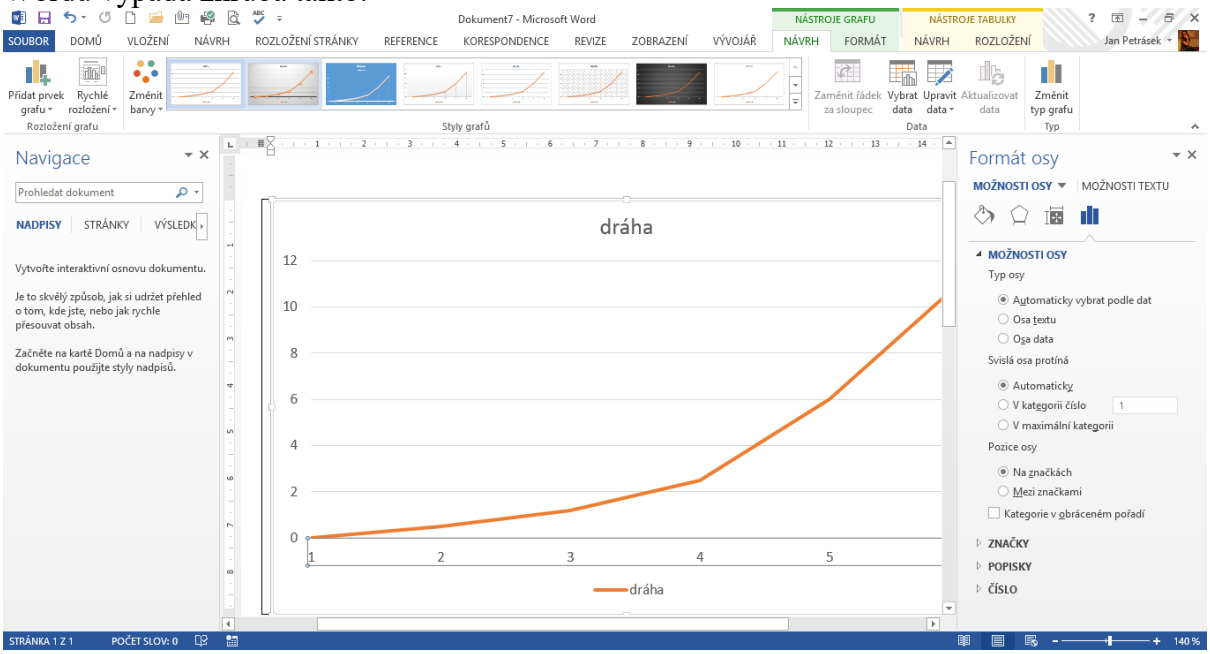
Pokud provedeme vložení dat, tak dostaneme následující:



A graf z toho vykreslený takto:



Jak je na grafu vidět, není zcela odpovídající uvedené zdrojové tabulce. Je to dáno špatným nastavením osy x, které změníme pomocí dvojkliku na osu. Po změně celá obrazovka Wordu vypadá zhruba takto:



Na výše uvedením obrázku též vidíme možnosti, které nám nabízí nástroj pro editaci grafů. Ještě dodám jeho druhou záložku.



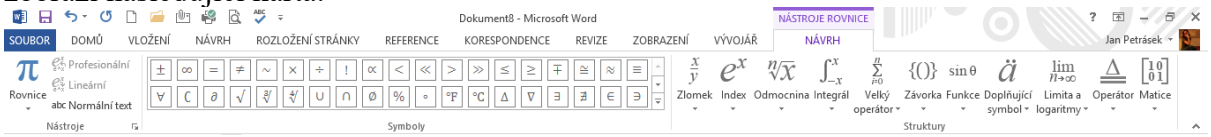
Nástrojů tabulky si nevšímejme, graf je vložený do původní zdrojové tabulky dat, proto tam tyto nástroje nalezneme. Další nástroje pro editaci grafu nalezneme vždy přímo poklikáním na danou část grafu, ať jsou to osy či křivka, nebo název.

Blíže se problematice tvorby grafů budeme věnovat v kapitole věnované Excelu, protože tento tabulkový kalkulátor je základní aplikací pro tvorbu grafů. Word má člověku jen ušetřit práci při upravování drobností, nebo tvorbě velmi jednoduchých grafů.

4.1.13. Editor rovnic

Všichni, kdo již psaly nějaký referát na fyziku, chemii nebo jiný odborný předmět, nebo dokonce již měli tu čest vypracovávat technický či fyzikální protokol, se setkali s problémem úprav rovnic, ať je to psaní odmocnin, lomených výrazů a podobně. Pokud někdo do této kniny nahlédl, protože přechází v rámci Office z verze 2010, tak může tuto kapitolu přeskočit, protože od verze 2010, kdy se povedlo editor rovnic dostat na špičkovou úroveň (nějaká ta moucha se najde, třeba jako bych ocenil jednodušší psaní exponentů) se nic nezměnilo.

Editor rovnic je v této verzi schován obdobně jako ve verzi předchozí. Nalezneme jej na kartě vložení v sekci symboly a je pojmenován jako rovnice. Pokud chceme vložit všeobecný předpis jedné z předdefinovaných rovnic, tak rozbalíme nabídku pomocí šipečky, pokud chceme psát vlastní rovnici, tak klikneme rovnou na nápis rovnice. Po otevření editoru se nám zobrazí následující karta.



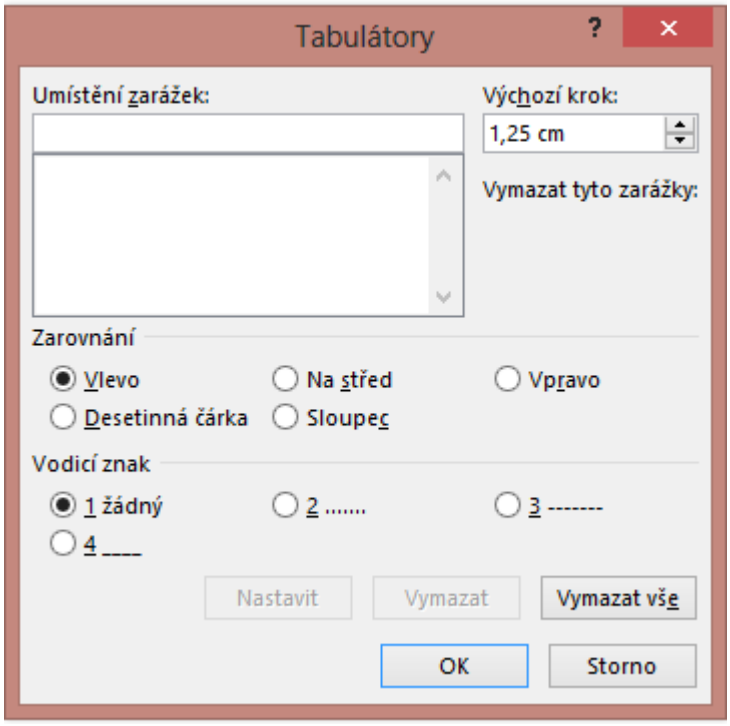
Práce s editorem je velmi lehká, jen je nutné dodržet jedno jednoduché pravidlo, pokud píši odmocninu, index, zlomek, matici, limitu a podobně, tak nejprve zadám operaci a až následně do vyhrazeného políčka vpisuji hodnoty či proměnné. Pokud to chci udělat opačně, tak musím hodnotu nebo proměnnou nejprve označit a pak zadat funkci. Co se týká zarovnání rovnice v textu, je automaticky na střed, pokud jej chci změnit, tak to provedu pomocí karty domů, ale až po zadání celé rovnice. Ven z pole vyhrazeného pro rovnici si kurzor dostaneme pomocí šipky doprava, nebo pomocí dvojice mezer. Rovnice se chová v textu jako nedělitelné slovo, nebo spíše textové pole, může ležet kdekoli v textu, ale dle platných norem ČSN ISO se rovnice umisťuje na samostatný řádek se zarovnáním na střed a na ten samý řádek se umístí k pravému okraji číslo, kterým se v textu odkazujeme na danou rovnici.

Těm šťasným, kteří máte dotykový monitor nebo grafický tablet a disponujete perem, nebo umíte poměrně slušně psát i prstem na monitoru, doporučuji zkusit mžnost rukopisného zadávání rovnice, která je ještě pohodlnější a ryhlejší, nežli její přímá sazba.

4.1.13.1. Tabulátory, aneb jak očíslovat rovnici

Tabulátory jsou mocným nástrojem, který se využívá nejen pro číslování rovnic, ale také třeba při tvorbě ceníků, jídelních a nápojových lístků a podobně. Pro využití tohoto nástroje je nutné mít zobrazené pravítko, což se dá udělat zaškrtnutím na kartě zobrazení. Tabulátory umožňují umístit text na požadované místo na stránce (v rámci její šířky) a také nastavit, zda stanovený bod je středem, koncem, či počátkem umisťovaného textu.

Kompletní nastavení tabulátorů nalezneme v nabídce odstavec (na kartě domů) po otevření dialogového okna, kde vlevo dole je tlačítko tabulátory. Celá vyvolaná nabídka vypadá následovně:



Tabulátor se umisťuje kliknutím na pravítko v místě, kde má být umístěn, je zobrazen například takto:  a kurzor se na něj v rámci řádku přesouvá pomocí stisknutí klávesy tab, tím se odsune kursor na nejbližší napravo ležící tabulátor, pokud je užito výchozí nastavení, tak přitom nevnikne žádná řada znaků mezi původním a novým umístěním kurzoru. Očíslování rovnice vypadá tedy například následovně.

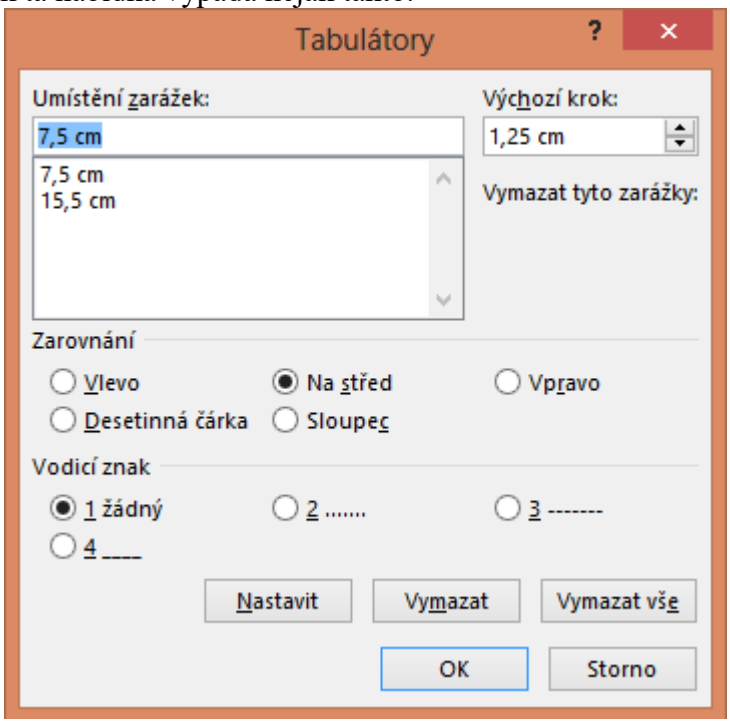


$$S = \pi r^2 \qquad (1)$$

Rovnice je na tomto obrázku ve skutečnosti uprostřed stránky, ale strana není vyfotografována celá. Dvojice tabulátorů je použitá následovně. První tabulátor je zajišťuje vystředění rovnice a pro jeho užití byl kursor umístěn před rovnici a následně byla stištěna klávesa tab, protože je užit středový tabulátor, tak se střed rovnice umístil na značku tabulátoru. Následně jsme kurzor přendali za rovnici (je vidět i na obrázku) a stiskli tab podruhé, čímž jsme se dostali na umístění druhého tabulátoru a napsali číslo rovnice.

V souvislosti s tabulátory jsem vám zapomněl sdělit ještě jednu velmi důležitou věc a to jak se mění jednotlivé typy tabulátorů. V levém okraji pravítka nalezneme tento symbol

na který když klikneme, tak se nám postupně mění symboly pro jednotlivé tabulátory. Tabulátor má symbol vždy černé barvy, ty šedé jsou něco jiného, to vůbec v naší úrovni nebudeme řešit. Další možností je, že tabulátory umístíme na pravítko a vyvoláme kompletní nabídku nastavení, která se ukrývá v položce odstavec. V případě, že již máme umístěné nějaké tabulátory, tak ta nabídka vypadá nějak takto.

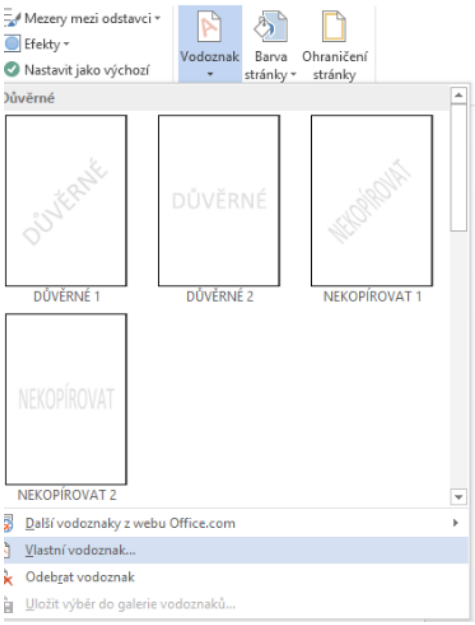


Zde můžeme provést jak změnu umístění, tak připojit vodící znaky i změnit typ tabulátoru.

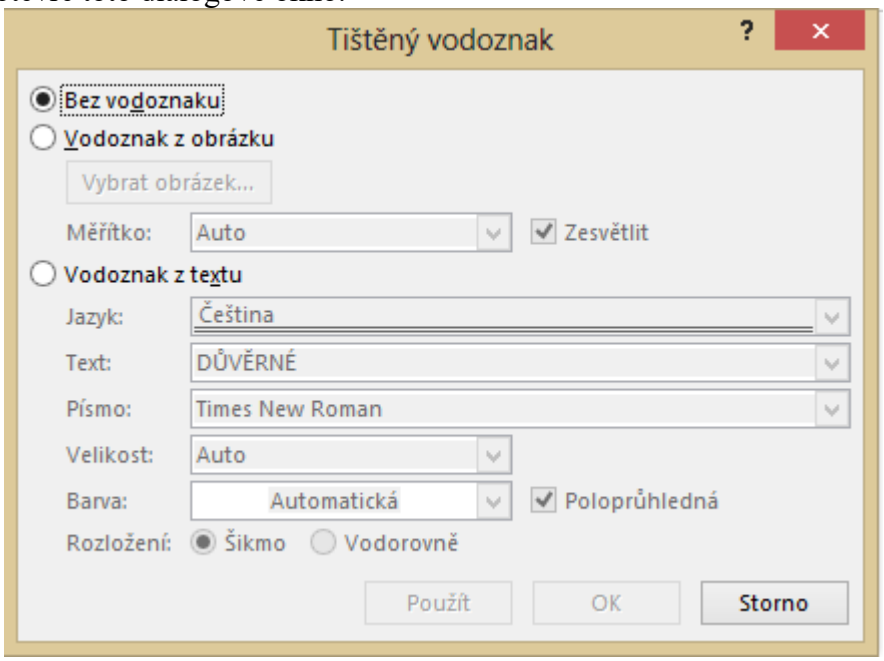
Pokud budeme chtít tabulátor odstranit, tak není nutné vyvolávat toto okno, ale stačí jej uchopit myší a stáhnout z pravítka směrem do papíru.

4.1.14. Pozadí stránky

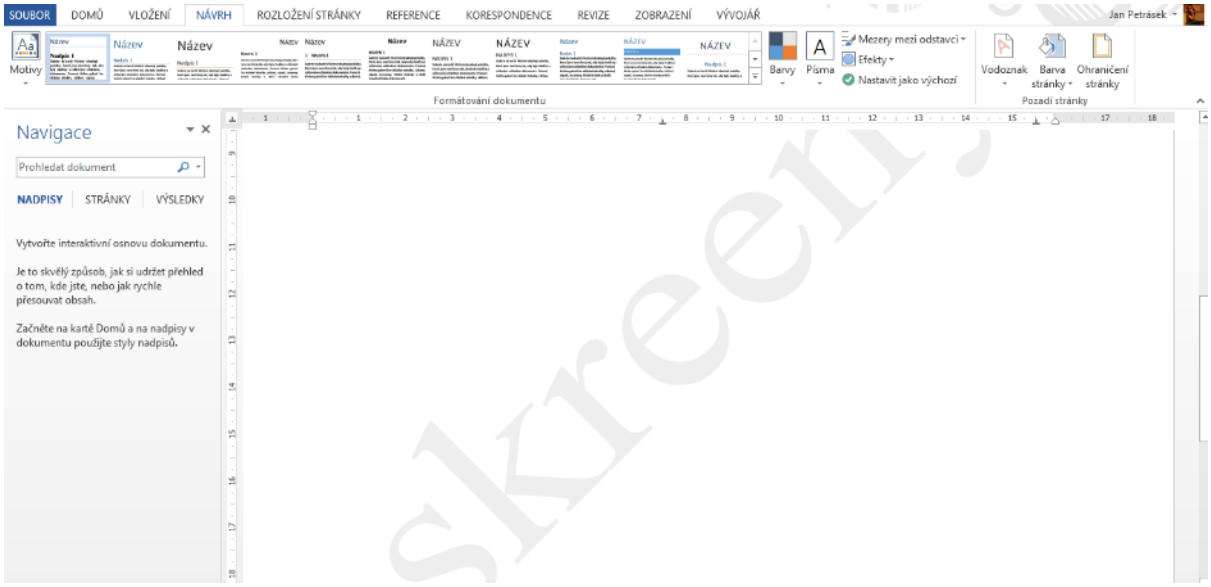
Stránka nemusí mít jen bílé pozadí. Karta návrh nám nabízí rozličné možnosti, které jsou mnohdy velmi užitečné a nejedná se jen o grafické vylepšení dokumentu. Jistě někdy publikujete na webu a nechcete, aby někdo Váš dokument vydával za svůj. Jeden ze způsobů ochrany je přidání vodoznaku do pozadí, kdy vodoznak jednoznačně říká, že dokument je váš, či že je důvěrný nebo se nemá kopírovat. Vodoznak je možné přidat na kartě návrh, kde v sekci pozadí stránky máme položku vodoznak. Zde nalezneme předdefinované vodoznaky, nebo si můžeme vytvořit vlastní.



Pokud zvolíme vlastní vodoznak, tak musíme být struční, musíme si uvědomit, že se nám tam text musí vejít a písmo není žádná 12, kterou píšeme běžně. Volba vlastního vodoznaku otevře toto dialogové okno.



Zvolíme si vodoznak z textu (pokud tedy chceme psát vlastní vodoznak) a rozbalíme menu u položky text, pokud si nevybereme žádnou z nabídek, tak pomocí klávesy back space vymažeme text a vložíme vlastní. Pokud to není nutné, tak nic dalšího neměníme a potvrdíme pomocí použít a ok. Výsledek pak může vypadat např. takto:

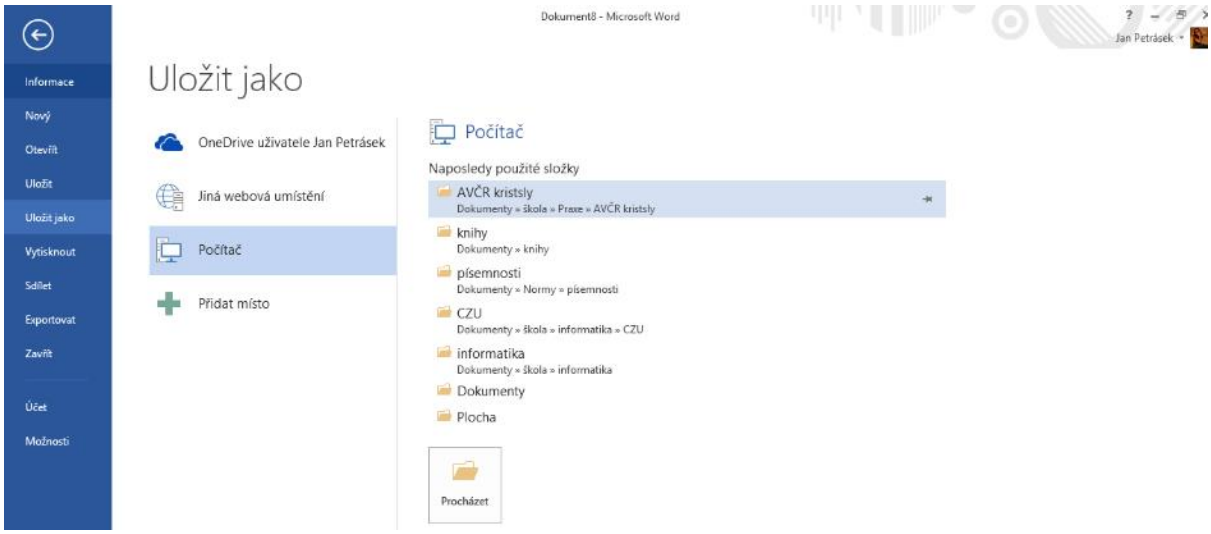


Další, a ne již tolik praktickou možností, kterou nám skupina prvků pozadí nabízí je barva stránky, což jek již název napovídá, umožňuje zbarvit pozadí stránky na jinou barvu, nežli je bílá. Nalezneme zde i spoustu efektů, jako je texturován či přechodová výplň či obrázek v pozadí. Tato možnost je určena především pro reklamní dokumenty, oficiální dokumenty typu diplomová práce tyto prvky nést nemohou.

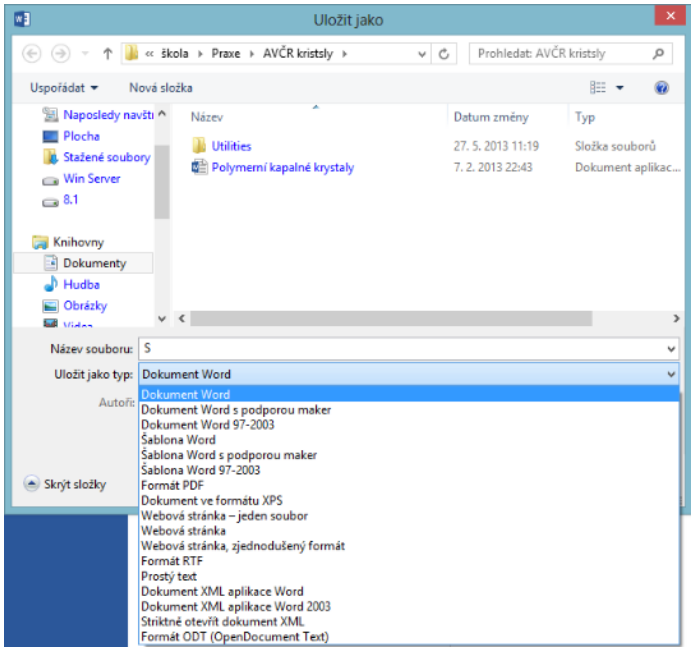
Poslední z trojice možností je ohraničení, které umožňuje tvorbu rámečků či zvýrazněných okrajů stránky. Tento nástroj otevírá následující dialogové okno a pracuje se s ním obdobně jako s orámováním u tabulky, je pouze na kreativitě tvůrce a zadání, jestli tuto možnost využije, a jakým způsobem.

4.1.15. Uložení do jiných formátů

Aplikace Word kromě svého hlavního formátu docx a starého doc nabízí uložení i do spousty dalších formátů, jako jsou: pdf, xps, html, rtf a odt. Na rozdíl od staré verze není nutné provádět nějaký export, ale prostě jen dáme soubor => uložit jako a zobrazí se nám zhruba něco takového:



Když vybereme adresář, kam chceme dokument uložit, tak se nám zobrazí staré známé dialogové okno:



Kde rozklikneme typ (jako je to na obrázku) a vybereme požadovaný formát, pak jen dáme uložit. Pozor, tuto cestu je třeba absolvovat i v případě, že pomocí Wordu upravujeme pdf soubor. Teď bych vám rád řekl něco málo o trojici nejčastějších formátů, které jsou výstupem aplikace Word.

4.1.15.1. Formát docx

Office Open XML (OOXML) je specifikace pro souborový formát na ukládání dokumentů kancelářských balíků jako textových dokumentů, tabulek či prezentací. Formát byl navržen společností Microsoft a poprvé byl použit v Microsoft Office 2007. Finální specifikace byla vytvořena pod hlavičkou sdružením ECMA ve skupině TC45, do které například patří Apple, Barclays Capital, BP, Britská knihovna, Essilor, Intel, Microsoft, NextPage, Novell, Statoil, Toshiba a Knihovna kongresu USA pod číslem ECMA-376.

Formát Office Open XML je soubor ZIP, který v sobě obsahuje XML a další potřebné soubory. Výsledkem jsou tak menší soubory než ty binární, které byly vytvářeny předchozími verzemi Microsoft Office. Podle vyjádření společnosti Microsoft je hlavním cílem formátu zpětná kompatibilita s existujícími dokumenty a plná podpora jejich rozšiřujících vlastností.

Specifikace je rozdělena do několika částí:

- ✓ Open Packaging Conventions (OPC) – popisuje vnitřní strukturu souborů
- ✓ WordProcessingML – popisuje XML pro textové procesory
- ✓ SpreadsheetML – popisuje XML pro tabulkové kalkulátory
- ✓ PresentationML – popisuje XML pro prezentace
- ✓ DrawingML – popisuje XML pro grafické elementy dokumentů
- ✓ Podpůrné „ML“ – sada XML pro velmi specifické úkoly, např. matematické vzorce
- ✓ Custom XML - definuje způsob jak vkládat uživatelská XML do dokumentu a propojovat je s obsahem

Open Packaging Conventions (OPC) popisuje vnitřní strukturu archívu ZIP, který obsahuje dokument. Základním prvkem jsou soubory s příponou .rels, které udržují informace o vztazích mezi ostatními soubory. Příkladem může být vztah mezi sešitem v tabulkovém kalkulátoru a listy, které sdružuje. Pomocí OPC lze vytvářet i vlastní formáty, protože se jedná o obecnou specifikaci. K obsahu dokumentu lze přistupovat dvěma způsoby – s využitím OPC nebo přes adresářovou strukturu archivu.

WordProcessingML struktura vychází ze schématu body – paragraph – run. Tělo dokumentu je děleno na odstavce (text, obrázek, tabulka, …) a tyto části jsou dělené na tzv. runs. Zjednodušeně řečeno, run nastavuje formátování informace. Všechny textové informace, které se mají zobrazit uživateli, jsou umístěny vždy v elementu <w:t>.

SpreadsheetML dokument tabulkového kalkulátoru se nazývá sešit (workbook) a skládá se s listů (sheet). Sešit i listy jsou umístěny v samostatných souborech, typicky workbook.xml, sheet1.xml, sheet2.xml, … Velmi specifické je ukládání textové informace v tabulkách. Existují dva způsoby – inlinestring nebo sharedstring. Inlinestring ukládá informace přímo v buňce tabulky. Tento způsob je v podstatě identický s html. Sharedstring ukládá texty do samostatného souboru, který je sdílený pro všechny listy a každý textový obsah buňky je v něm unikátní. Buňky se pak odkazují na hodnotu v tomto seznamu.

PresentationML obsah dokumentu s prezentací se dělí na snímky (slide), jejich obsah a animace. Každý snímek je opět uložen v samostatném XML souboru. To samé platí např. i pro poznámky ke snímkům.

DrawingML DrawingML popisuje vektorovou grafiku, která se používá v rámci WordProcessingML, SpreadsheetML a PresentationML. Kromě DrawingML se pro vektorovou grafiku používá i VML, které má být nahrazeno právě DrawingML.

Během standardizace u ISO došlo k několika podstatným změnám ve standardu ECMA-376 na základě požadavků národních standardizačních institutů. U nás se touto problematikou zabývá ČNI. Tou nejdůležitější změnou je definování dvou typů dokumentů, podobně jako u XHTML, Strict a Transitional. Verze Strict neobsahuje žádné elementy, které jsou zatížené zpětnou kompatibilitou směrem ke starším dokumentovým formátům. Verze Transitional pak může obsahovat elementy, které jsou pro zpětnou kompatibilitu nezbytné. Tyto verze byly schváleny jako samostatné části standardu ISO pod číslem ISO/IEC 29500. Ve standardu se přímo uvádí, že verze Transitional slouží jen pro přechodné období a neměla by se používat pro vytváření nových dokumentů.

4.1.15.2. Formát pdf

PDF (zkratka anglického názvu Portable Document Format – Přenosný formát dokumentů) je souborový formát vyvinutý firmou Adobe pro ukládání dokumentů nezávisle na softwaru i hardwaru, na kterém byly pořízeny. Soubor typu PDF může obsahovat text i obrázky, přičemž tento formát zajišťuje, že se libovolný dokument na všech zařízeních zobrazí stejně. Vytvářet PDF dokumenty lze jak v komerčním softwaru Acrobat od Adobe, tak v dalších programech (často však pouze jako export do PDF). Prohlížení je snazší, existují volně dostupné prohlížeče pro mnoho platforem, nejznámějším je oficiální prohlížeč mateřské firmy Adobe Reader. Některé aktivní typy obsahu (jako například interaktivní formuláře, 3D grafika, videa, zvuk) nejsou v mnohých PDF prohlížečích podporované. PDF soubory mají příponu .pdf popřípadě .PDF. PDF je otevřeným standardem a je snadno přenositelný (jeho reprodukce je nezávislá na použitém softwaru i hardwaru), i proto je velice rozšířený a hojně využívaný. 1. července 2008 byl tento formát publikován jako standard ISO 32000-1:2008 (vycházející z PDF 1.7). PDF soubory vytvořené podle ISO normy nadále umožňují použití proprietárních technologií definovaných jen firmou Adobe (např. Adobe XML Forms Architecture, Adobe JavaScript), přičemž v rámci ISO probíhá snaha o jejich standardizaci.

4.1.15.3. Formát XPS

XPS neboli „XML Paper Specification" (v začátcích byl znám pod kódovým označením „Metro") je formátem určeným k reprezentaci dokumentů. Jako takový je součástí nové grafické architektury Windows Presentation Foundation (WPF) uvedené spolu s operačním systémem Windows Vista v roce 2007. Zde má několik způsobů užití. Především slouží jako formát souborů tiskové fronty (spool file), tj. formát, ve kterém jsou předávány dokumenty z aplikací tiskovým driverům systému, či též přímo coby jazyk pro popis stránky řídící chod

tiskárny – na trhu přibývá stolních tiskáren (kancelářský, ale třeba i velkoformátový tisk) schopných takto nativně s XPS pracovat (produkty Canon, Epson, HP, Xerox aj.).

Z hlediska polygrafie významnější využití má pak XPS coby formát pro univerzální, na aplikaci a platformě nezávislou elektronickou reprezentaci dokumentů sloužící k výměně dokumentů mezi uživateli, archivaci, elektronickém publikování apod. Tímto způsobem nasazení se XPS zásadním způsobem podobá Adobe PDF. Na rozdíl od PDF je ale XPS v současnosti dobře použitelný pouze k reprezentaci dokumentů kancelářských aplikací, typicky dokumentů Microsoft Office. Takovéto XPS dokumenty lze použít coby podklady pro tiskové zakázky, a to způsoby, které naznačíme v dalším výkladu.

Vytvořit XPS reprezentaci dokumentu libovolné aplikace podporující tisk je ve Windows Vista možno s pomocí zabudovaného virtuálního tiskového driveru (Microsoft XPS Document Writer). K prohlížení XPS lze použít XPS Viewer dostupný skrze Internet Explorer 7 či samostatně. Příslušné nástroje jsou (po instalaci platformy .NET Framework 3.0 a Internet Exploreru 7 či balíku „XPS Essentials Pack") k dispozici i pro starší podoby Windows, tedy Windows XP a 2003 Server. Specifické možnosti převodu do XPS v aplikacích Microsoft Office 2007, 2010 a 2013 + 365, (Windows Vista i XP) poskytuje dále plugin Save as XPS.

Použití XPS nekončí u technologií Microsoftu. Vzhledem k tomu, že Microsoft zveřejnil specifikaci XPS k volnému užití bez poplatků (royalty free) a XPS je založen na všeobecně podporovaném standardu XML, lze poměrně snadno vytvářet XPS aplikace v podstatě pro libovolnou platformu. Již dnes jsou takto k dispozici nástroje nejen pro Windows, ale i OS X či Linux. Další podpoře by měla výrazněji pomoci připravovaná standardizace XPS prostřednictvím organizace Ecma.

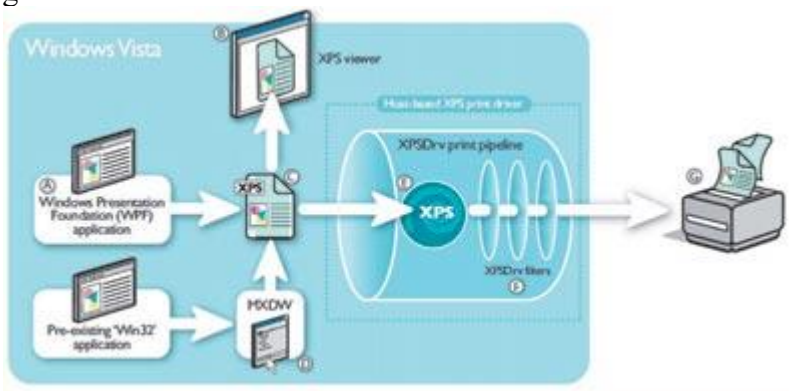


Schéma naznačující způsoby fungování XPS v Microsoft Windows Vista (Převzato z materiálů společnosti Global Graphics)

4.1.15.3.1. Ke schopnostem formátu

XPS dokument je vlastně souborovým archivem komprimovaným algoritmem ZIP, ve kterém jednotlivé soubory nesou informace o obsahu stránek dokumentu. K popisu (layout, text, vektorové objekty aj.) je použito jazyka XML, dále jsou součástí archivu soubory použitých bitmapových obrázků či písem (OpenType), ICC profily spojené se správou barev, print tickety, metadata aj. Daný způsob vyjádření má řadu výhod: založení na XML znamená možnost využití nepřeberného množství nástrojů pro daný standard umožňujících mimo jiné snadný vývoj aplikací pro zpracování XPS. Popsaná hierarchická struktura pak dovoluje snadnou manipulaci s komponentami dokumentu jak v odpovídajících nástrojích, tak třeba i s použitím obvyklých prostředků pro správu souborů a ZIP archivů (možnost snadno extrahovat či nahradit použité obrázky apod.).

Způsob, jakým má být XPS dokument tištěn, je popsán v tzv. print ticketech vložených do dokumentu. Tyto jsou zaměřeny na aplikace ve světě stolního tisku, při zpracování XPS na pokročilejším tiskovém zařízení by měly být ignorovány.

Unikátní vlastností XPS je podpora formátu Microsoft HD Photo (resp. JPEG XR), který by se mohl díky řadě zajímavých vlastností (efektivní komprese, 32bitová barevná hloubka na kanál, HDR aj.) výhledově stát nástupcem formátu JPEG při reprezentaci bitmapové grafiky, zejména digitálních fotografií. Barvy mohou být v XPS dokumentu jak v RGB (sRGB, scRGB), tak i CMYK, přímé či N-Channel, oproti PDF chybí podpora Gray, Lab a ICC-RGB. Ke správě barev v XPS dokumentech je použito vlastního systému Windows Vista WCS (Windows Color System), barvám dokumentu jsou nicméně (povinně) přiřazeny ICC profily použitelné standardními způsoby. XPS podporuje rovněž barevné přechody a základní nasazení průhlednosti. Z hlediska elektronické výměny XPS dokumentů je důležitá podpora digitálních podpisů a systému správy autorských práv v daném formátu.

4.1.16. Poznámky pod čarou

Když jsem přemýšlel, kde zakončím naše putování aplikací MS Word, protože je to natolik obsáhlá aplikace, že vydá na celou knihu, tak mě napadlo to putování zakončit právě tady, u poznámek pod čarou. Důvod je jednoduchý, tato kapitola by se také dala smrsknout jako jedna drobná poznámka pod čáru. Určitě ještě máte v hlavě citace, které jsme si zkoušeli pár podkapitolek zpátky, poznámky pod čarou fungují dá se říct stejně a také proto je nalezneme hned vedle citací na kartě reference. Přidat poznámku pod čarou není těžké, vlastně stačí jen kliknout na vložit pozn. pod čarou a na místě kurzoru se objeví v horním indexu číslo, a kurzor se přenese k tomuto číslu na dolní okraj stránky pod čáru, kam napíšeme poznámku a poté kurzor přesuneme zpět a pokračujeme v psaní.

4.2. PowerPoint

Program PowerPoint slouží k tvorbě prezentací, které zobrazijí výsledky výzkumu, bádání či práce v jiném programu (např. Excel). Hlavním úkolem programu je zpřehlednění výstupu a přidání grafiky a popisků tak, aby byl výstup srozumitelný pro další lidi a nějak tu činnost propagoval. Myslím, že každý z nás někdy viděl, jak vypadá prezentace, nebo často jen pokus o ni, minimálně od nějakého učitele. V této kapitole si prozradíme, jak se taková prezentace tvoří, jaké jsou možnosti programu PowerPoint a ukážeme si i nějaké triky, ale pro ty znalejší, kteří něco již umí, to opět bude nuda, protože obdobně jako u Wordu se bude jednat spíš o nějaké základní seznámení.

4.2.1. Pravidla pro tvorbu

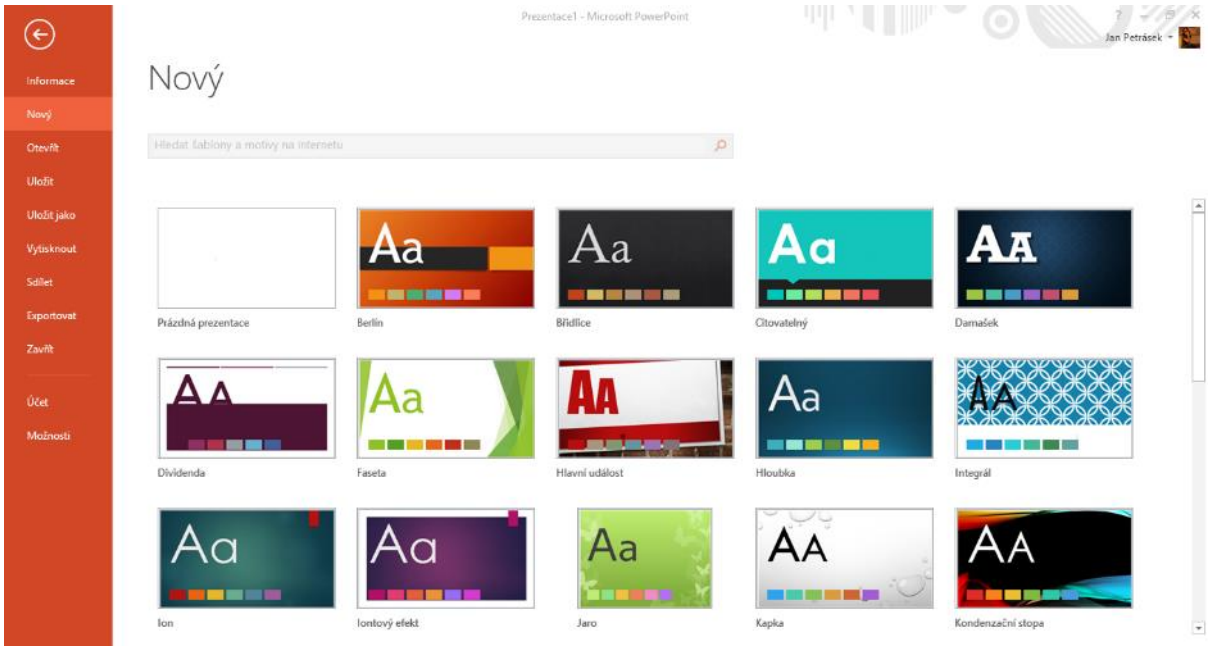
Dříve, nežli se můžeme pustit do tvorby prezentace, tak musíme znát nějaké základní věci, které třeba dodržet, aby prezentace byla použitelná. Především je nutné si uvědomit, že prezentace je často promítána pomocí dataprojektoru, který nemá tak přesné barvy a nedosahuje tak kvalitního vykreslení barevných tónů, celkově mívají nižší kontrast. Pak je nutné brát v potaz, že na různých místech je různá verze programu, tudíž nemusí být vše vykresleno tak, jako na naší verzi, k této problematice se vrátíme později. Nebudu již déle se rozepisovat a zdůvodňovat to, co nemáme před očima, a v bodech vám sdělím ona pravidla.

- Užívat kontrastní písmo s podkladem (např. černé pozadí a bílé či žluté písmo)
- Dostatečně velké písmo (běžný text 30 a více, nadpis 42 a více)
- Nepoužívat velké množství animací
- Psát stručně a jasně (ideálně v bodech)
- Užívat multimediální podklady (video a zvuk)
- Dávat si pozor na kompatibilitu mezi sadami Office (starý a nový formát (2007 a novější))
- V režimu kompatibility se starší sadou nepoužívat automatický přechod po určitém čase (lépe se mu vyhnout pokud je prezentace doprovázena komentářem či více sdílena)
- Při automatickém přechodu dbát na to, aby nebyla doba na přečtení snímku příliš krátká, ale ani příliš dlouhá a vždy nechávat možnost přechodu při kliknutí
- Na výukové prezentace nepoužívat automatické přechody po čase
- Užívat dobře čitelné řezy písma a zvýraznění částí textu (psací písma mohou být jen na nadpisy nebo úvodní snímek, pokud je to spíše zaměřeno na grafiku, jinak doporučuji Times New Roman, Arial apod.)
- Na úvodním snímku kromě názvu prezentace uvést i jméno a rok
- Na druhý snímek uvést údaje o firmě či akci, pokud jde o prezentaci určenou jako firemní či pro jednu jedinou konkrétní akci
- Na druhý či třetí snímek (dle konstrukce prezentace) je vhodné uvést hypertextovou osnovu prezentace
- Na posledních snímcích uvést zdroje v souladu s normou ČSN ISO 690 pro knihy a webové zdroje jako seznam odkazů přímo na využitý materiál
- Jako externí soubory užívat video či audio, nebo webovou stránku, poté pozor na cestu k souboru pokud je prezentace sdílená
- Není vhodné jako externí soubor užívat pdf, pokud se nejedná o online dostupné rozšiřující materiály, které nejsou přímou součástí prezentace
- Užívat kvalitní obrázky (nutné si uvědomit, že obrázky budou ještě výrazně zvětšeny)
- Každý snímek musí mít nadpis, pokud jde o pokračování textu z předchozího snímku, tak je nadpis stejný. Nadpis by měl být i na snímcích, kde je jen fotografie, ale tam lze odpustit

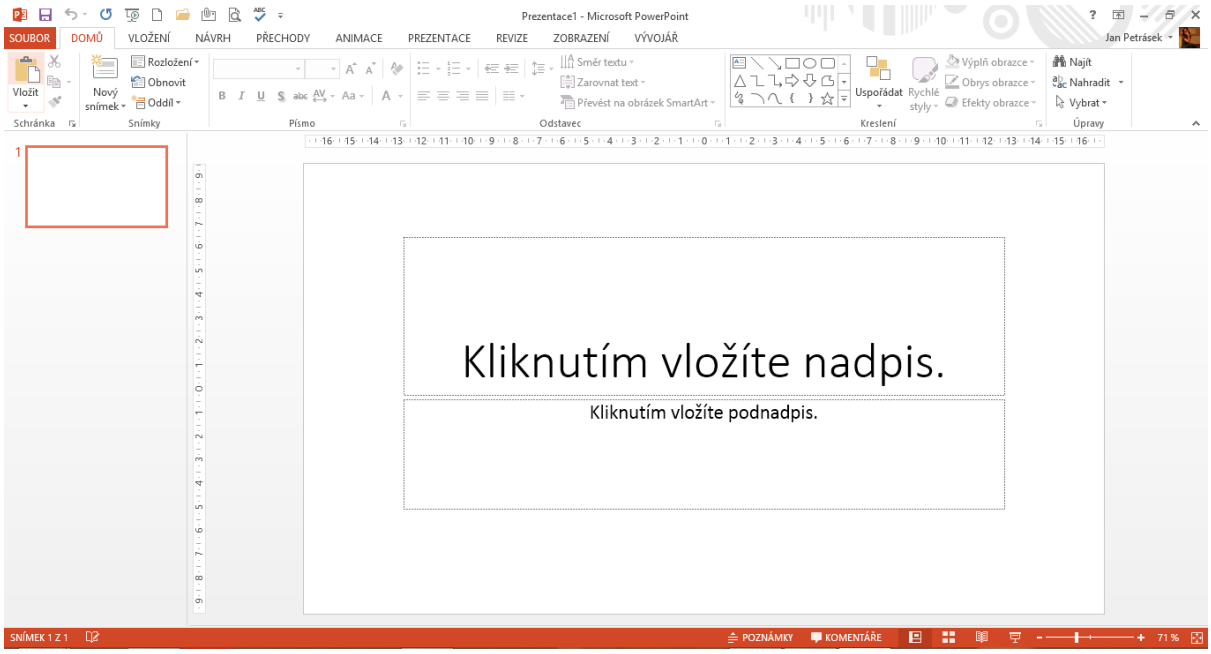
4.2.2. Krátké představí

Obdobně jako v případě Wordu si nejprve uvedeme přehled umístění jednotlivých prvků v aplikaci. S ohledem na to, že jste pilní studenti, tak již nebudu uvádět celé cesty v dalších kapitolkách a je na Vás, abyste si to našli v programu, nebo v této kapitolce.

PowerPoint je jednoduchá, ale velmi mocná aplikace, která nás uvítá následujícím oknem.



Zde vidíme přehled předpřipravených šablon, které nám PowerPoint pro naší tvorbu nabízí. Pokud chceme být originální, nebo jen dobře čitelní, tak volíme hned první, tedy prázdnou šablonu.



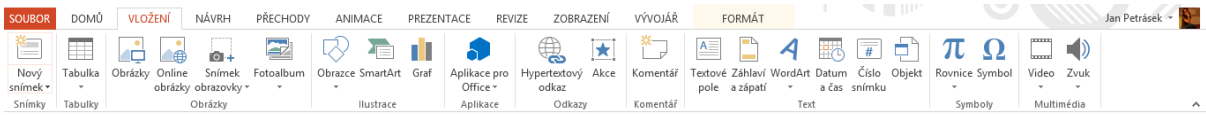
4.2.2.1. Karta Domů

Obdobně jako v případě Wordu nám umožňuje karta domů provádět základní úkony při tvorbě. Obsahuje následující sektory a možnosti:

- ✓ Schránka (vložení, kopírovat formát, vyjmout a kopírovat)
- ✓ Snímky (nový snímek, rozložení, obnovit, oddíl (seskupení snímků do oddílů))
- ✓ Písmo (velikost, font, tučné, podtržené, kurzíva, stínované, barva, prodloužení znaků, přeškrtnutí, vymazání formátování, záměnu velkých a malých písmen ve vybrané oblasti)
- ✓ Odstavec (odrážky, číslování, zarovnání, řádkování, směr textu, zarovnání textu, převod na obrázek SmartArt)
- ✓ Kreslení (obrazce, uspořádat, rychlé styly, výplň obrazce, obrys obrazce, efekty obrazce)
- ✓ Úpravy (najít, nahradit, vybrat)

Popisovaná karta je vidět na obrázku nad podkapitolkou.

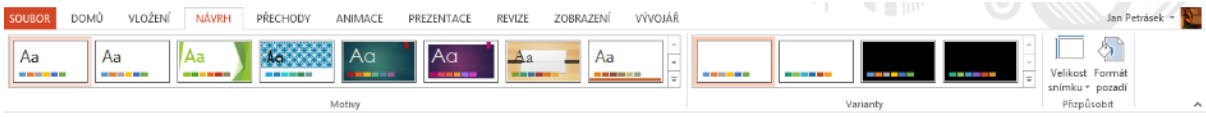
4.2.2.2. Karta vložení



Obdobně jako u Wordu jde o velice důležitou kartu a obsahuje podobné položky, pojďme si ji tedy společně projít.

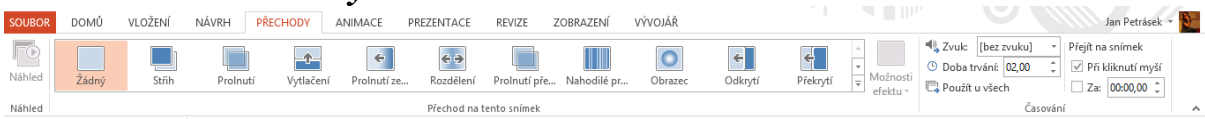
- ✓ Snímky (nový snímek)
- ✓ Tabulky (tabulka – stejný nástroj jako ve Wordu)
- ✓ Obrázky (obrázky, online obrázky, snímek obrazovky, fotoalbum)
- ✓ Ilustrace (obrazce, SmartArt, graf – stejný nástroj jako ve Wordu)
- ✓ Aplikace (Aplikace pro Office)
- ✓ Hypertextový odkaz
- ✓ Komentář
- ✓ Text (textové pole, záhlaví a zápatí, datum a čas, číslo snímku, objekt)
- ✓ Symboly (rovnice, symbol)
- ✓ Multimédia (video, zvuk)

4.2.2.3. Karta návrh



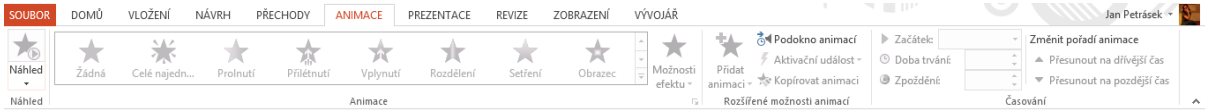
Umožňuje využít předefinované motivy a upravovat je, či vytvářet vlastní pozadí snímku. K této kartě nebudu uvádět bodový výčet, protože vše je vidět na obrázku a v programu, a také bylo řečeno v předchozí větě.

4.2.2.4. Karta Přechody



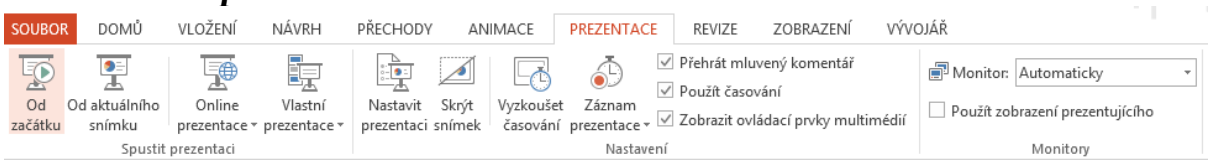
Umožňuje nastavit styl záměny snímků a upravovat jej. Zde se mimo jiné nastavuje také čas, po kterém dojde k přepnutí na další snímek, či doba záměny snímků.

4.2.2.5. Karta Animace



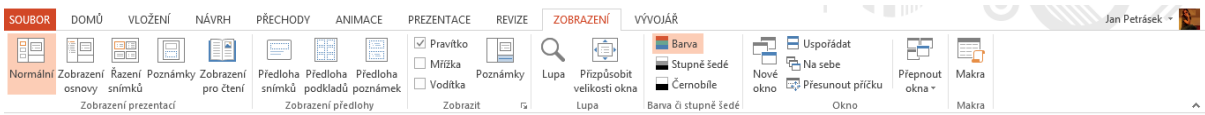
Umožňuje textu a obrázkům přidat další pohyb pro postupné zobrazení, zvýraznění či postupné mizení a nachází se zde i veškeré nastavení těchto efektů včetně časování. Tato karta je dobrá pro nějaká přání, nebo pokud na snímek vkládáme fotoalbum, tak aby se mezi obrázky nějak „listovalo“ graficky pěkně, jinak nedoporučuji příliš užívat, tedy kromě efektů zvýraznění, které je možné užít například u vzorců, a nijak nenarušují strukturu a čitelnost snímku.

4.2.2.6. Karta prezentace



Na této kartě nalezneme přehrávání prezentace a jeho nastavení. Hlavně ti, kdo přednáší v počítačových učebnách ocení možnost online prezentace, která umožňuje sledování prezentace ve webovém prohlížeči. Většinou není třeba zde nic měnit, ale je zde pár vychytávek, ke kterým se vrátíme.

4.2.2.7. Karta zobrazení

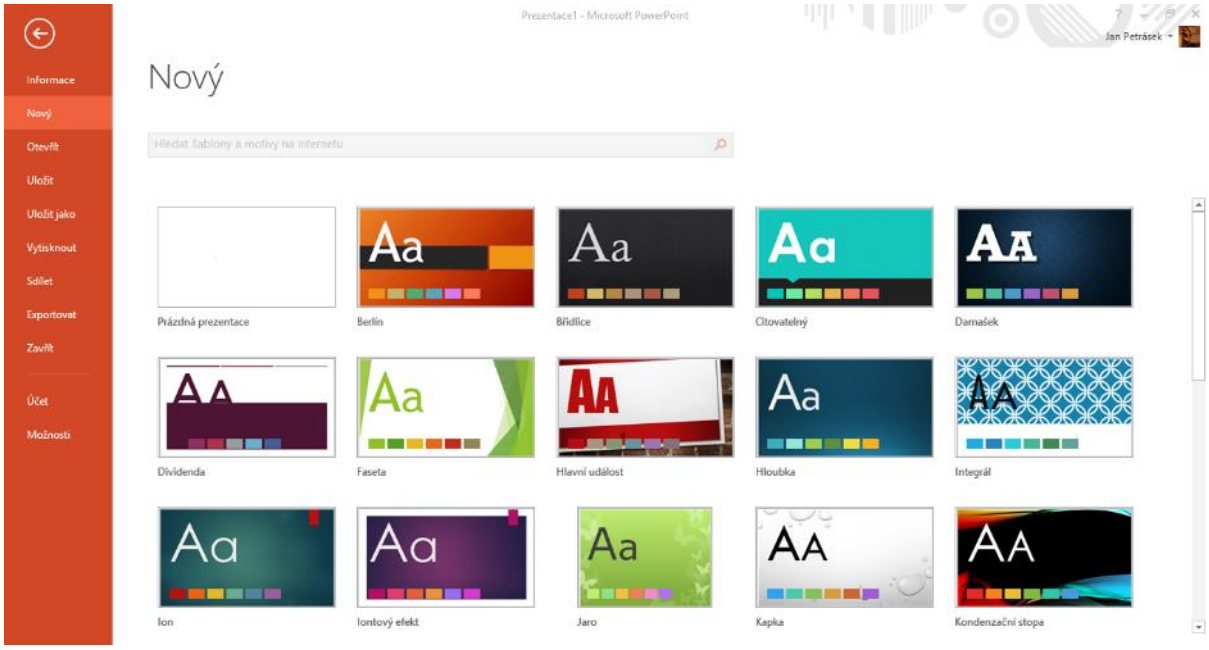


Tato karta umožňuje uzpůsobit nastavení zobrazení při tvorbě, pokud by nevyhovovalo výchozí nastavení, nicméně bych doporučil nejdříve si vyzkoušet funkce programu v základním nastavení.

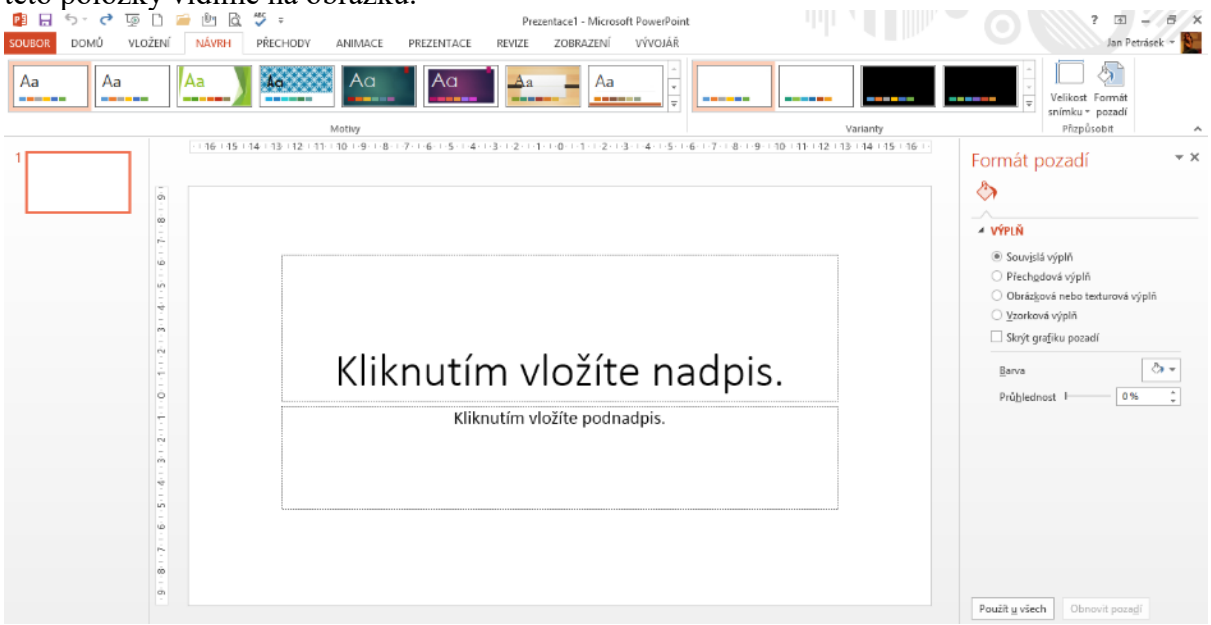
4.2.3. Základní postup tvorby

PowerPoint je velmy rozmanitým a silným nástrojem, který umožňuje každému mít individuální styl práce a různými postupy se dopracovat k prezentaci, která splní jeho požadavky a vize. Úkolem této kapitoly není z vás udělat designery a přeborníky v tvorbě prezentací, úkolem je představit nějaké základní dílčí postupy a možnosti, které si každý skloubí dle svého. Sám tento program využívám velmi hojně, takže je pro mě dost těžké, vymyslet odkud začít a jak to popsat. Práce s PowerPointem je intuitivní a velmi lehká, proto se jí věnuje na školách málo pozornosti, což je škoda. Ve skutečnosti ani po několika letech práce sám neznám dostatečně, alespoň mám tento pocit, možnosti tohoto programu. Sestavit opravdu kvalitní prezentaci, hlavně pokud je to prezentace určená pro web, není nic lehkého a vyžaduje to i trošku nadání, které mají většinou lepší ženy, zato muži mají lepší tu technickou stránku, proto není od věci pracovat na prezentacích v páru.

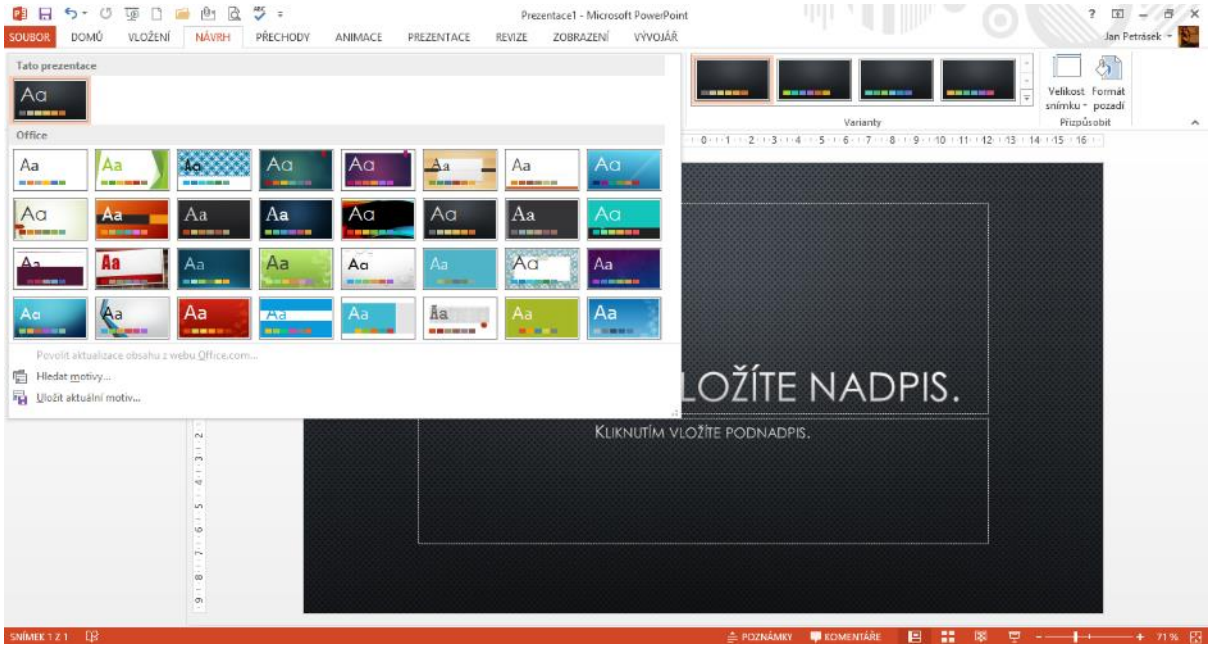
Osobně si na úvodní obrazovce vždy vybírám prázdnou šablonu a pozadí si nastavuji sám, tak tedy v obrázcích.



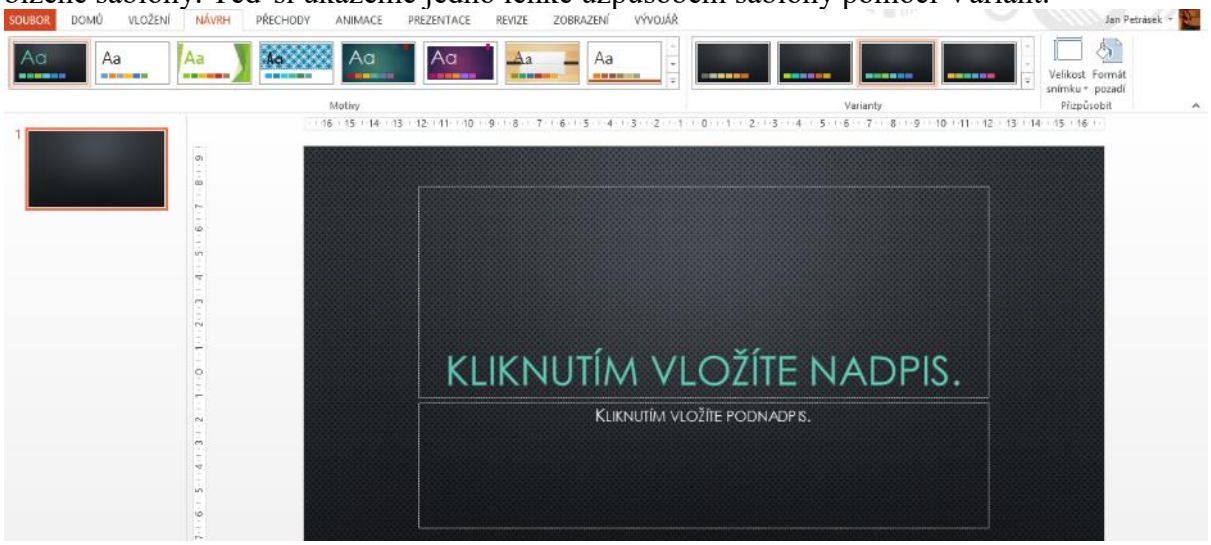
Následně si tedy vytvoříme pozadí prezentace. Nejlépe volíme černou či tmavě modrou, nebo můžeme nechat bílou, i když je lepší mít nějakou barvu v pozadí, aby prezentace vypadala trošku profesionálně a bylo na první pohled vidět, kde je umístěno zorné pole, protože ne vždy je k dispozici plátno, nebo je obraz umístěn přes celé plátno. Pozadí vybereme na kartě návrh, kde máme k dispozici jak předefinované typy (stejně jako v případě úvodní obrazovky) a také další nastavení šablony v sekci varianty. Pokud se nám nebude líbit ani žádné uzpůsobení šablony pomocí sekce Varianty, tak máme k dispozici nabídku formát pozadí. Základní nabídku této položky vidíme na obrázku.



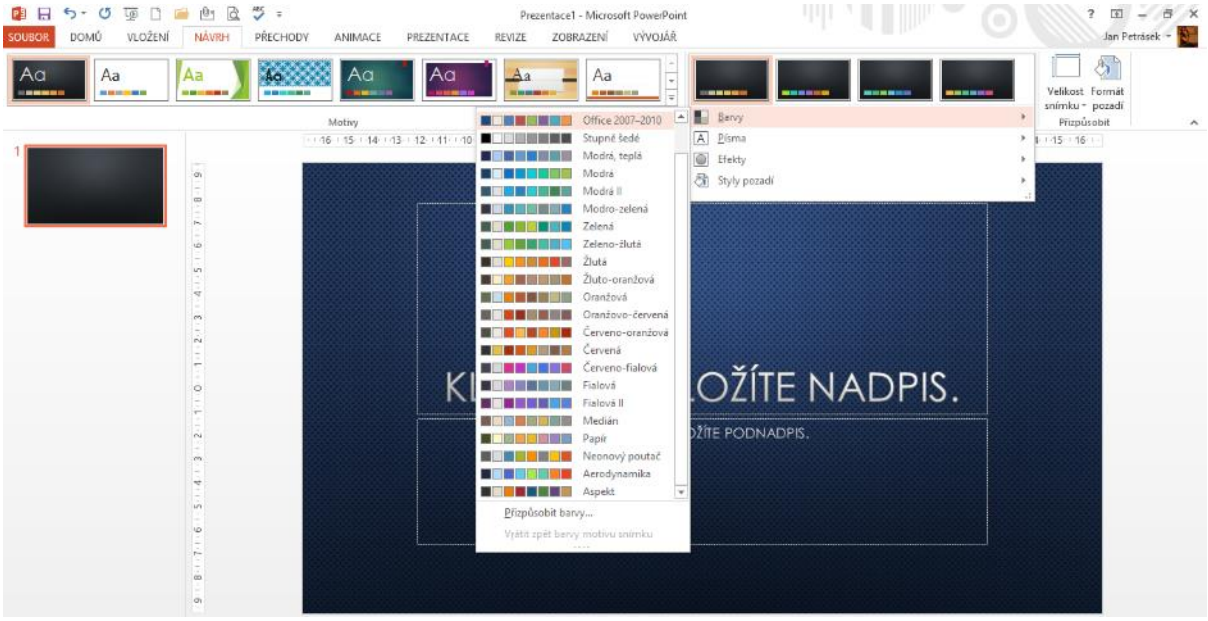
Teď si ukážeme, jak se nám na pár kliknutí změní šablona. Nejprve zvolíme šablonu a poté si ukážeme jednoduchá základní uzpůsobení. Podrobné nastavování pomocí nabídky formát pozadí si ukážeme vzápětí.



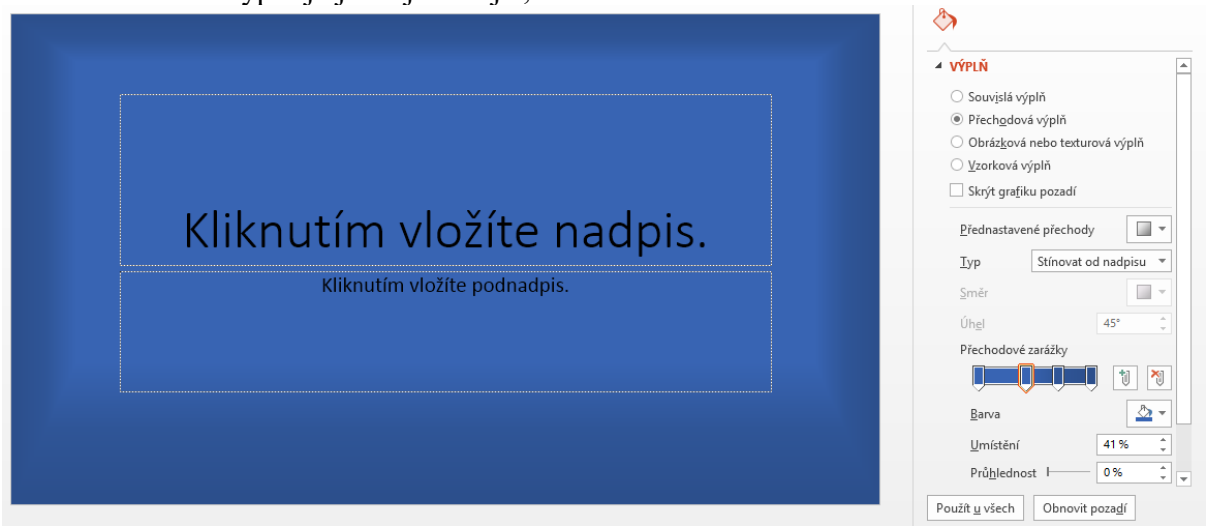
Na obrázku vidíme kromě použité šablony a jejího zobrazení na snímku, také další nabízené šablony. Teď si ukážeme jedno lehké uzpůsobení šablony pomocí Variant.



I Varianty lze dále uzpůsobit, teď si ukážeme další předdefinované možnosti.



Nyní si společně projdeme možnosti nabídky formát pozadí. Tato nabídka v základu dává na výběr ze souvislé, přechodové, obrázkové a texturové, a vzorkové výplně. Souvislou výplň si nebudeme ukazovat, ta je všem jasná, prostě celý snímek má jednu barvu všude stejnou. Přechodová výplň je již zajímavější, na možnosti se koukneme na obrázku.



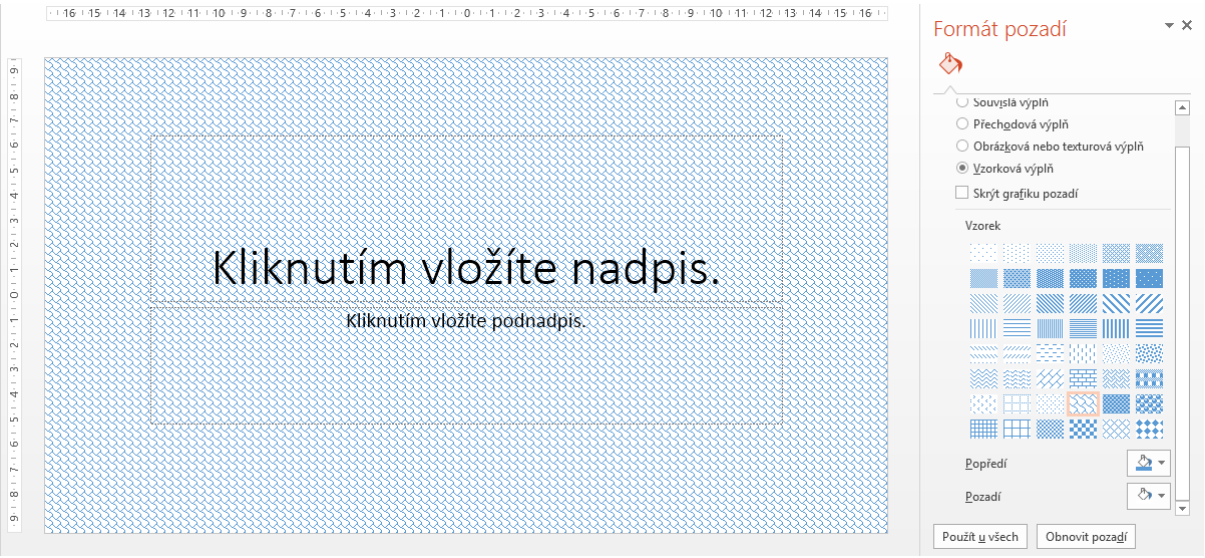
Tady již jde o hraní si s barvami a jejich sytostí. Jako další možnost je texturová či obrázková výplň, kdy je k dispozici spousta zajímavých textur, které můžeme vidět na obrázku.



Nepochybně i hraní si s obrázky na pozadí může být krásné a zábavné, ale musíme celou dobu myslet na to, že na snímku bude text, takže je potřeba, aby byl čitelný. Jeden celkem nevinný příklad je třeba následující.

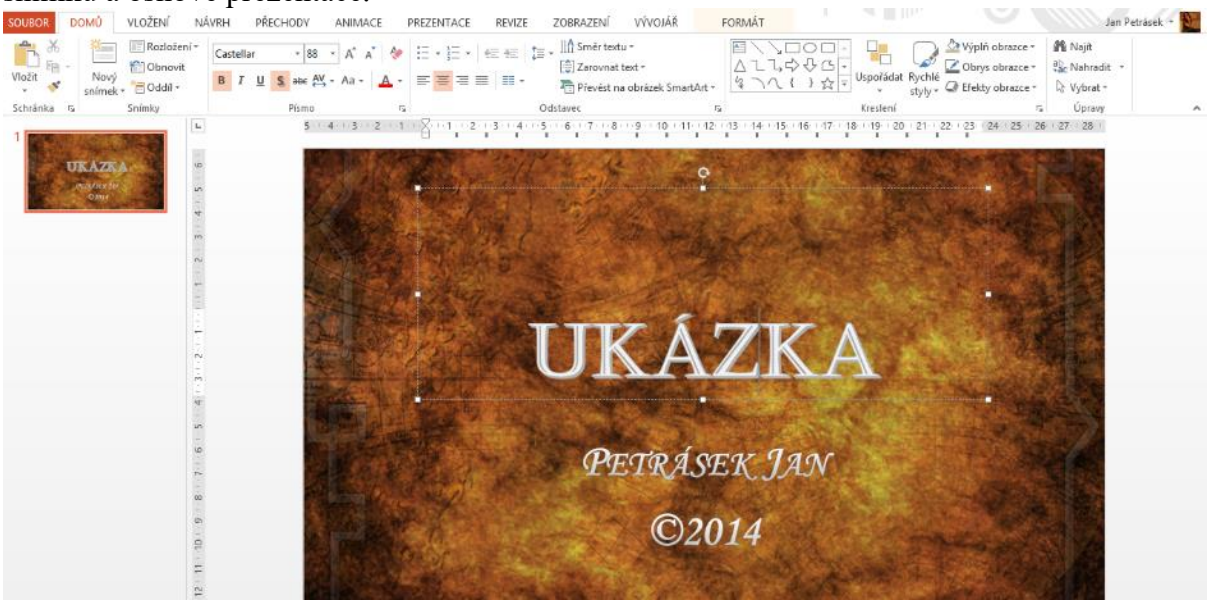


Doporučuji záměnu za obrázek z bílého podkladu a hrát si s průhledností, aby bylo písmo dobře čitelné. A poslední možností, jak udělat pozadí je vzorková výplň.



Zde si kromě vzorku (je vidět oranžový rámeček) vybereme také dvojici barev, barvu pozadí a popředí vzorku.

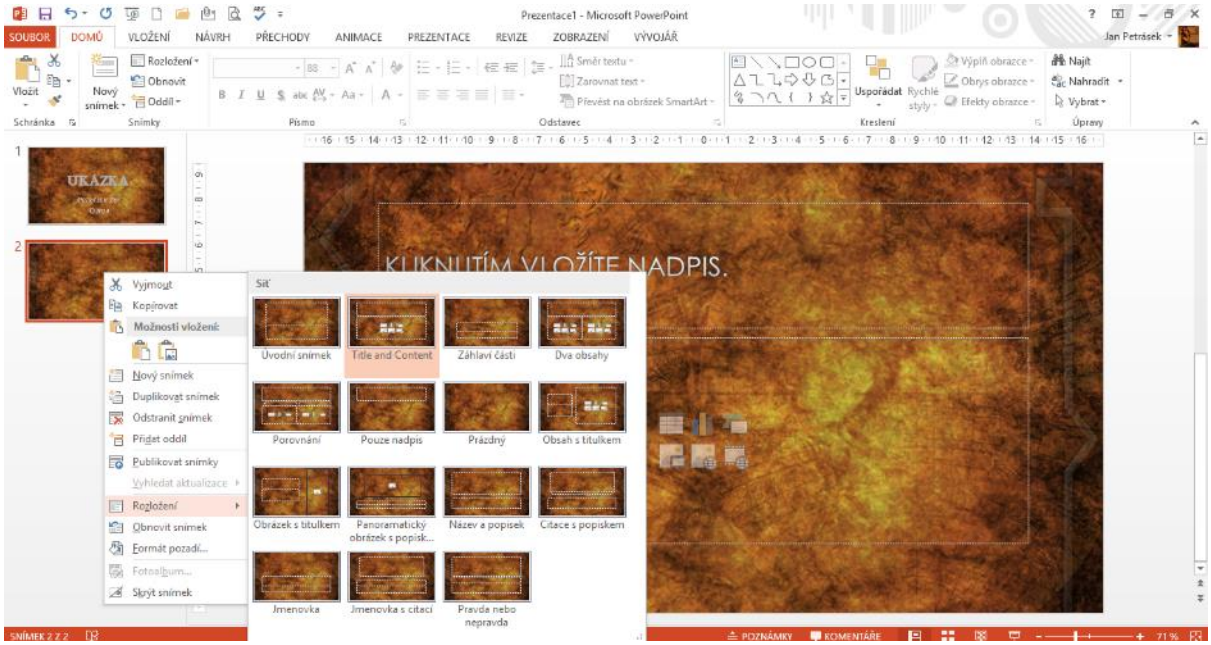
Dalším krokem je vložení a práce s textem. Připravil jsem jeden návrh, který splňuje to, co jsme si říkali v teorii, ale nemusí to písmo vypadat zrovna takto, nicméně je tam jednoznačně vidět jeden ze základních požadavků, čitelnost. Velká a malá písmena je dobré rozlišovat, ale na úvodních snímcích prezentací to nemusí být dodrženo, obdobně jako v nadpisech snímků a osnově prezentace.



Jistě jste si všimli karty formát, která se objevila. Popravdě jsem tak trošku nedočkavě předbíhal, protože jsem ji již užil. Vypadá následovně:



A umožňuje nám kouzla stylů Word Art a další úpravy či efekty textových polí a textu. Pomocí volby nový snímek si přidáme další snímek do prezentace. Rozložení snímku lze upravit na pomocí rozkliknuti nabídky nový, nebo pomocí pravého tlačítka myši, kterým klikneme na snímek v seznamu snímků.



Povšimněme si nabídky duplikovat, která nám vytvoří stejný snímek, čehož lze užít při práci s textem, který se nevejde na jeden snímek. Na duplicitním snímku prostě jen označíme stať a začneme znovu psát, tímto postupem si ušetříme přenášení formátu a také psaní nadpisu, pokud je stejný. Samozřejmě snímek může mít jiné pozadí nežli ostatní snímky. Ale zpět k rozložení snímku, pokud si zvolíme prázdný, tak musíme pomocí textového pole vytvořit vlastní rozložení, nebo užít obrázek jako pozadí jednoho snímku, který pak bude přes celé. Dvousloupcová rozložení snímků se používají pro porovnávání. Jinak snímek není nic jiného nežli určitý druh textového pole, takže jej lze kopírovat a vkládat do jiné prezentace, jako obrázek i do Wordu či Outlooku nebo jen na jiné místo v prezentaci.

V hlavním textovém poli (tzv. těle) nalezneme šestici ikon, které zmizí hned, jak začneme psát první text. Tyto ikony slouží ke vložení multimédií, tabulek a grafů, které se rozměrem přizpůsobí poli. Ony zmíněné iony ukazuje následující obrázek.



Typicky by druhý, respektive třetí snímek, prezentace mohl vypadat třeba nějak takto:

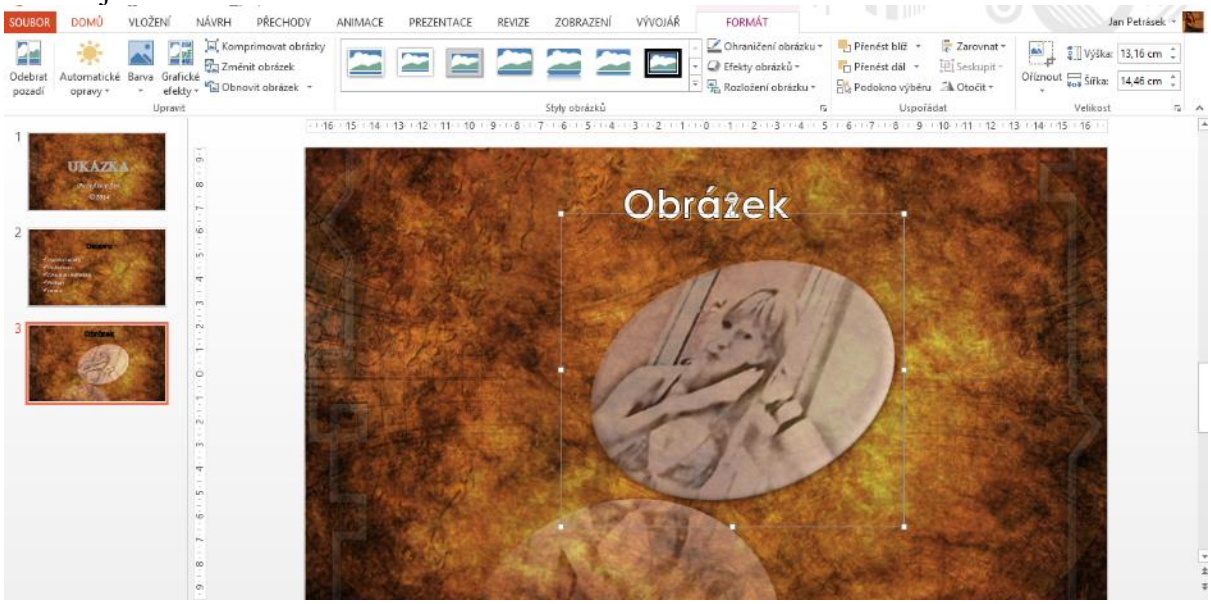


Jak je možné si všimnout ne jen v této knize, dost často se nadpisy středí, ač nastavení všech šablon to dělá jen na úvodním snímku prezentace.

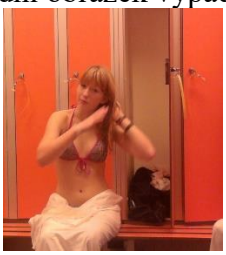
K této kapitolce to bude všechno a zbývající práci si rozdělíme do dalších kapitolek, abychom se snáze orientovali a hledali se.

4.2.4. Obrázky

Pokud jste pozorně četli a vše si zkoušeli ve Wordu, tak teď oceníte sjednocení základního ovládání všech programů Office, protože tuto kapitolu můžete přeskočit. Důvod je prostý, PowerPoint má stejné nástroje pro úpravu a práci s obrázky jako Word. Jen v jedné věci se liší, pokud v PowerPointu vložíte obrázek ze schránky klávesovou zkratkou ctrl+v, tak se vloží obrázek v plné velikosti a většinou se uchytí k pravému hornímu rohu snímku, pokud nepřesahuje velikost snímku, což se může také stát. Takto vložený obrázek se vloží do popředí, tedy překryje všechny objekty na snímku. Pokud obrázek vložíme pomocí pravého tlačítka uvnitř textového pole, tak svou velikost automaticky uzpůsobí rozměrům pole. Pokud pro vložení obrázku užijeme ikonu umístěnou v těle, tak obrázek přesně vyplní tělo, to samé platí při vkládání obrázku pomocí pásu karet. Příklad zpracovaného obrázku za pomocí karty formát je na následujícím obrázku.



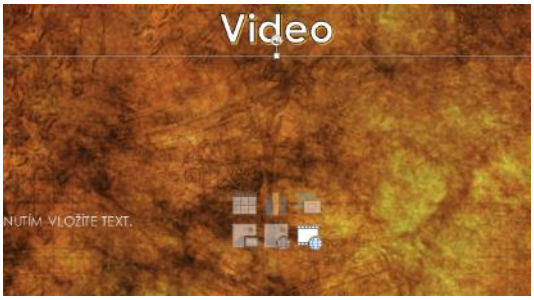
Jinak jen tak pro zajímavost, původní obrázek vypadal takto:



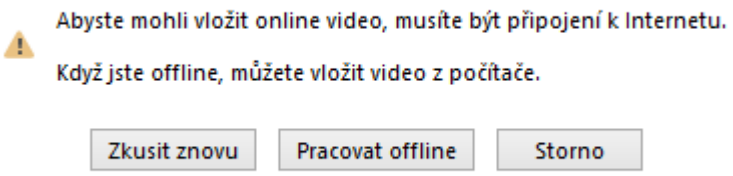
Jak je vidět na příkladu dvojice obrázků, tak nám Office nabízí velké možnosti úpravy a je jen na nás, jak toho využijeme. A jestli si někdo z vás myslí, že takováto proměna byla nějak složitá a trvala dlouho, tak jej mohu ujistit, že proměna nezabrala více jak 20 kliknutí a celkový potřebný čas, pokud se člověk seznámí na nějakém obrázku se všemi možnostmi, je necelé 2 minuty.

4.2.5. Video

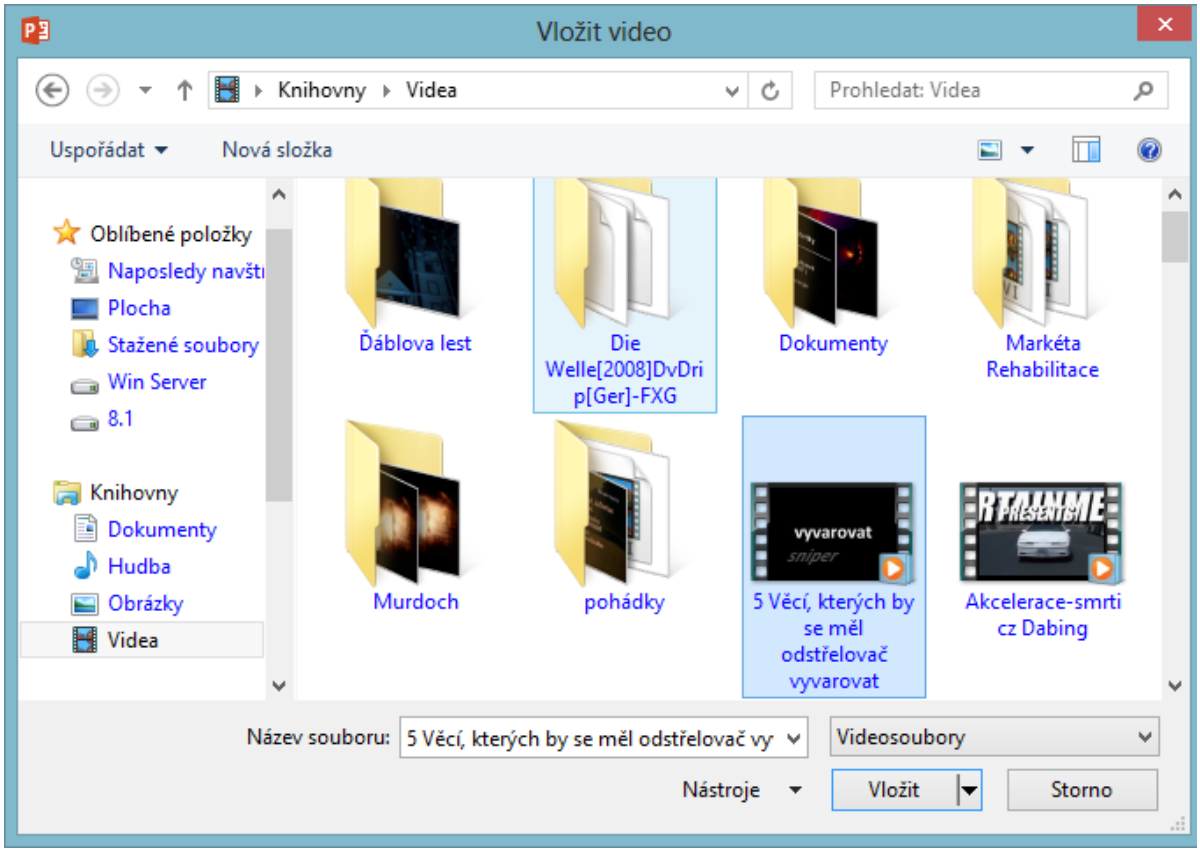
Video tvoří nedílnou součást některých, především obchodních, prezentací. Možnost zpracovat video bylo až do této verze doménou pouze PowerPointu, pokud se budeme držet programů sady Office. Podporovány jsou formáty: asf, avi, MP4, QuickTime movie file, mpeg, wmw, Adobe Flash Media. Délka videa není nijak omezena, jen je nutné, aby pro efektivitu bylo co nejmenší a bylo pokud možno ve stejné složce jako prezentace. Video přidáme pomocí karty vložení, nebo využijeme tlačítek na snímku.



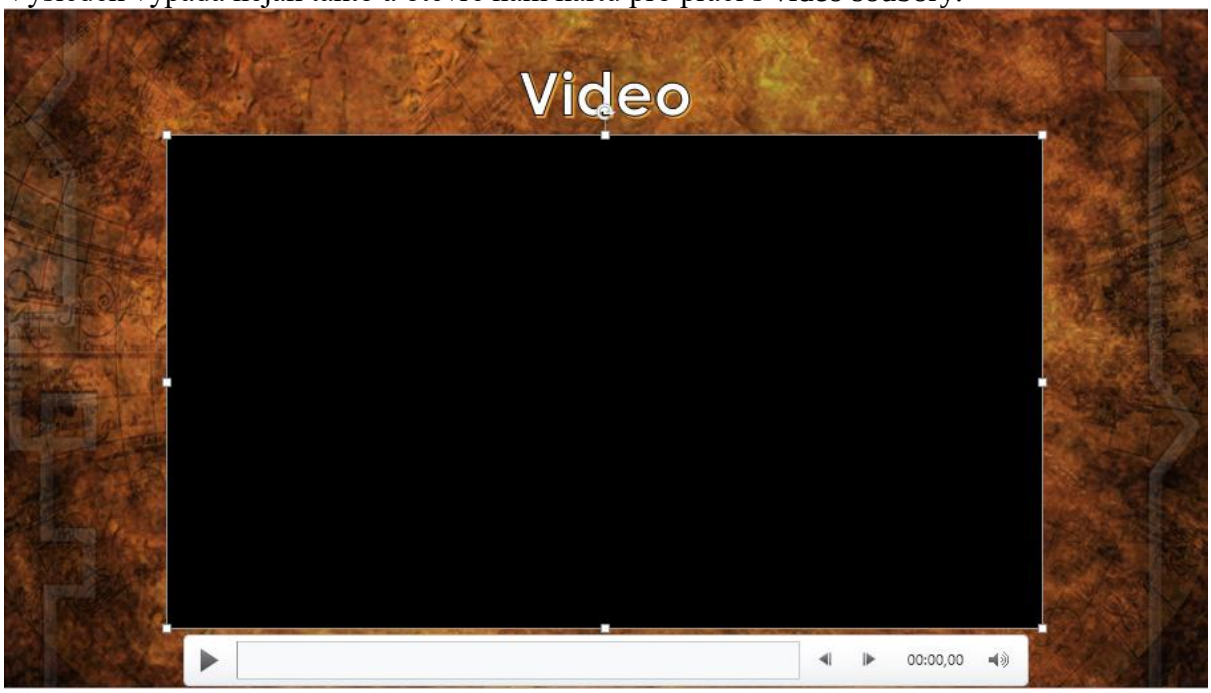
Jen pokud využíváme tlačítka na snímku, tak je nutné si uvědomit, že toto tlačítko je pro online režim, tudíž je nutné pomocí nabídky přejít do režimu souborů.



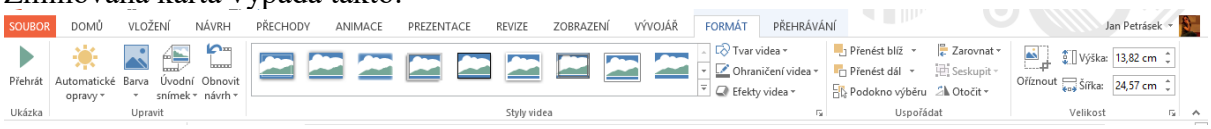
Volbou práce offline se dostaneme do průzkumníku, kde vybereme video soubor.

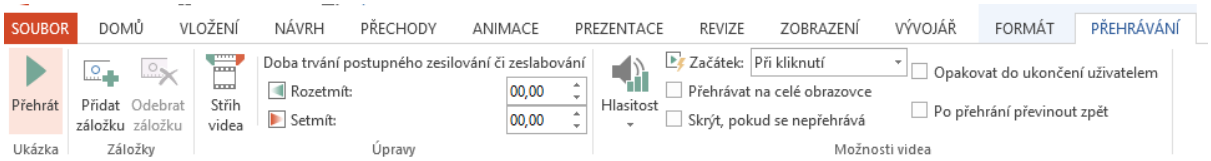


Výsledek vypadá nějak takto a otevře nám kartu pro práci s video soubory.

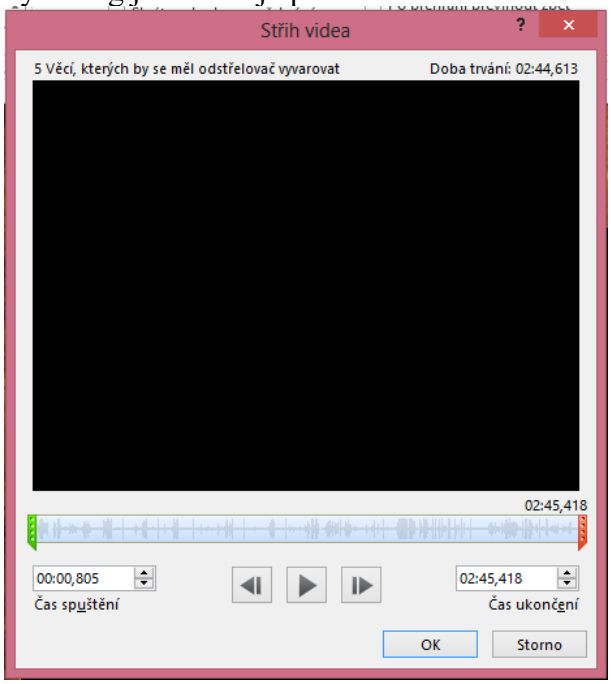


Zmiňovaná karta vypadá takto:

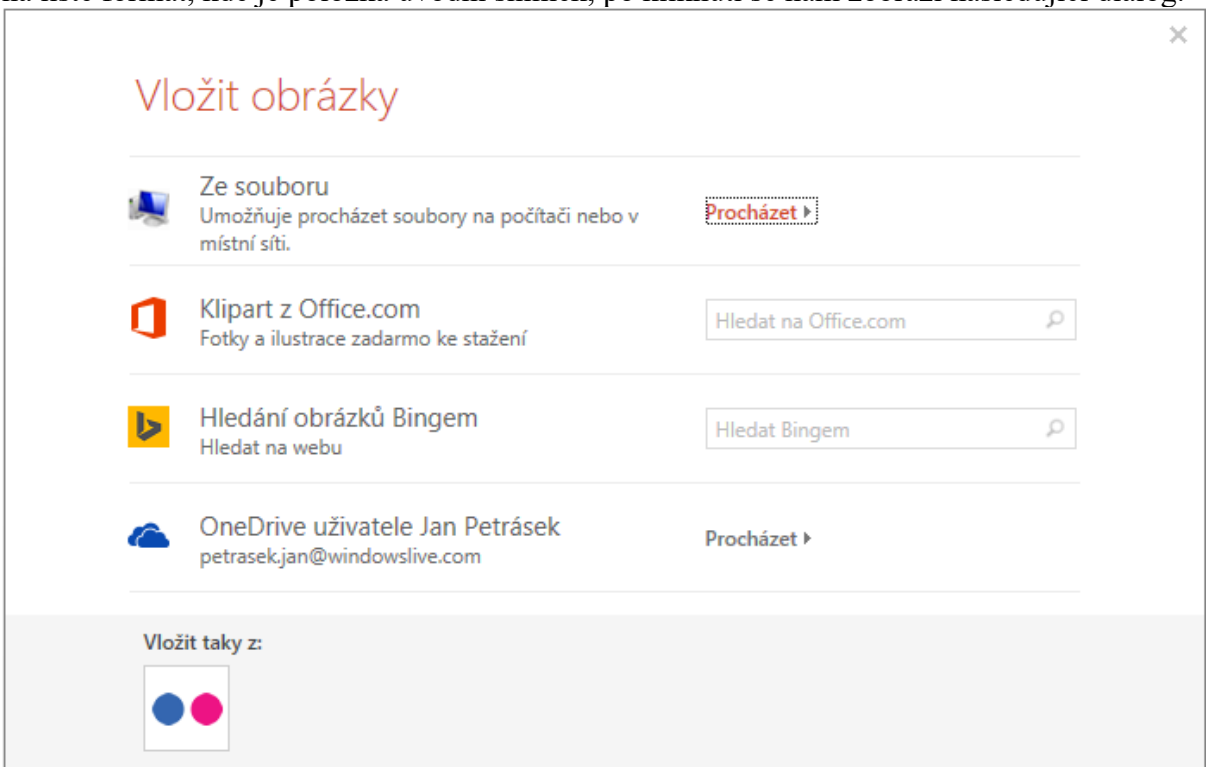




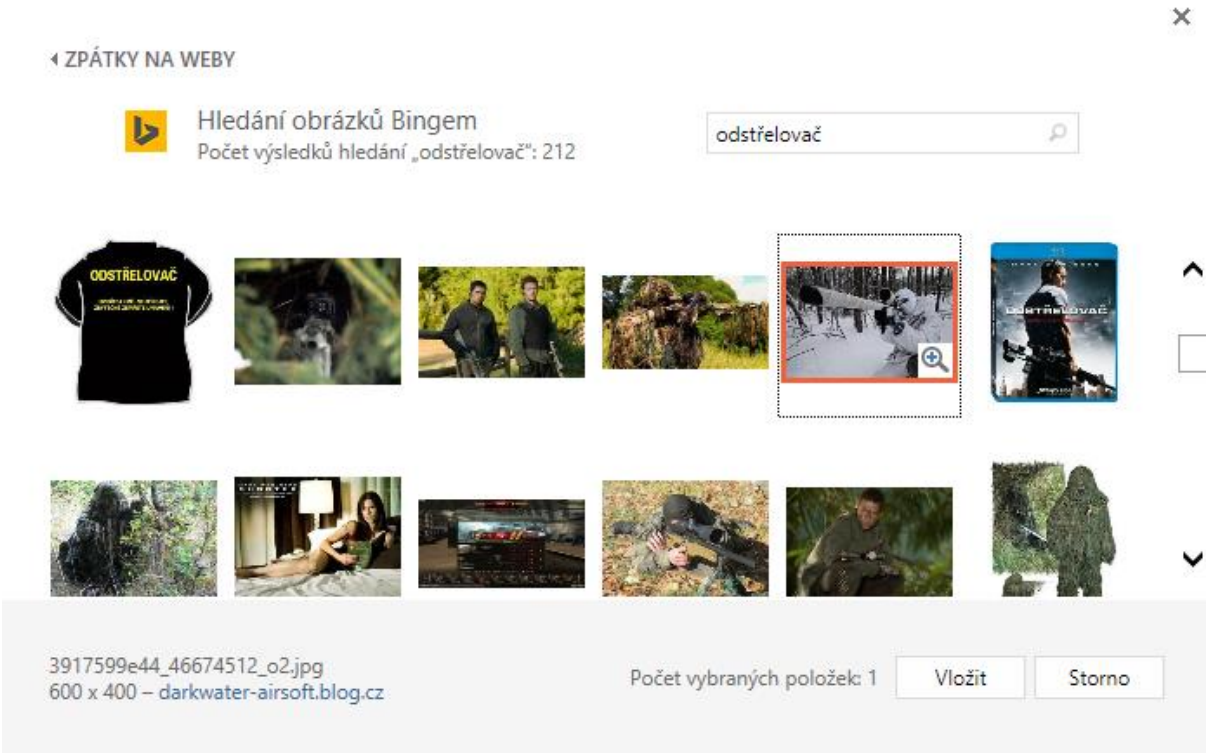
Zásadní nastavení se provádí na záložce přehrávání, kde řídíme jak zvuk, tak spouštění videa. Je zde také funkce střihu. Střih umožňuje ustříhnout začátek či konec videa, dle potřeby. Vyvolaný dialog je toho nejlepším důkazem.



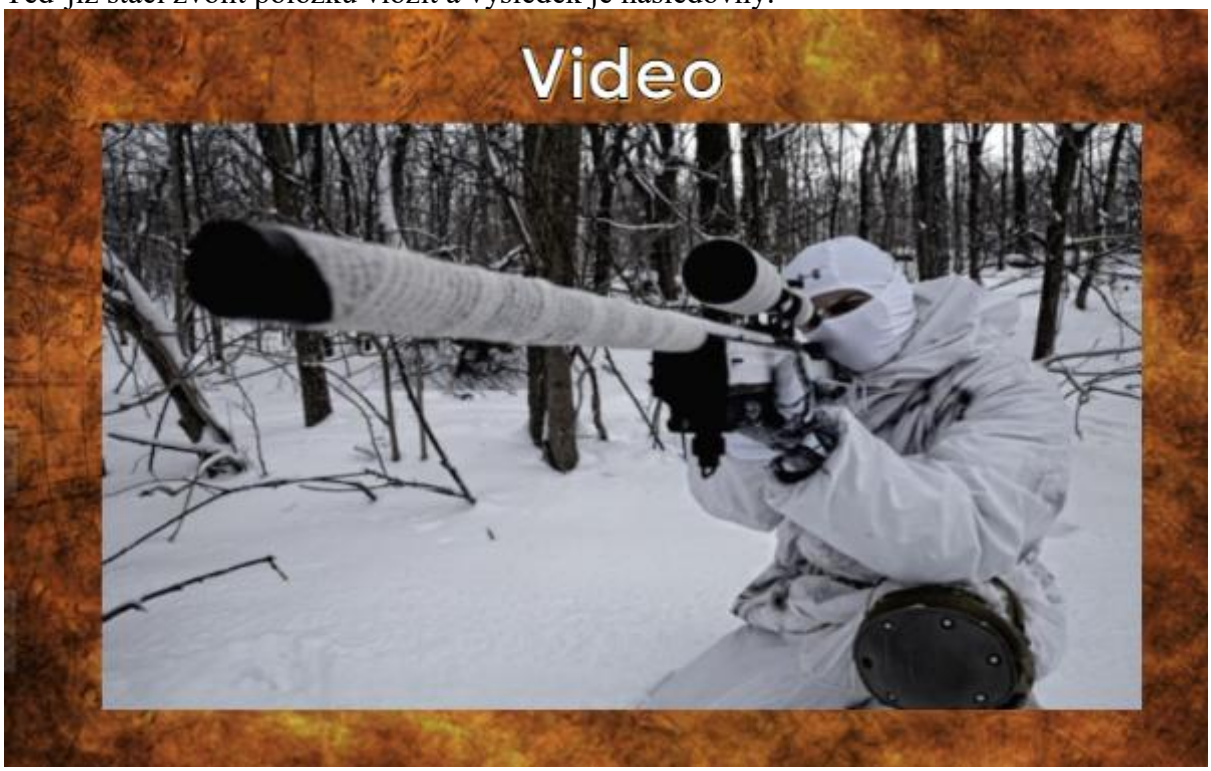
Protože není zrovna fajn, aby video začínalo černou obrazovkou, když se zobrazí při promítání, tak je dobré dohrát úvodní obrázek, pokud jej nenabídne video samo. To provedeme na liště formát, kde je položka úvodní snímek, po kliknutí se nám zobrazí následující dialog:



Kde si vybereme soubor v počítači, Claudu, nebo jej vyhledáme na webu. Pokud vyhledáváme pomocí služby bing, může vidět následující:

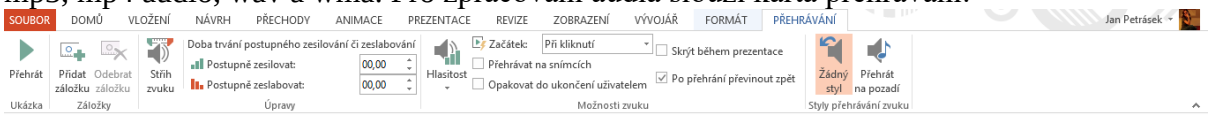


Teď již stačí zvolit položku vložit a výsledek je následovný:



4.2.6. Audio

Pokud chceme například umístit na snímek nahrávku s provázejícím výkladem, aby byla prezentace použitelná online, tak se mám hodí karta vložit, kde vedle videa je položka zvuk. Zde si můžeme vybrat, zda potřebujeme zvuk nahrát, nebo jej máme, či je dostupný online. Pro soubory uložené v počítači jsou podporované formáty: ADTS Audio, aiff, au, midi, mp3, mp4 audio, wav a wma. Pro zpracování audia slouží karta přehrávání.



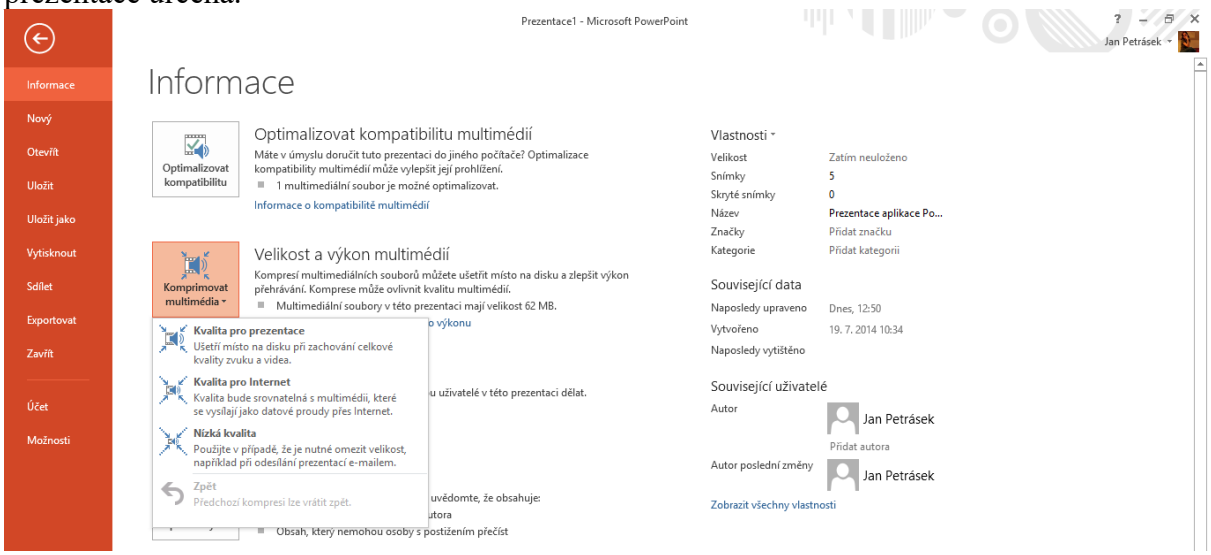
Obdobně jako u videa zde můžeme nastavit parametry přehrávání. Ikonu přehrávače upravujeme na kartě formát jako obyčejný obrázek.

Pokud nemáme zaškrtnuté automatické přehrávání při přechodu na snímek, tak se nám po najetí na ikonu přehrávače vlastní přehrávač, kterým je přehrávání zvuku řízeno.

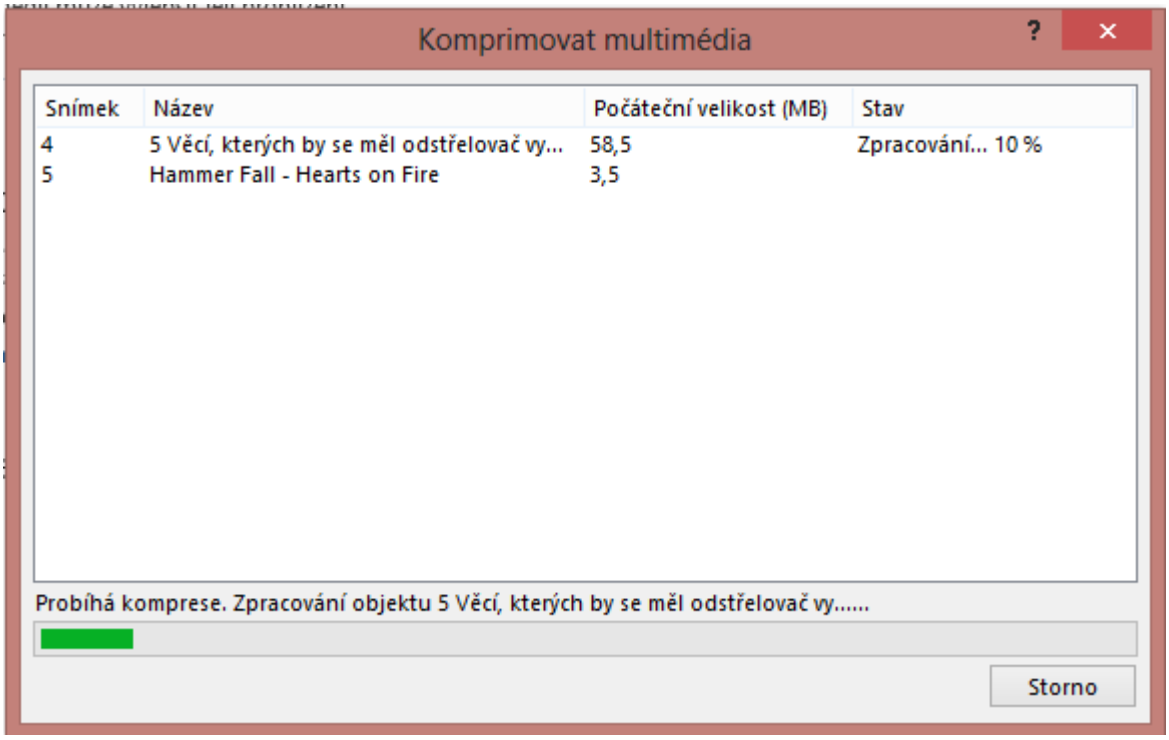


4.2.7. Komprese multimédia

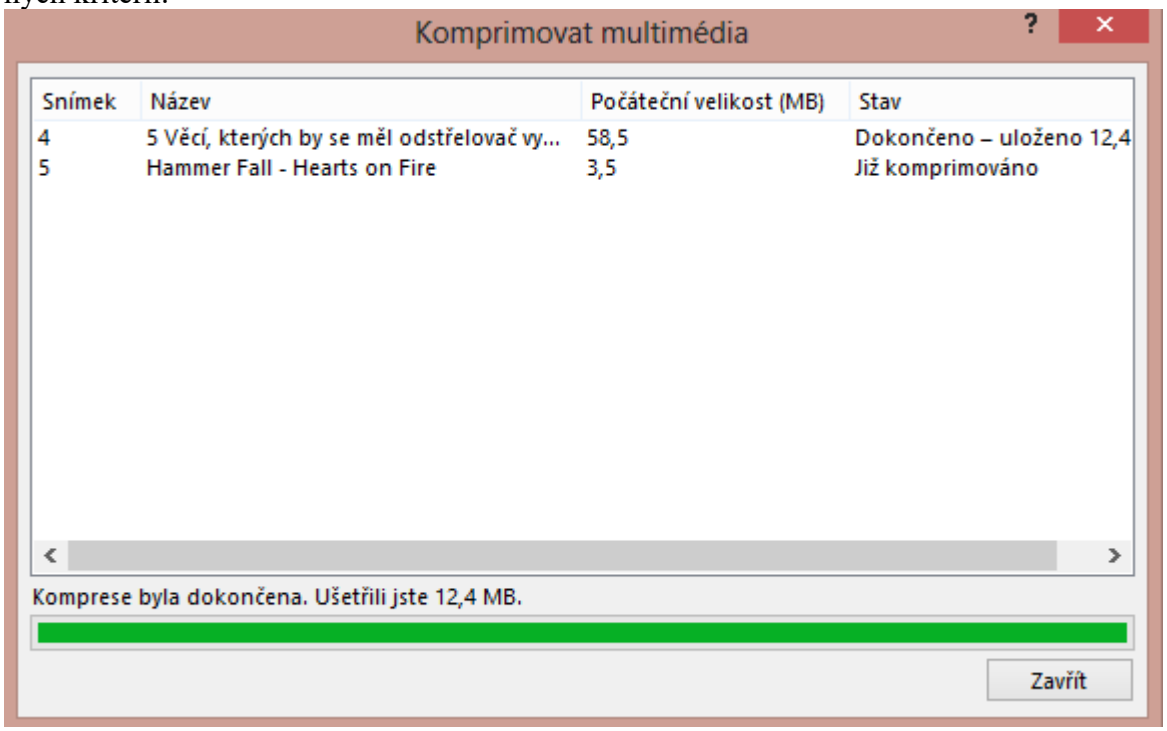
Aby naše prezentace, která obsahuje video a audio soubory byla nějak rozumně publikovatelná, tak je potřeba, aby jako celek měla nějakou rozumnou velikost. Jednou z možností, jak zmenšit prezentaci je komprimovat multimédia. Pro tento úkon se nám hodí tlačítko soubor, které nám umožní hned v položce informace zvolit kvalitu multimédií, dle toho, k čemu je prezentace určená.



Pokud prezentujeme z flash disku nebo svého počítače, tak volíme první variantu, pokud audio soubory plní možnost prezentace online, tak zvolíme kvalitu pro internet.



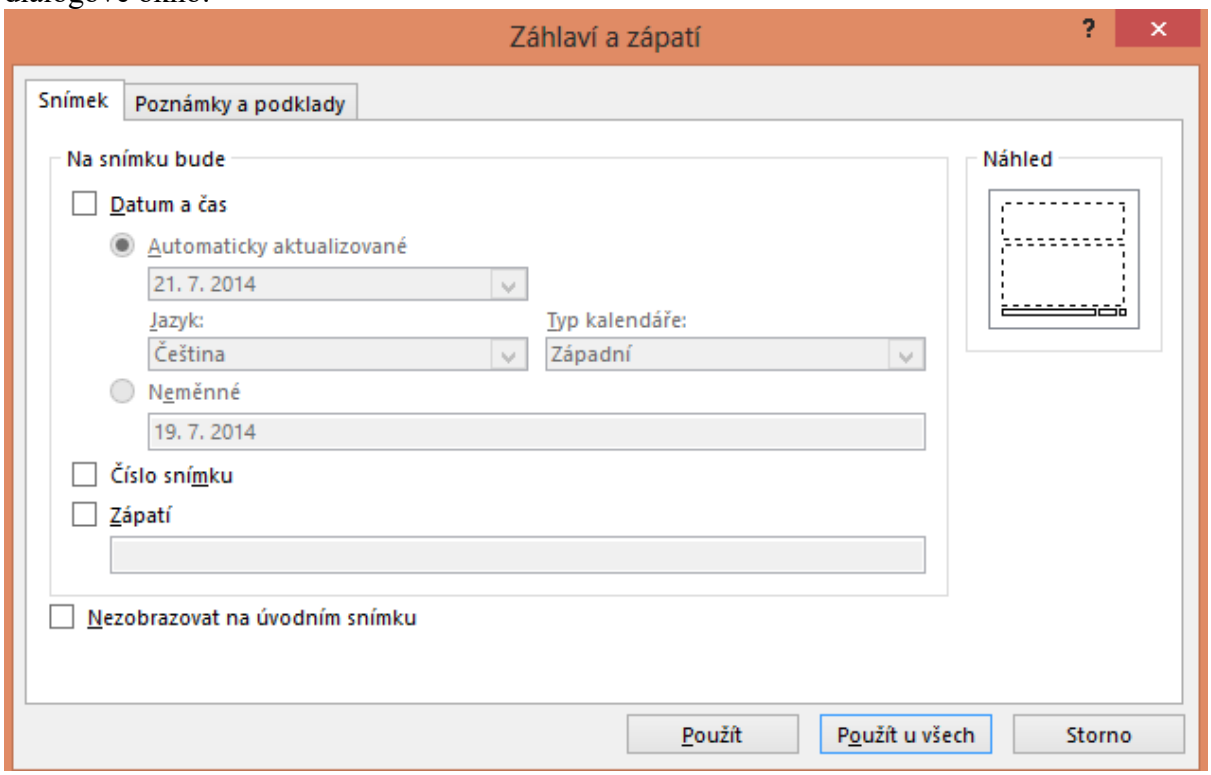
Počkáme na dokončení tohoto dialogového okna a koncová velikost bude snížena dle nastavených kritérií.



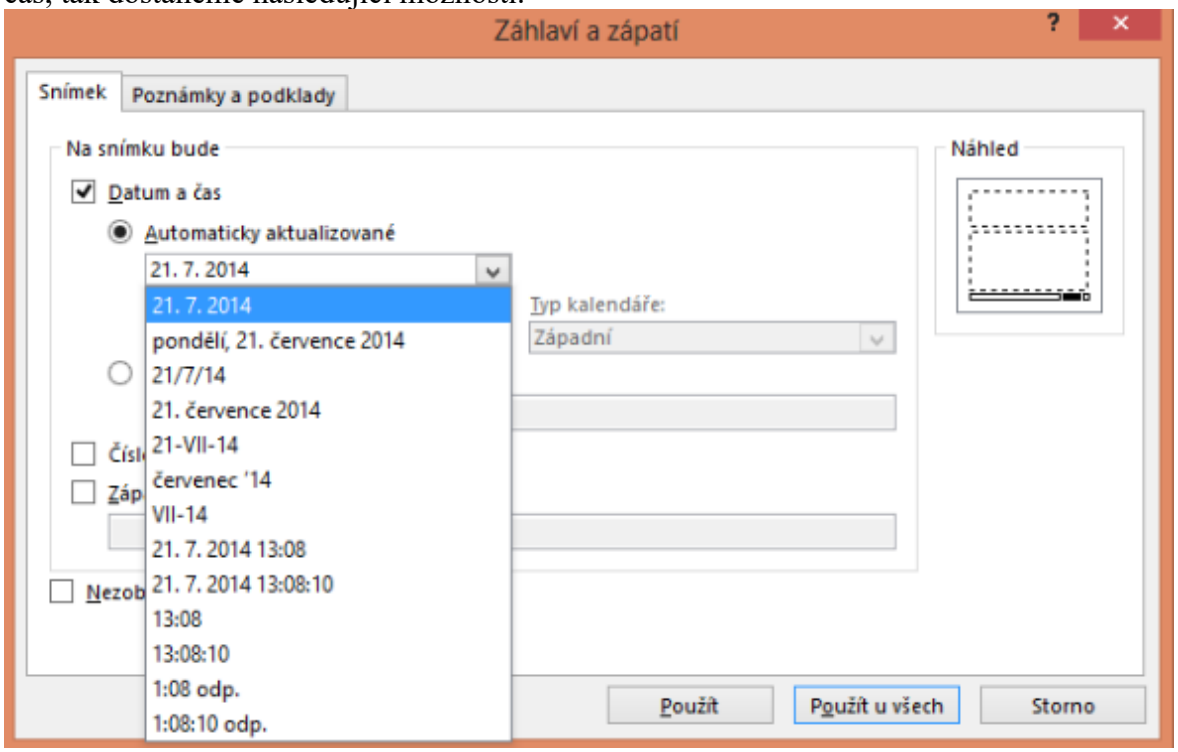
Klepneme na zavřít a můžeme prezentaci uložit či pokračovat v práci.

4.2.8. Čas, datum, číslo snímku, zápatí

Dost často je u přednášek vidět v zápatí aktuální čas a datum, či datum vytvoření. Přednášející mají raději ten aktuální, protože jim tím počítač pomáhá orientovat se v časovém prostoru, který je pro přednášku vyhrazen. Vytvořit si takovéhoto pomocníka není nikterak těžké, opět nám pomůže karta vložení, kde je přímo položka datum a čas, která vyvolá následující dialogové okno:

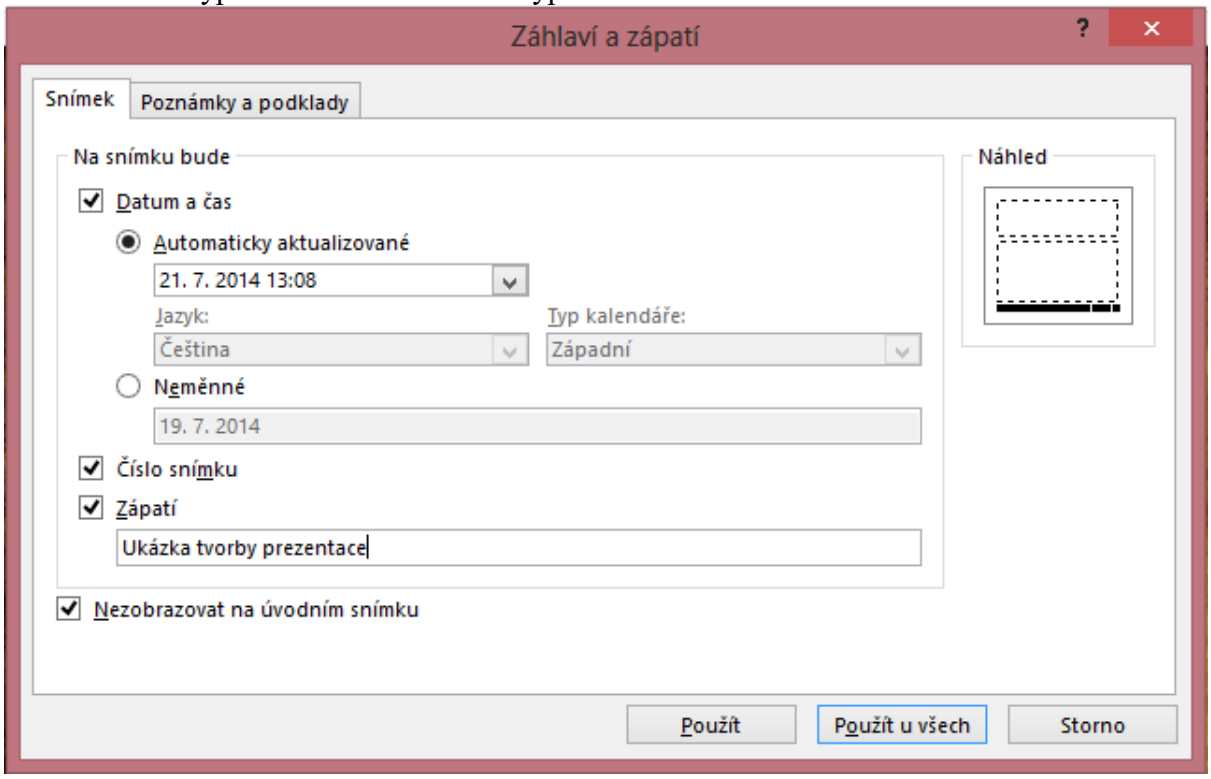


Nyní si můžeme výběrem zvolit, co chceme do zápatí umístit. Když zvolíme položku datum a čas, tak dostaneme následující možnosti.



Tím naše možnosti nekončí, můžeme si vymýšlet dále, například číslo snímku, což ulehčí studentům orientaci v prezentaci. Také můžeme název prezentace zkopírovat do zápatí, nebo jej v tom zápatí mírně rozvést. Také pro nějaké to designové naladění posluchačů můžeme vynechat úvodní snímek, což bývá dobrým zvykem.

Takže vyplněné nastavení může vypadat takto:

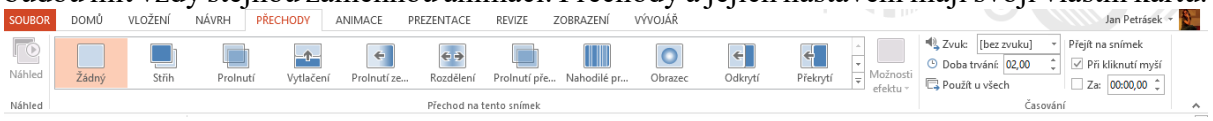


Nyní stačí zvolit **požít u všech**, jak je vidět, tak PowerPoint nás k tomu i přímo vybízí a reagoval by touto možností i na stisk klávesy enter. Výsledek může vypadat třeba následovně:

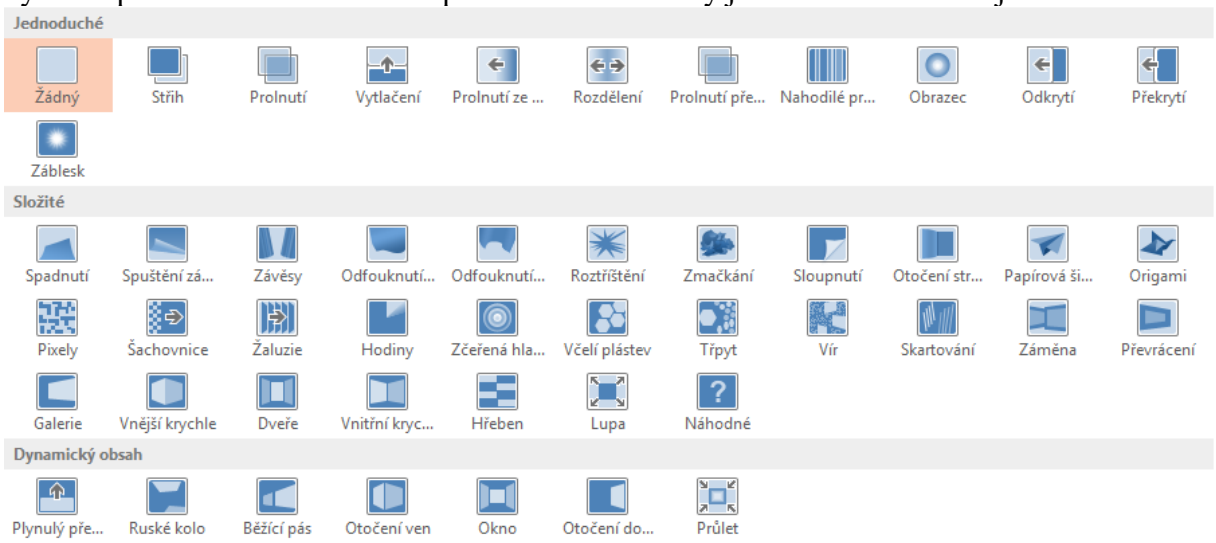


4.2.9. Přechody

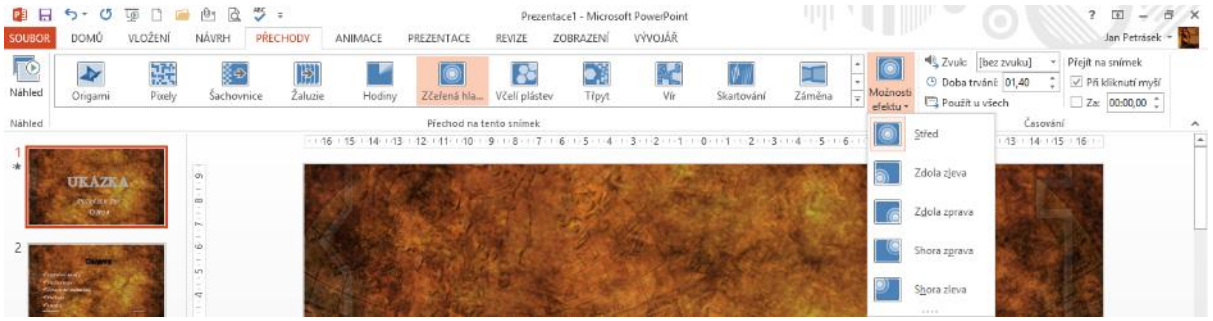
Aby ze záměny snímků nebolely oči, nebo nedocházelo k tomu, že záměnu posloucha-jící ani nezaregistruje, tak nenecháváme ostré přechody (prostá záměna za zlomek sekundy), ale mezi jednotlivé snímky vkládáme animaci, která se postará o jejich záměnu. Krom udržení pozornosti a šetření zraku se dají přechody užít k rozdělení prezentace do menších bloků, které budou mít vždy stejnou záměnnou animaci. Přechody a jejich nastavení mají svojí vlastní kartu.



Je potřeba dávat pozor a volit jednoduché přechody bez nějaké „průhlednosti“ pokud prezentaci chystáme s vědomím, že bude užívána na staré verzi Office, nebo ji bude přehrávat v online sadě Office, která je součástí uložiště OneDrive. Na této kartě můžeme také zvolit podměty k přepnutí snímku. Pokud se rozhodneme využít přepnutí snímku po určitém čase, tak nechme přepínání i při kliknutí myší a nechme trošku delšní časový úsek, protože časování není při přenosu na jinou verzi programu vždy přesné. S efektky to není dobré příliš přehánět, naopak by nám měly pomoci vytvořit bloky, nebo by jich mělo být co nejméně, aby prezentace byla alespoň trošku seriozní. Kompletní základní efekty jsou videt na následujícím snímku.



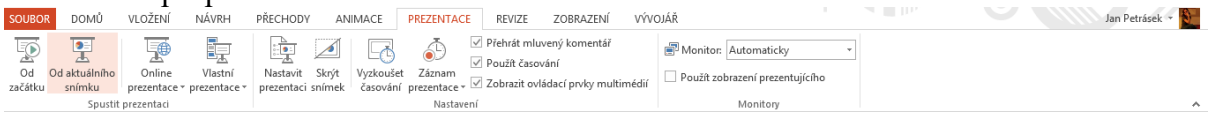
Pokud si zvolíme přechod, tak je možné jej i mírně upravit, příklad je vidět na dalším obrázku.



Rychlost efektu je možné také měnit pomocí položky doba trvání. K přechodu je možné přidat i zvuk, ale u oficiálních prezentací se to nedělá, to si můžeme dovolit u nějakého přání pro známé.

4.2.10. Konfigurace prezentace

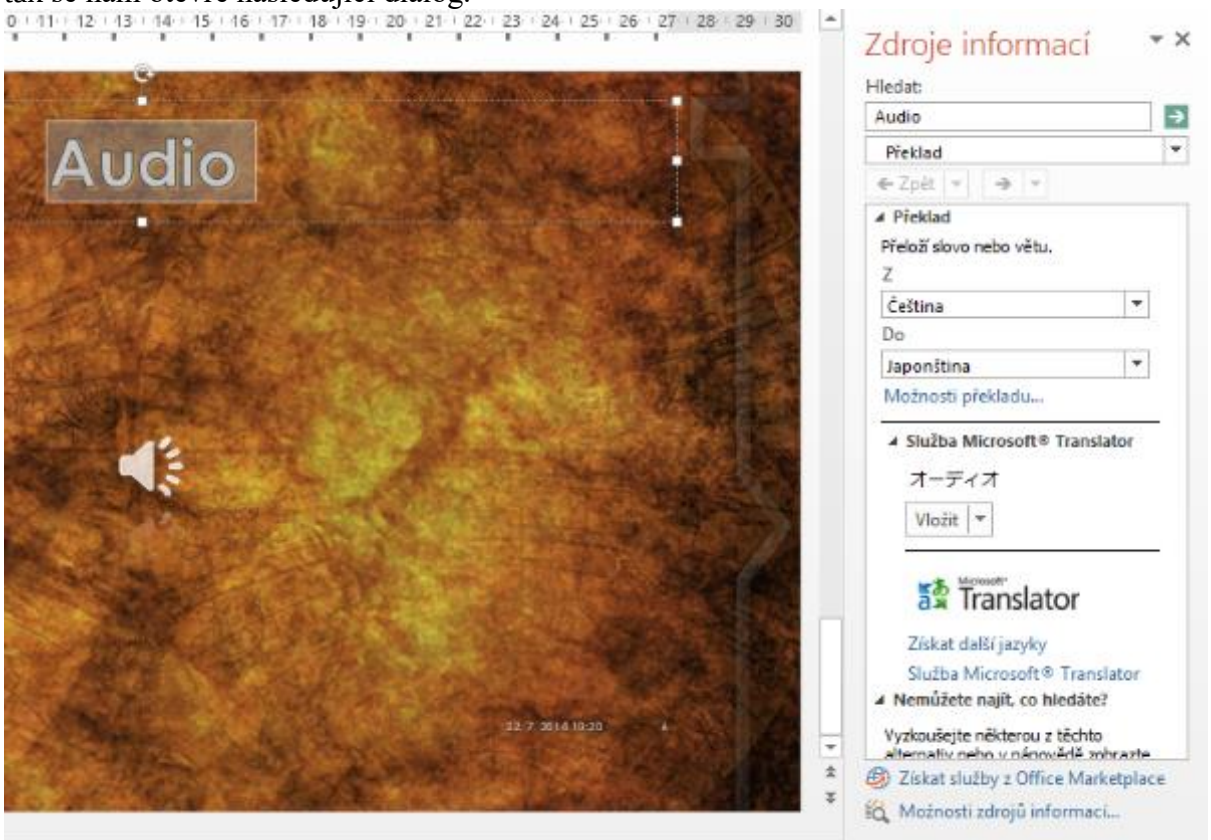
Ke konfiguraci přehrávacího okna slouží karta prezentace. Tato karta nám umožňuje nejen volitelně spustit prezentaci, vytvořit odkaz, pomocí něhož je možné sledovat prezentaci ve webovém prohlížeči, což umožňuje zobrazení na studentských PC bez potřeby speciálního SW na sdílení obrazu či vzdálené řízení PC. Tato akta nám hlavně umožňuje testovat časování, pořídit a vyhodnotit záznam z prezentace, a nakonfigurovat nástroje, které nám poskytuje PowerPoint při přehrávání a to včetně monitoru.



4.2.11. Strojový překlad prezentace

Pokud bude prezentace přehrávána na počítači, který bude jak při tvorbě, tak při prezentování připojen k internetu, tak PowerPoint nabízí i možnost strojového překladu textu v prezentacích celkem asi do 60 jazyků, což je dobré hlavně pro učitele, protože pokud přijde na českou přednášku někdo, kdo nemluví česky, ale třeba jen japonsky, tak může mít k dispozici prezentaci s dalšími přednáškami, ze které může vše sledovat a učitel nemusí shánět někoho, kdo to pro takového člověka přeloží.

Překladač bing translator je integrovaný na kartě revize, nebo jej můžeme vyvolat po označení textu pomocí pravého tlačítka a nabídky přeložit. Pokud takto aktivujeme překladač, tak se nám otevře následující dialog.



Zvolíme nastavení jazyků a pokračujeme kliknutím na vložit. Dojde tak k záměně označeného textu jeho překladem.



Pro co nejvyšší přesnost překladu je potřeba heslovitost a vyvarování se složitých větných struktur, opravdu je potřeba dodržet jednoduchost a stručnost textu.

4.2.12. Ukládání

Již několikrát jsem se v textu zmínil o tom, že si práci máme průběžně ukládat. Všechny tyto aplikace mají sice automatické ukládání dat po nějakém časovém úseku, ale ze zkušenosti mojí drahé polovičky mohu říct, že toto ukládání je silně nedostatečné, zvláště pokud práci necháte ukládat jen do obnovovacího souboru a dáte hibernovat počítač, který se špatně probere, nebo jej stihnete s touto chybou dát hibernovat před prvním automatickým uložením. Pravda, tyto časy se dají zkrátit, ale nevím jak na 32 bitových sadách Office, ale na 64 bitových pokud se prezentace blíží 150 snímkům a hojně užívá multimédia, takže celková velikost prezentace je kolem 200 MB, již je ukládání schopné vám zamezit v práci, protože trvá dlouho a je možné, že pokud budeme pracovat, tak se aplikace na chvilku zakousne, proto není dobré tento čas příliš zkracovat, protože bychom se z toho zbláznili.

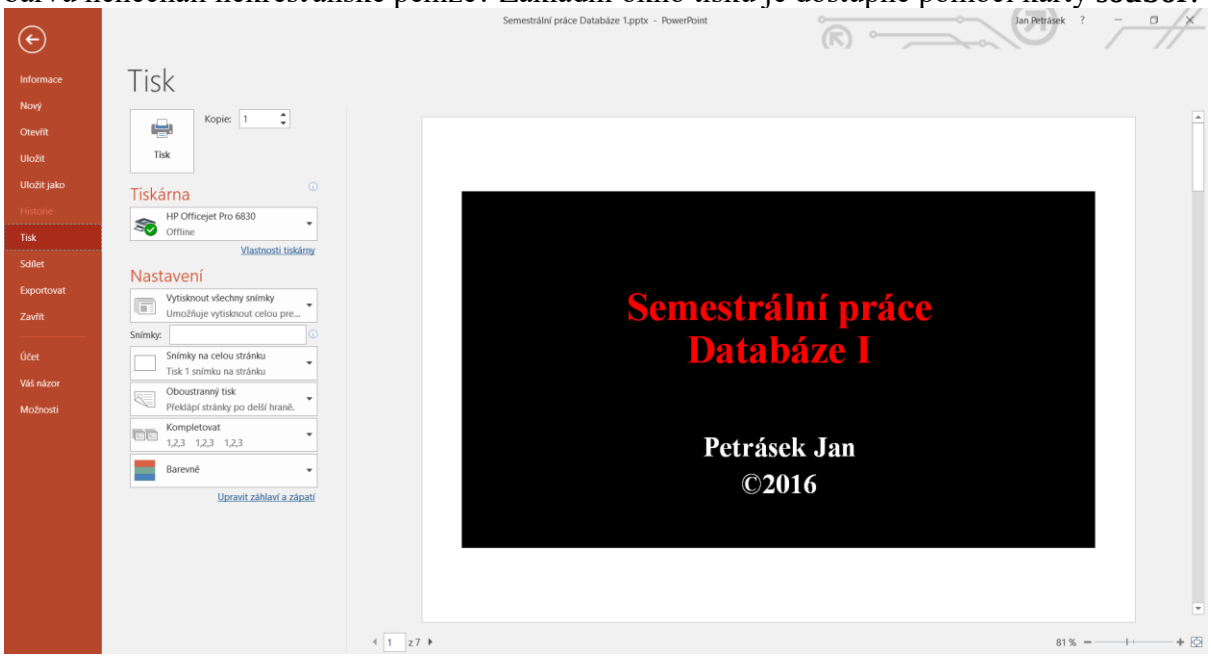
Práci je potřeba vždy uložit ručně a průběžně na to myslet a ukládat, ukládání není nikdy dost a PowerPoint umí sledovat i předchozí veze dokumentu s každým naším uložením, které je ruční, takže v případě nějakého problému, ze kterého se neumíme vysekat ani s pomocí možnosti zpět, či přepracováním, dojde k chybě přímo v dokumentu nebo šabloně ze které je odvozen, můžeme tento problém často řešit právě pomocí předchozí verze dokumentu.

Ukládáme obdobně jako ve Wordu pomocí karty soubor, kde máme možnost uložit a uložit jako. Je tam i možnost exportovat, kde PowerPoint nabízí formáty pdf, xps, videosoubor, balíček CD (vypálí všechny užité soubory spolu s prezentací na CD a umožní tak zachování cest), vytvořit podklady (také si ukážeme), změnit typ souboru (nahradí stávající uložený soubor souborem jiného typu). PowerPoint v základu ukládá do nového formátu, tedy pptx, ale pojďme si představit i další možnosti které máme, protože mohou příjemně překvapit.

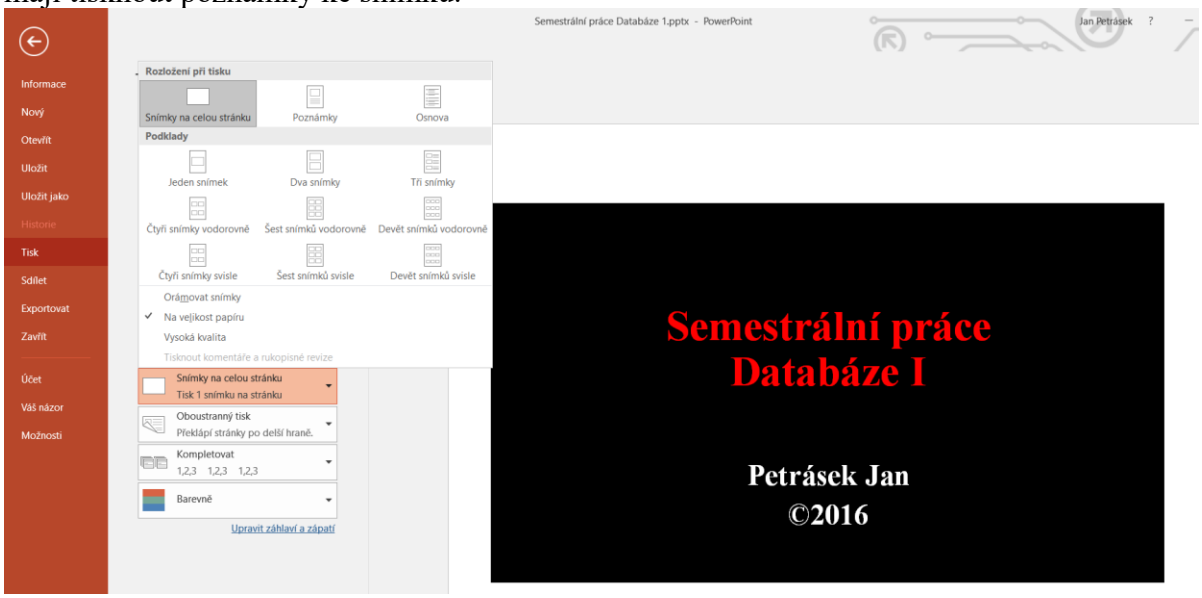
- Ppt – soubor pro verze 97 až 2003
- Pdf / xps – převede na klasický dokument
- Potx – šablona PowerPoint 2013
- Pot – šablona PowerPoint 97 – 2003
- Ppsx – přejde rovnou do spuštění prezentace bez otevření okna PowerPointu
- Pps - přejde rovnou do spuštění prezentace bez otevření okna PowerPointu 2003
- Mp4 – videosoubor
- Wmv – videosoubor
- Složka obrázků (fif, jpg, ng, tif, bmp)
- Odp – formát pro Open Office
- RTF – osnova pro zpracování ve Wordu, Adobe readeru apod.

4.2.13 Tisk a podklady

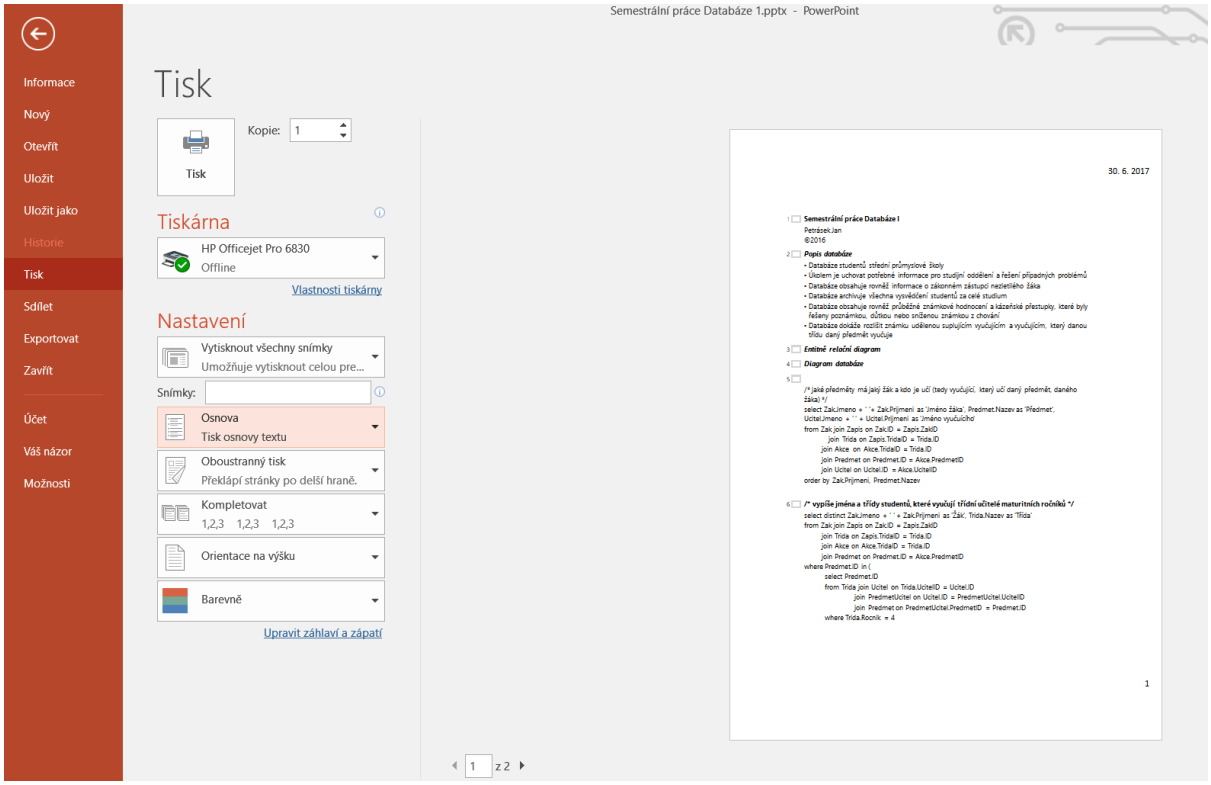
Asi nikoho nepřekvapí možnost tisku z PowerPointu, ale jak to udělat, abychom za barvu nenechali nekřesťanské peníze? Základní okno tisku je dostupné pomocí karty **soubor**.



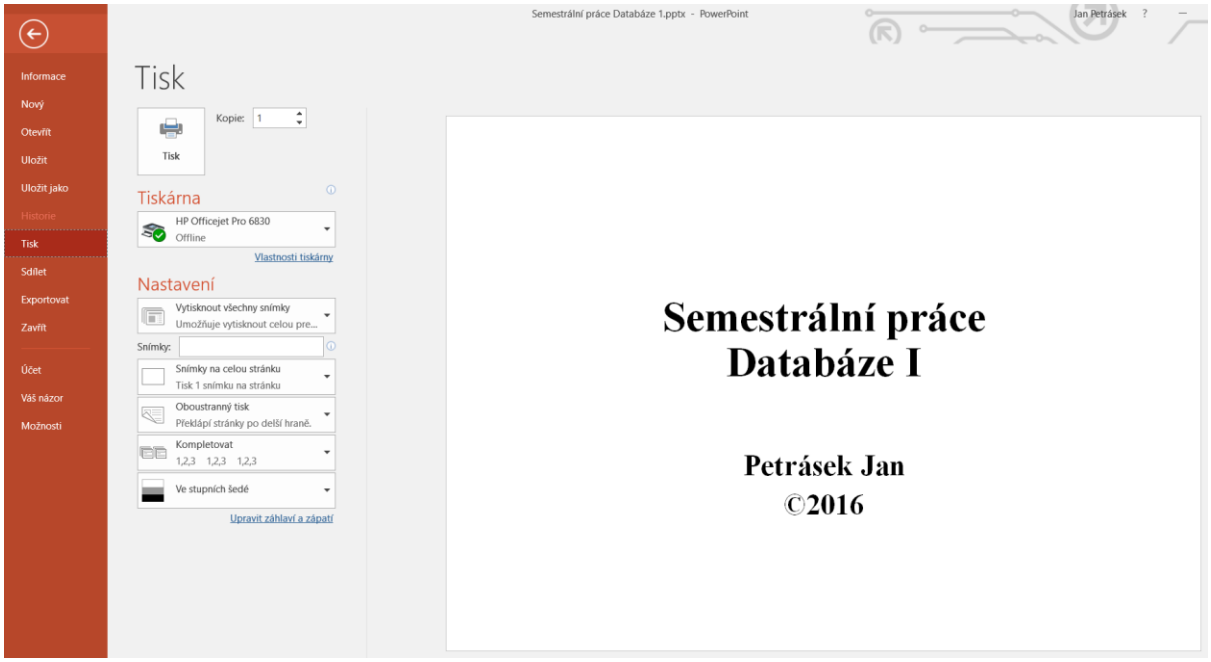
Nejprve si nastavíme, jak se mají snímky tisknout, tedy kolik jich má být na straně a zda se mají tisknout poznámky ke snímku.



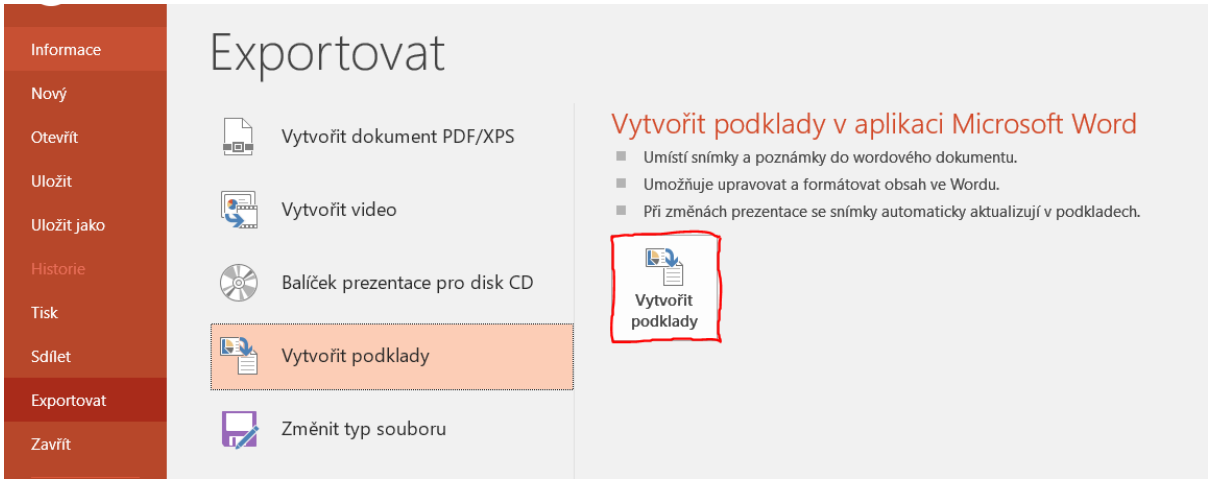
Velmi šikovnou a úspornou možností tisku je Osnova. Tato mžnost ale nazachovává formátování prezentace, takže pozor, pokud prezentace obsahuje šipky a další pomocná vodítka.



Dalším typem, který pomůže ušetřit barvu je volba černobílého tisku, která nám pomůže tím, že pozadí nahradí bílou a písmo je černé, pokud jsou v prezentaci obrázky či grafy, tak doporučuji užít volbu stupně šedé, která má obdobný účinek, ale umoožní čitelnost grafům a obrázkům.



Pokud si potřebujeme samoostaně vytvořit podklady pro prezentování, pak nám je pomůže možost Exportovat odeslat do MS Word a dále si je uzpůsobit k obrazu našemu.



4.3. Excel

Microsoft Excel je tabulkový procesor od firmy Microsoft pro operační systém Microsoft Windows a počítače Macintosh. Už od verze 5 z roku 1993 má dominantní postavení na trhu. Dnes se prodává hlavně jako součást kancelářského balíku Microsoft Office. Jeho hlavním konkurentem je Calc, který je součástí balíku kancelářských aplikací OpenOffice.org.

Microsoft vstoupil na trh tabulkových kalkulátorů v roce 1982 s programem nazvaným Multiplan, který byl ve své době velmi rozšířený na počítačích s operačním systémem CP/M. V prostředí počítačů řízených systémem MS-DOS prohrál v souboji s Lotusem 1-2-3. To vyvolalo rozhodnutí zahájit vývoj nového typu tabulkového kalkulátoru nazvaného Excel s cílem dosáhnout řešení, které řečeno slovy Douga Klundera „bude umět vše, co umí Lotus 1-2-3 a bude to umět lépe“. První verze pro Macintosh vyšla v roce 1985. MS Excel byl první program, jehož GUI využívalo rozbalovacích menu ovládaných klikáním myší. Práce s ním byla mnohem pohodlnější než v kterémkoli DOS-ovém programu. Rovněž se dalo psát ve 256 fontech. Mnoho lidí si kvůli němu koupilo počítač Macintosh. Verze Excelu pro MS-DOS vyšla až 31. října 1992.

A později v listopadu 1987, kdy byl uveden jako jedna z prvních aplikací pro MS Windows, byl jedním z hlavních programů, které táhly uživatele k používání Windows. Tato verze byla z důvodu sjednocení s verzí pro Mac označena jako 2.0 – neexistuje tedy žádný Excel 1.0 pro Windows. Od roku 1988 překonával v prodejnosti hlavního konkurenta Lotus 1-2-3. V roce 1989 Windows 3.0 dosáhlo širokého rozšíření a přesto až do léta 1992 se neobjevil jediný konkurenční tabulkový procesor pro MS Windows.

V roce 1993 Microsoft přidal podporu pro Visual Basic for Applications (VBA), čímž sice umožnil široké využití maker, ale zároveň pomohl snad ještě většímu rozšíření makrovirů. V tomto roce také vyšla první verze Microsoft Office a Microsoft Word a Microsoft PowerPoint byly upraveny tak, aby jejich GUI odpovídalo právě Excelu.

V počátcích vývoje musel být Excel označován jako „Microsoft Excel“ v důsledku sporů o ochranou známku, později se od této praxe upustilo. Aktuální verze Excelu pro prostředí Windows nese označení Microsoft Office Excel 2013. Aktuální verze Excelu pro platformu Mac OS je označena jako Microsoft Excel 2011.

Excel je tak těžký a obsáhlý program, že mu nerozumí úplně ani lidé, kteří jej rozvíjejí, nicméně základ aplikace se již po leta nemění. Mezi studenty není Excel nijak oblíbený na rozdíl třeba od PowerPointu, je pro ně náročný a výstupy z něj jsou na základní škole a gymnáziu celkem nadhodnoceny potřebám. Tento program nejvíce ocení studenti vysokých škol a pak studenti technických, ekonomických a přírodovědných oborů. Asi nikoho ze čtenářů nepotěším tím, že tomuto programu budeme věnovat nemalé prostředky, nejprve si představíme

možnosti a základních operací ve Verzi 2013, o které se tady celou dobu bavíme a poté se podíváme podrobně na spoustu jeho funkcí, nicméně k tomu využijeme starších materiálů vytvořených pro Verzi 2010. I já tomuto programu nemohu 100 % rozumět a umět všechno, proto si tu představíme 2013 v 10 krocích a pak se pustíme do starších materiálů, protože jsem neměl čas je přetvořit pro novou verzi. Pokud se tedy budeme bavit o některém z úkonů, který si představíme v desateru, tak se řiďte desaterem, cesta uvedená v postupu bude funkční, ale nebude tolik efektivní. Editovat obrázky umí stejně jako Word a základní editace grafické podoby grafu je také podobná Wordu.

4.3.1. Desatero Excelu 2013

4.3.1.1. Spuštění Excelu, pojmenování, uložení

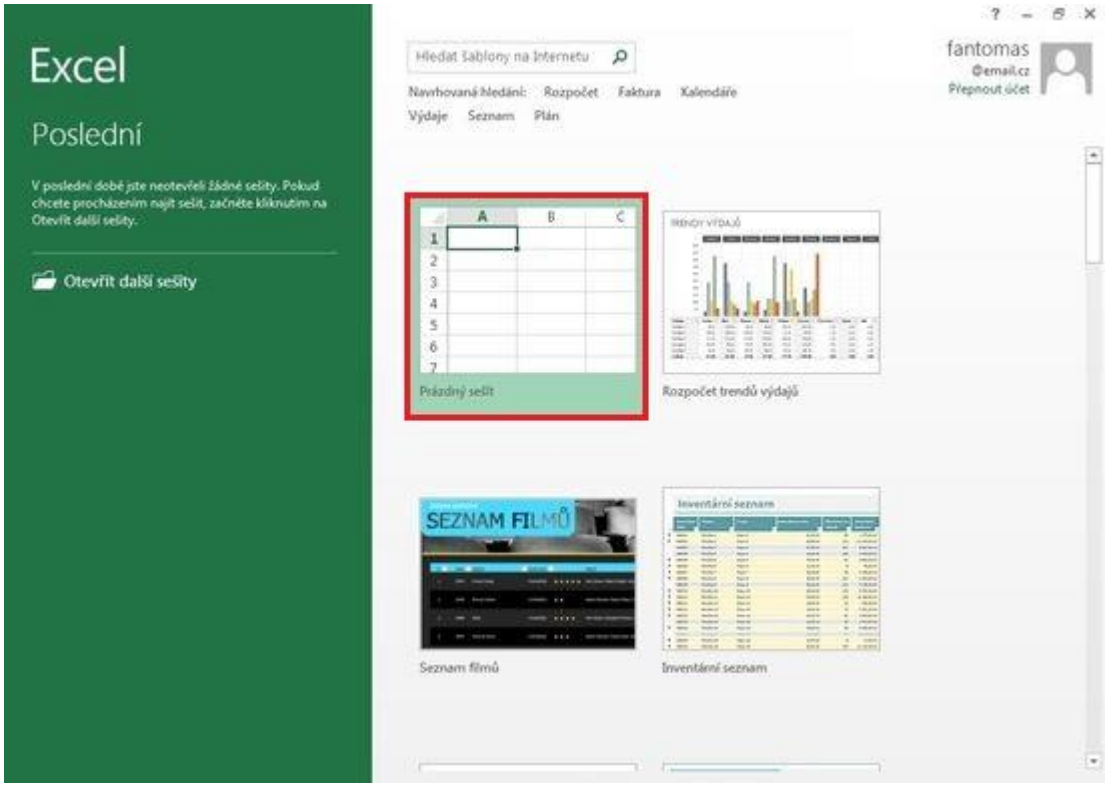
Nejprve spusťte Excel 2013, klikem na **ikonu na ploše**, nebo přes nabídku **Start – Všechny programy – Microsoft Office – Excel 2013** (pokud máte Windows 8, najděte příslušnou ikonu na ploše).



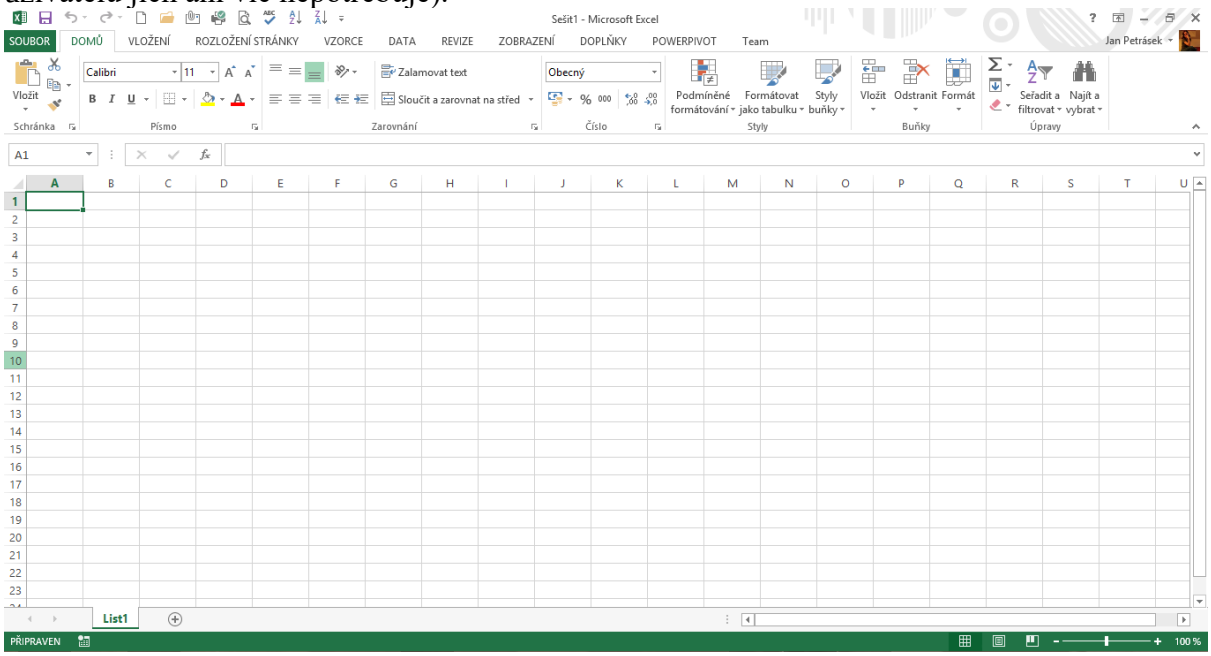
Spuštění Excelu je indikováno dialogovým oknem:



Novinkou od verze Excel 2013 je, že není zobrazeno okno aplikace, ale ***Úvodní okno Excelu***, zde si vyberete buď některý z naposled otevřených sešitu (jsou zobrazeny, pokud jste již s Excel pracovali), případně můžete otevřít *Prázdný sešit* nebo si vybrat některou z předpřipravených šablon. Vyberte si: ***Prázdný sešit***

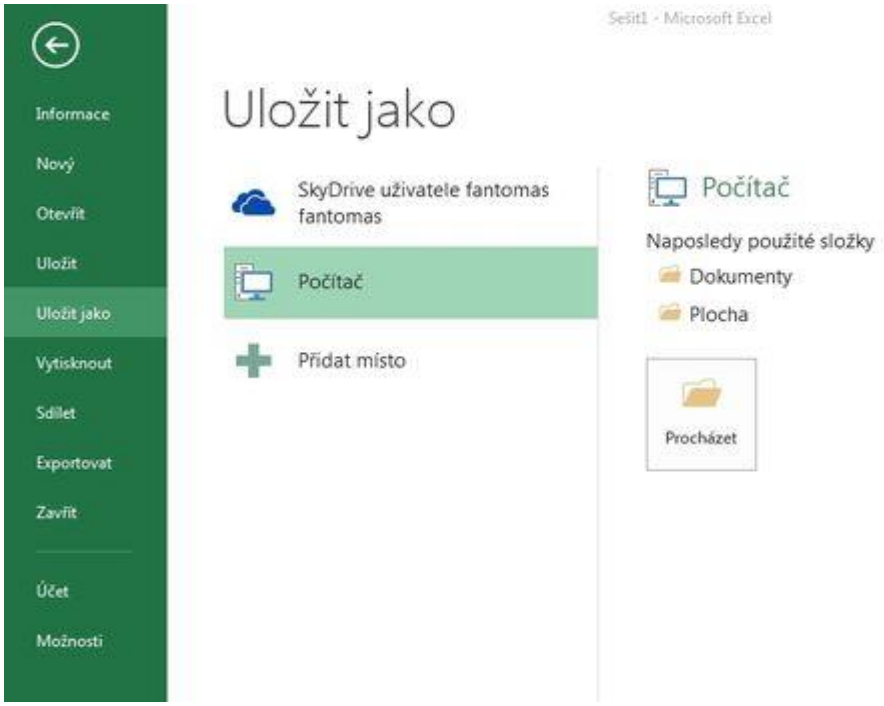


Teprve výběrem dojde k zobrazení aplikace Excelu s prázdným listem (pokud někdo nenastavil jinak). Oproti předchozím verzím je změna a zobrazuje se pouze jeden list (většina uživatelů jich ani víc nepotřebuje).

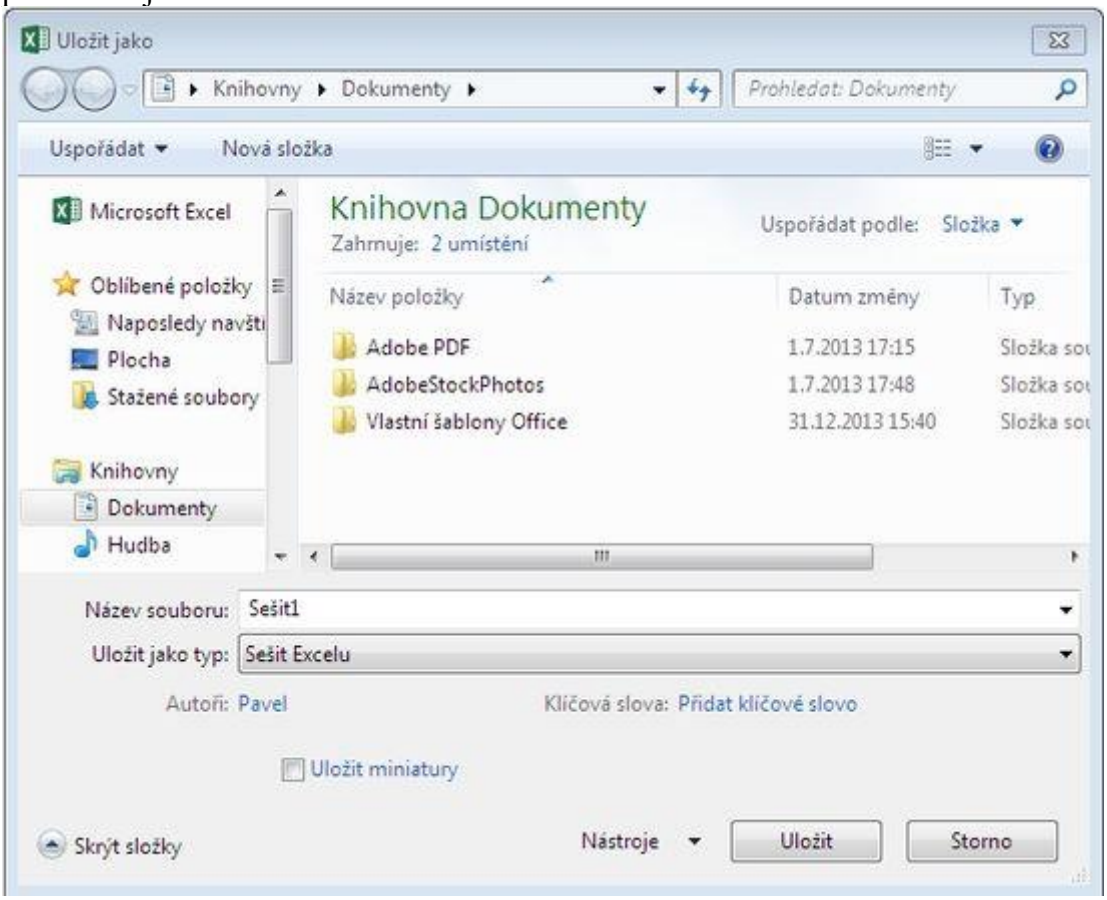


4.3.1.2. Uložení sešitu

Uložení sešitu se provedete – karta ***Soubor*** a z menu ***Uložit jako*** následně ***Počítač*** a klik na ikonu ***Procházet*** (Oproti předchozím verzím tlačí Microsoft do používaní cloudu a jejich SkyDrive).



V dialogovém okně Uložit jako vyberte místo kam chcete dokument uložit a pojmenujte jej například Moje-tabulka a uložte tlačítkem Uložit.



Pokud se Vám povedlo váš sešit pojmenovat a uložit, máte za sebou úspěšně první krok.

4.3.1.3. Vytvoření tabulky

Než začnete tvořit excelovskou tabulku trocha nezbytné teorie. Excel se skládá z listu (sice ve verzi 2013 máte k dispozici list jeden, ale není problém jich přidat více). List se skládá

z buněk. Představit si to můžete jako čtverečkovaný sešit, kdy listy jsou jednotlivé listy v sešitě. Jednotlivé čtverečky odpovídají buňkám. Do buněk se mohou vkládat čísla, vzorce, texty, (dokonce i minigrafy), poznámky, atd. Každá buňka má svou adresu (pozici například A1), takže umístění není neznámé. V Excelu jsou v základním nastavení sloupce označeny písmeny, řádky čísly. Takže první buňka má adresu A1, druhá buňka v první řádku B1. Buňka ve třetím sloupci a třetím řádku C3.

4.3.1.3.1. Příklad tabulky

Vytvořte si ukázkovou tabulku

Moje první tabulka s grafem		
Kdo zodpovídal dotazník?		
Pohlaví	osob	Procent
Muž	35	
žena	45	
Nezodpovězeno	5	
Celkem zodpovídalo osob		

Tabulka bude Excelu 2010 vypadat podobně:

	A	B	C
1	Moje první tabulka s grafem		
2	Kdo zodpovídal dotazník?		
3			
4	Pohlaví	osob	Procent
5	Muž	35	
6	žena	45	
7	Nezodpov	5	
8			
9	Celkem zodpovídalo osob		

Poznámka: Tabulka vypadá trochu nepřehledně, proto v další kapitole jí doplníme o základní formátování.

4.3.1.4. Formátování tabulky

- **Sloučení buněk**
- **Úprava fontu** – velikost barva
- **Ohraničení**

4.3.1.4.1. Sloučení buněk

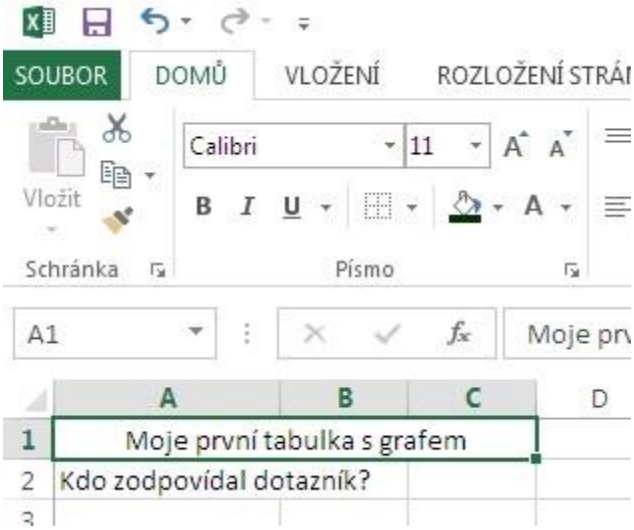
Označíte buňky A1 až C1. Na kartě ***Domů*** v sekci ***Zarovnání*** vyberete ikonu sloučit.



a máme hotovo



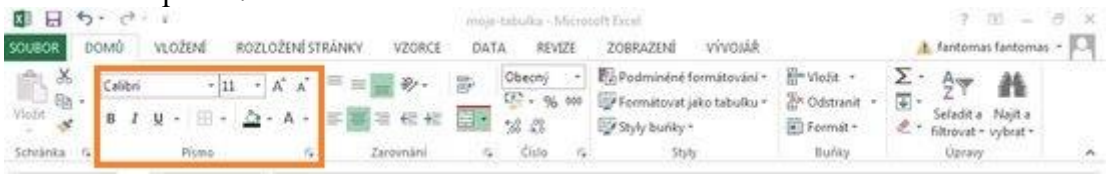
Výsledkem bude.



Podobně můžete provést i pro další buňky.

4.3.1.4.2. Úprava fontu

Protože už máte označeny (případně je znovu označte) buňky A1 až C1 provedete změnu fontu. Na kartě ***Domů*** v sekci ***Písmo*** Zvolíte velikost fontu a pak pomocí ikonu pozadí buňky a také barvu písma.



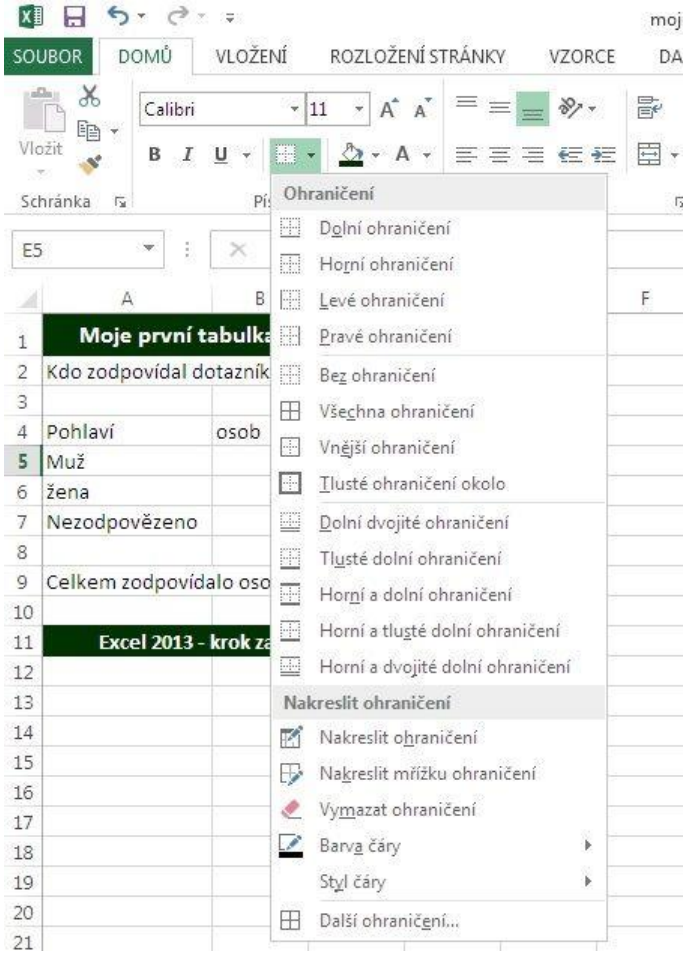
Máme hotovo

	A	B	C
1	**Moje první tabulka s grafem**		
2	Kdo zodpovídal dotazník?		
3			
4	Pohlaví	osob	Procent
5	Muž	35	

Podobně můžete upravit druh fontu, změnit jej na kurzívu nebo tučné písmo atd. i pro další buňky.

4.3.1.4.3. Ohraničení

Ještě ukážu, jak ohraničit tabulku a budete mít úplný základ za sebou. Označíte buňky A4 až C7 (prostě data která budou tvořit tabulku). Na kartě ***Domů*** v sekci ***Písmo*** vyberete ikonu ***Ohraničení***  z nabídnutého seznamu vyberte ***Všechna ohraničení***.



a máme hotovo. Tabulka je ohraničená například takto.

4	Pohlaví	osob	Procent
5	Muž	35	
6	žena	45	
7	Nezodpovězeno	5	
8			

Princip je jasný, v případě potřeby můžete tabulku doplňovat o tlusté ohraničení okolo, dolní dvojité ohraničení atd. Pokud si trochu pohrajete s formátováním, může vaše tabulka vypadat podobně (nebo ještě lépe).

	A	B	C
1	**Moje první tabulka s grafem**		
2	Kdo zodpovídal dotazník?		
3			
4	Pohlaví	osob	Procent
5	Muž	35	
6	žena	45	
7	Nezodpovězeno	5	
8			
9	Celkem zodpovídalo osob		

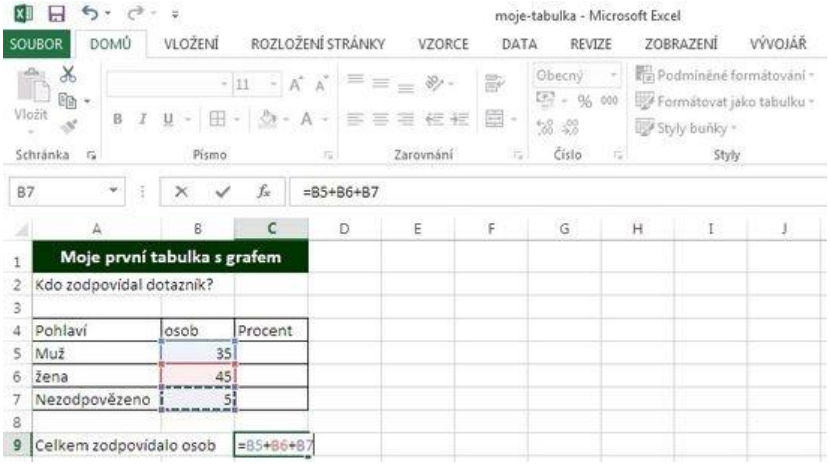
4.3.1.4.4. Výpočty v tabulce

Excel je tabulkový kalkulátor, takže v této kapitole se naučíte provádět jednoduché výpočty (+-*/) a funkce.

4.3.1.4.4.1. Součet

Protože do tabulky potřebujete doplnit procenta, potřebujete nejprve znát celkový počet osob, který odpovídal na dotazník. Můžete sečíst jednotlivé buňky B5, B6, B7 (v další kapitole ukáži, že jde použít i funkci SUMA). Vzorec se do buňky vkládá tak, že nejprve se zadá znaménko rovná se = a poté stačí jednotlivé buňky sečíst tj. do buňky B9 zadejte ***=B5+B6+B7*** stiskem ***Enter*** se provede výpočet. Poznámka: Buď napíšete do buňky její adresu (tj. B6), nebo na buňku kliknete a její adresa se vloží automaticky.

Zadání vzorce:

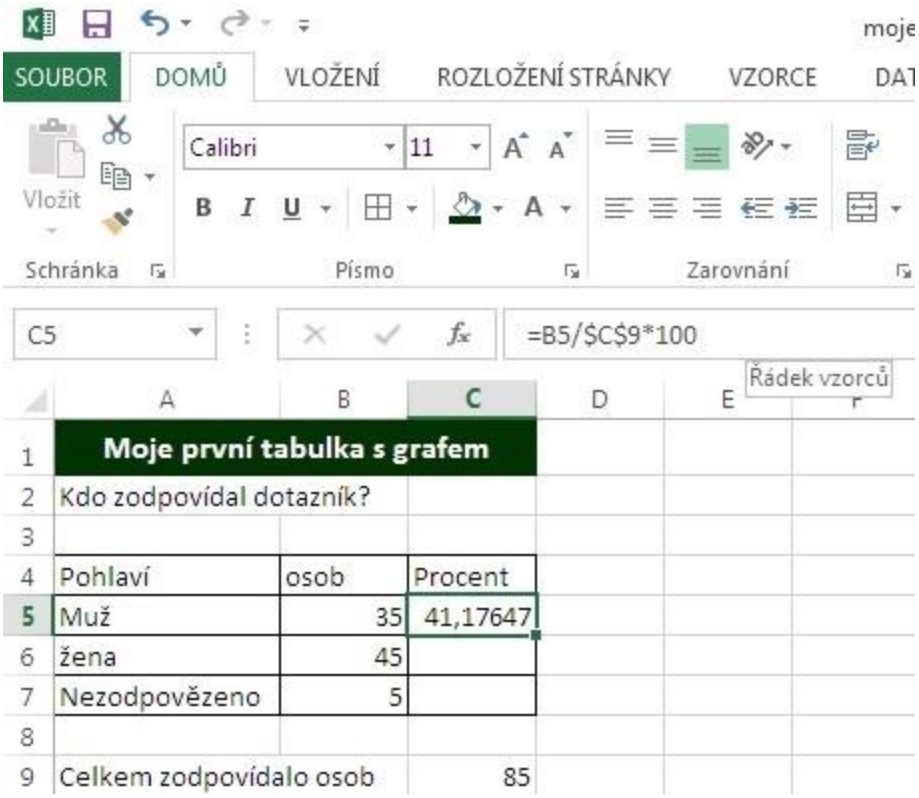


Výsledek

9	Celkem zodpovídalo osob	85
10		

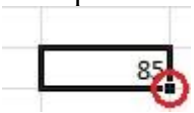
4.3.1.4.4.2. Výpočet procent

Pro výpočet procent jednotlivých odpovědí, použijete vypočtenou hodnotu celkového počtu osob, jež odpovídaly a vypočtete procenta ***=B5/$B$9*100***. Hodnota dolar***$*** u odkazu buňky B9 představuje absolutní odkazování (tj. aby se hodnota této buňky dynamicky neměnila při kopírování. Stisknete ***Enter*** a rozkopírujete vzorec.



4.3.1.5. Rozkopírování vzorce

Nejprve vyberete předchozí buňku (Enter způsobil, že stojíte pod buňkou s výpočtem). Kopírování se pak provede klikem na malý čtvereček (na aktivní buňce - silněji orámována), kurzor se změní na malý kříž. Za současného držení klávesy ***Ctrl*** táhnout myší dolů. Tím bude vzorec rozkopírován na další buňky.



Výsledek bude:

	A	B	C	D
1	**Moje první tabulka s grafem**			
2	Kdo zodpovídal dotazník?			
3				
4	Pohlaví	osob	Procent	
5	Muž	35	41,18	
6	žena	45	52,94	
7	Nezodpovězeno	5	5,88	
8				
9	Celkem zodpovídalo osob		85	
10				

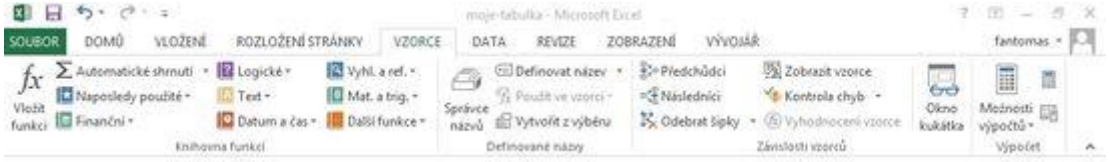
4.3.1.6. Využití vzorců

Excel obsahuje mnoho vzorců (funkcí), které jsou rozděleny do několika kategorií:

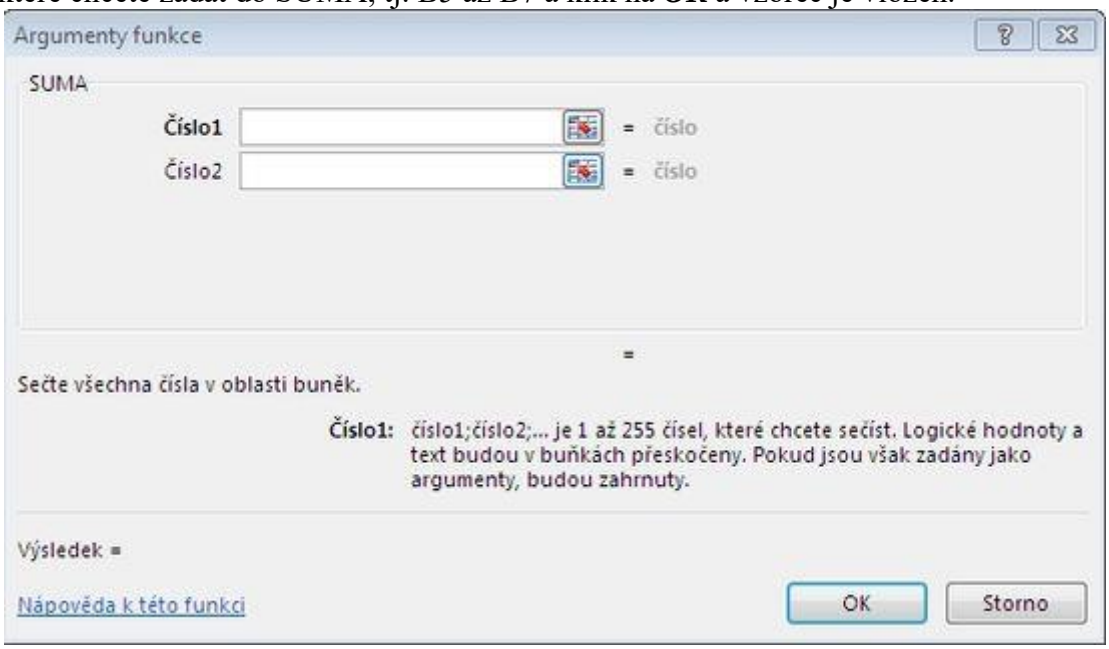
- ✓ Finanční
- ✓ Logické
- ✓ Text
- ✓ Datum a čas
- ✓ Vyhledávací a referenční
- ✓ Matematické a trigonometrické
- ✓ statistické
- ✓ konstrukce
- ✓ datová krychle
- ✓ informace
- ✓ kompatibilita

4.3.1.6.1. Vzorec - knihovna funkcí - SUMA

Sečíst jednotlivé buňky B5, B6, B7 vede ke správnému výsledku, ale je méně profesionální. Pokud buněk bude 200 tak se upíšete. V Microsoft Excelu existuje spousta funkcí. Pro tento případ se hodí funkce ***SUMA***, které provede stejnou akci. Do buňky B10 (pro porovnání) zadejte ***=SUMA(B5:B7)*** stiskem ***Enter*** se provede výpočet. Tento musí souhlasit s předchozím. Případně můžete vložit vzorec přes kartu ***Vzorce*** sekce ***knihovna funkcí*** vyberete ***Mat. a trig.*** z nabízeného seznamu zvolíte ***SUMA***.



V zobrazeném okně ***Argumenty funkce*** klikněte na ikonu  a vyberte požadované buňky které chcete zadat do SUMA, tj. B5 až B7 a klik na ***OK*** a vzorec je vložen.



Podobně vyzkoušejte, zda jsou procenta spočtena správně. Ověřte v buňce C9. Pokud jste vše udělali správně, dostanete hodnotu 100. Vzorec je ***=SUMA(C5:BC7)***.

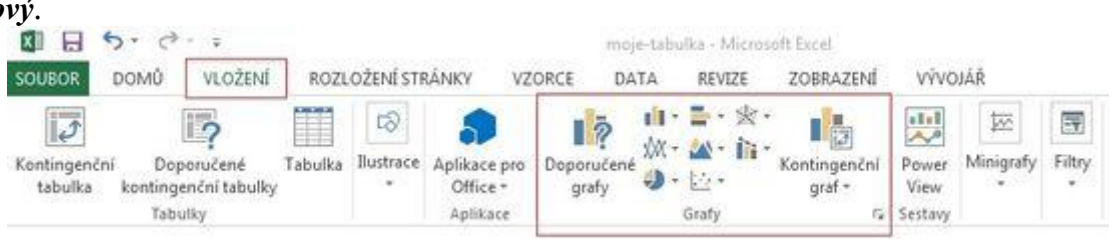
	A	B	C	D
1	**Moje první tabulka s grafem**			
2	Kdo zodpovídal dotazník?			
3				
4	Pohlaví	osob	Procent	
5	Muž	35	41,18	
6	žena	45	52,94	
7	Nezodpovězeno	5	5,88	
8				
9	Celkem zodpovídalo osob	85	=SUMA(C5:C7)	
10				

Výsledek může vypadat:

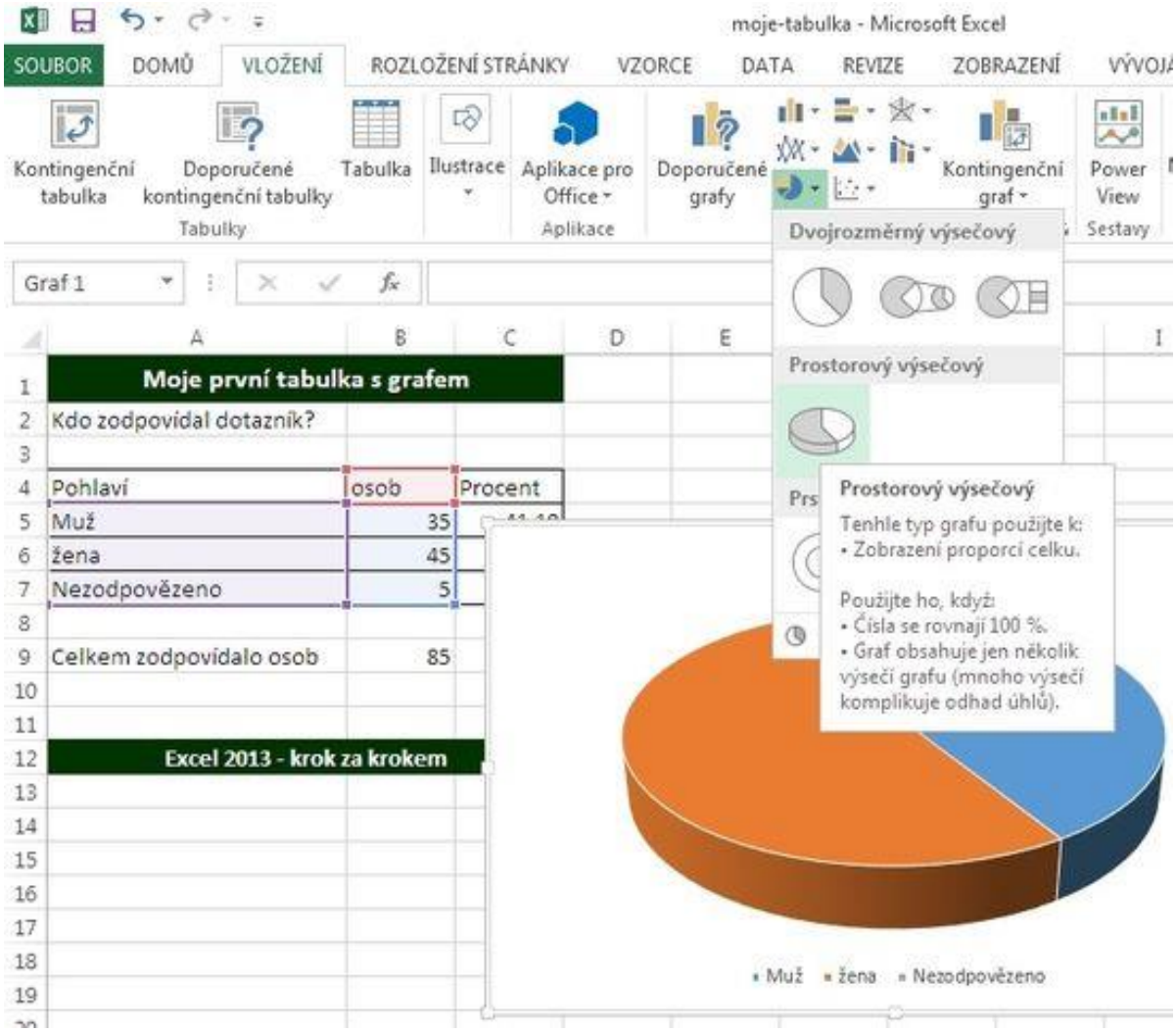
	A	B	C
1	**Moje první tabulka s grafem**		
2	Kdo zodpovídal dotazník?		
3			
4	Pohlaví	osob	Procent
5	Muž	35	41,18
6	žena	45	52,94
7	Nezodpovězeno	5	5,88
8			
9	Celkem zodpovídalo osob	85	100,00
10			

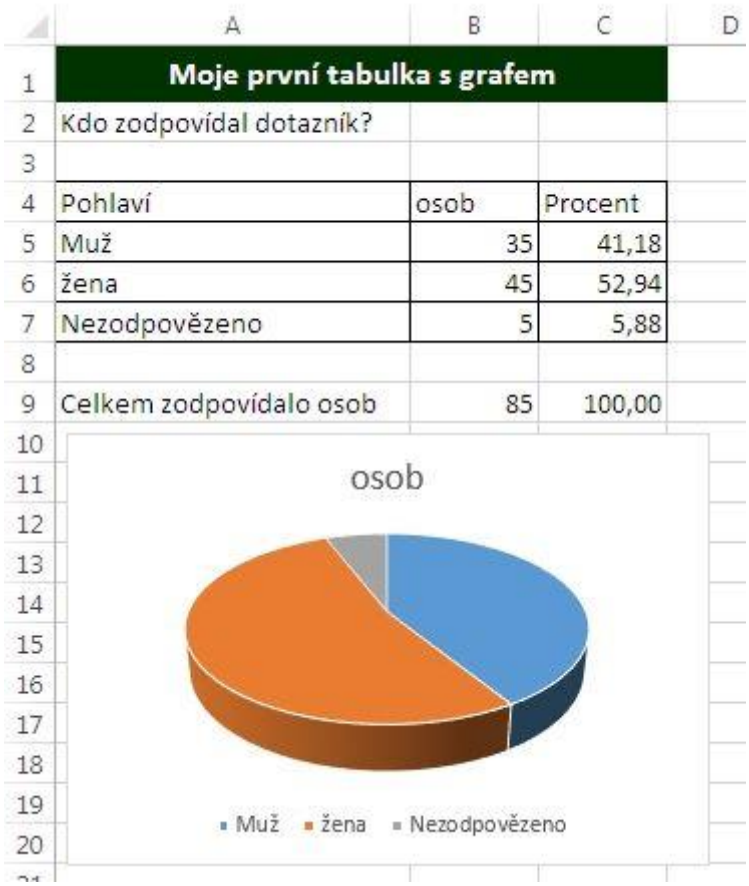
4.3.1.7. Vytvoření grafu

Tabulku máme hotovou včetně vypočtu, tak pro ní udělejte graf. Pro náš ukázkový příklad použijete výsečový (koláčový). Nejprve označte data, která chcete do grafu. Vyberte text odpovědí na otázku a počet odpovědí. Poté na z karty ***Vložení*** sekce ***Grafy*** vyberete stiskem ***Výsečový***.



Na výběr máte několik typů, můžete vybrat například ***Prostorový výsečový***

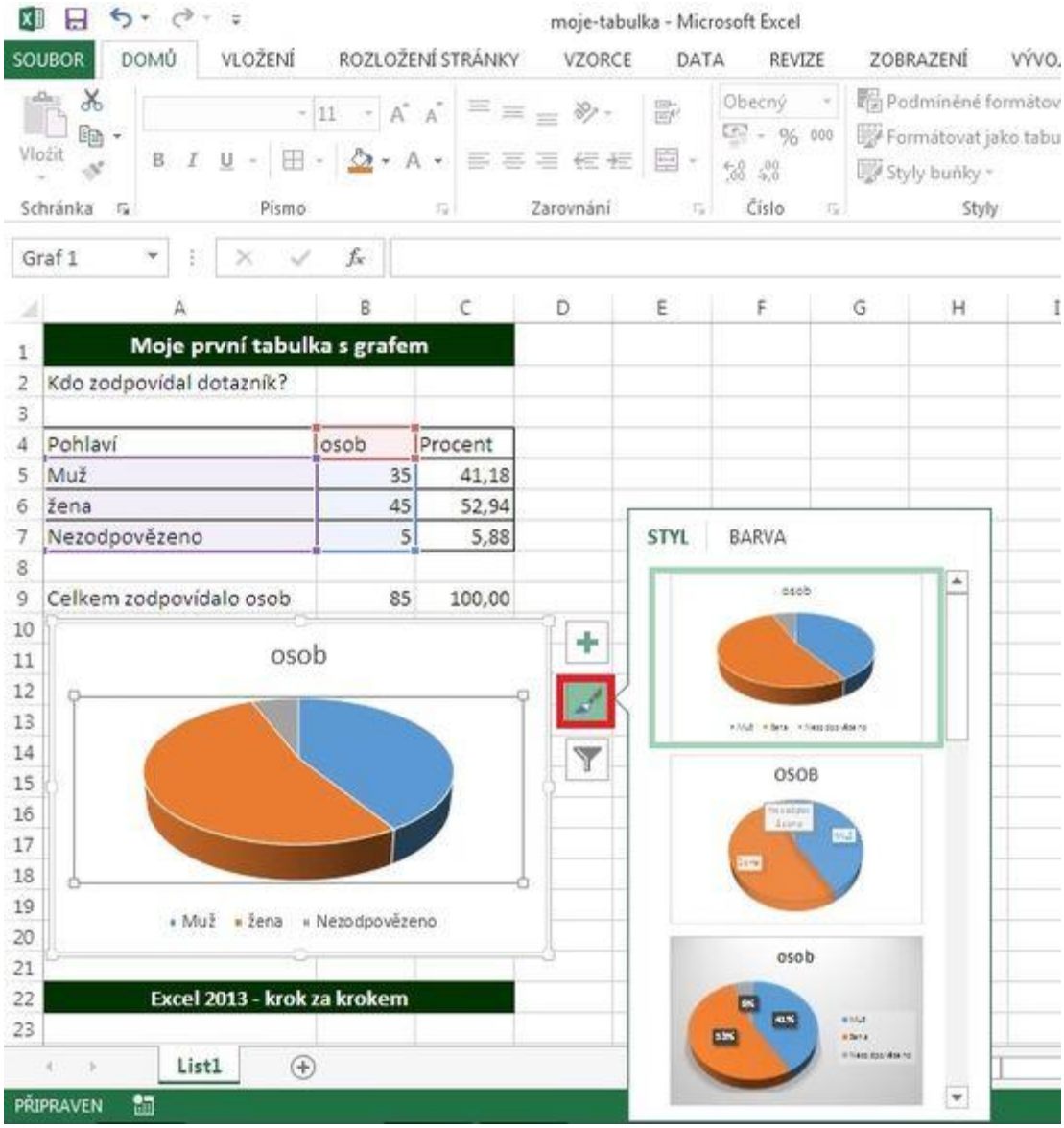

moje-tabulka - Microsoft Excel
SOUBOR
DOMŮ
VLOŽENÍ
ROZLOŽENÍ STRÁNKY
VZORCE
DATA
REVIZE
ZOBRAZENÍ
Kontingenční tabulka
Doporučené kontingenční tabulky
Tabulka
Ilustrace
Aplikace pro Office
Doporučené grafy
Kontingenční graf
Power View
Tabulky
Aplikace
Sestavy
Graf 1
Dvojrozměrný výsečový
Prostorový výsečový
Prostorový výsečový
Tenhle typ grafu použijte k:
• Zobrazení proporcí celku.
Použijte ho, když:
• Čísla se rovnají 100 %.
• Graf obsahuje jen několik výsečí grafu (mnoho výsečí komplikuje odhad úhlů).
Moje první tabulka s grafem
Kdo zodpovídal dotazník?
Pohlaví
osob
Procent
Muž 35
žena 45
Nezodpovězeno 5
Celkem zodpovídalo osob 85
Excel 2013 - krok za krokem
Muž
žena
Nezodpovězeno



Poznámka: Tento graf se dá dále upravovat a zdokonalovat. Je k tomu k dispozici dočasná karta ***Nástroje grafu***



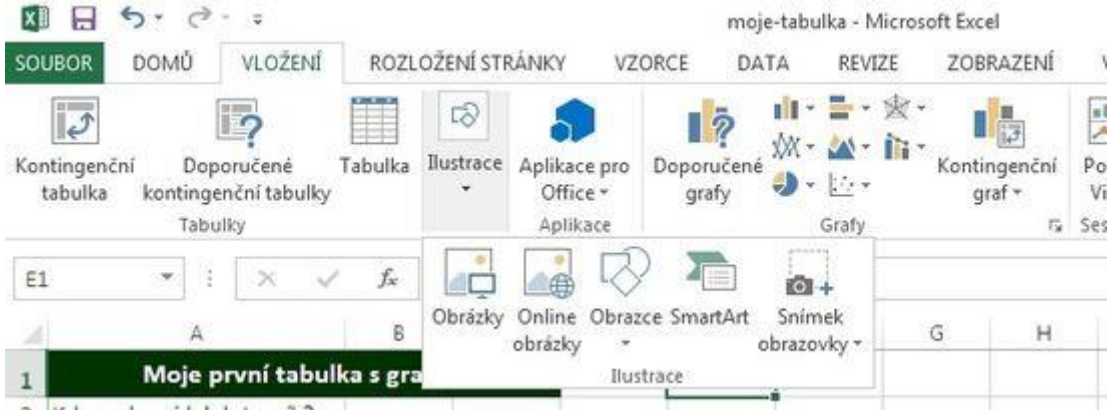
Případně můžete měnit graf pomocí inteligentních ikon:



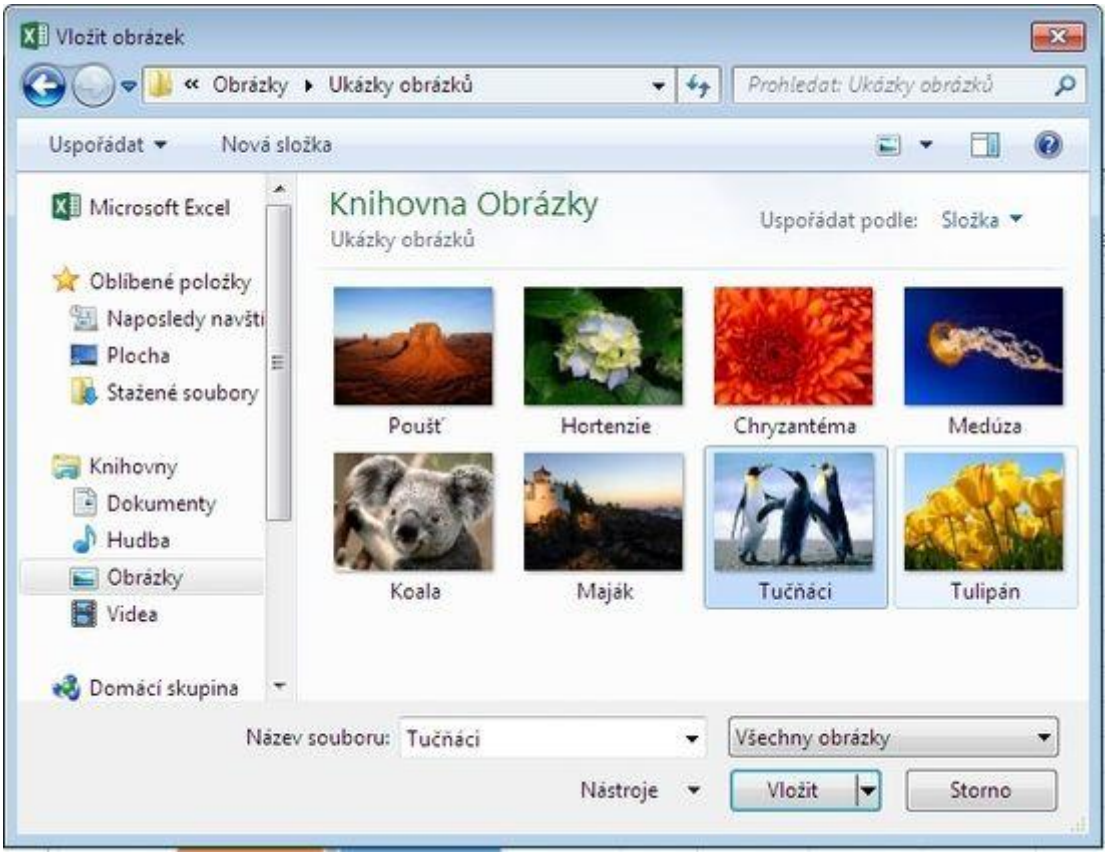
4.3.1.8. Vložení obrázku, klipartu

Vkládání obrázku se může zdát nepotřebné, ale při tvorbě firemních dokumentů se většinou tento neobejde bez loga společnosti, které je většinou obrázkové.

Vložení je jednoduché na kartě ***Vložení*** sekce ***ilustrace*** ikona ***obrázek***.



V zobrazeném dialogovém okně ***Vložit obrázek*** vyberete vhodný obrázek a kliknete na ***Vložit***.



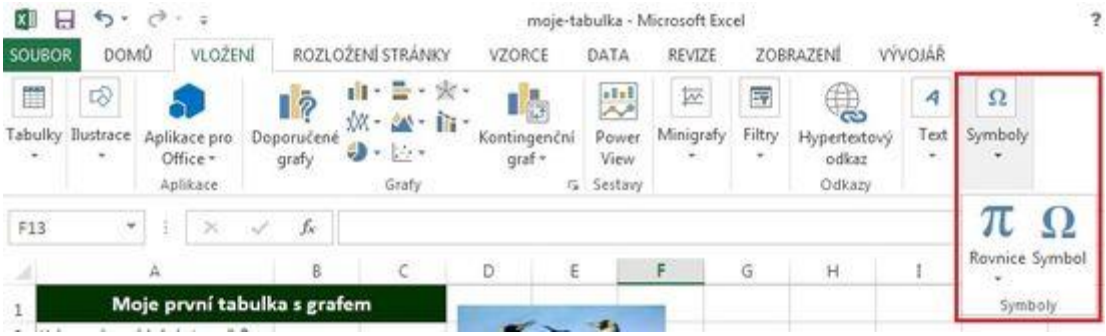
Obrázek vložen



Vložený obrázek můžete přesunout na požadované místo, dále upravovat (zvětšit/zmenšit). Pro další úpravy je k dispozici karta ***Nástroje obrázku - Formát***. Popis přesahuje rámec tohoto článku.

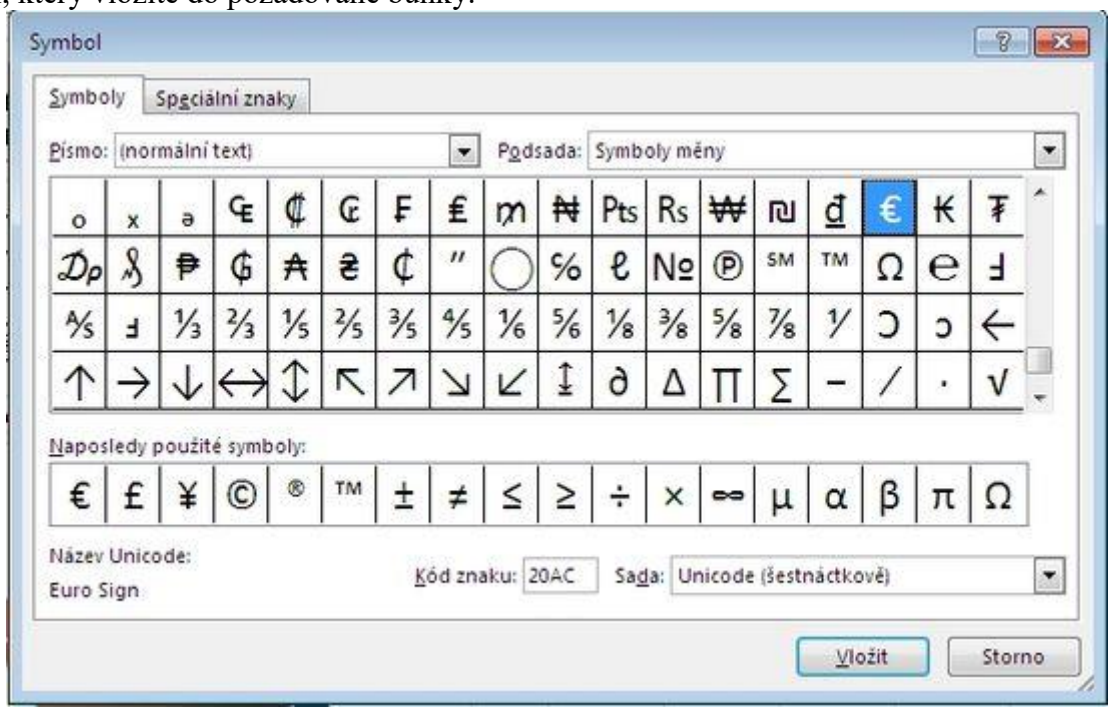
4.3.1.9. Vložení symbolů, vzorců

Podobně jako v MS Wordu i v Excelu lze do buněk vkládat symboly a vzorce. Na kartě ***Vložení*** v sekci ***Symboly*** vyberete ikonu ***Rovnice*** pro vkládání rovnic nebo ***Symbol***, pro vkládání symbolů.



4.3.1.9.1. Vložení symbolu

Klikem na ikonu ***Symbol*** obdržíte okno: ***Symbol***. Zde si můžete vybrat požadovaný symbol, který vložíte do požadované buňky.



4.3.1.9.2. Vložení rovnice

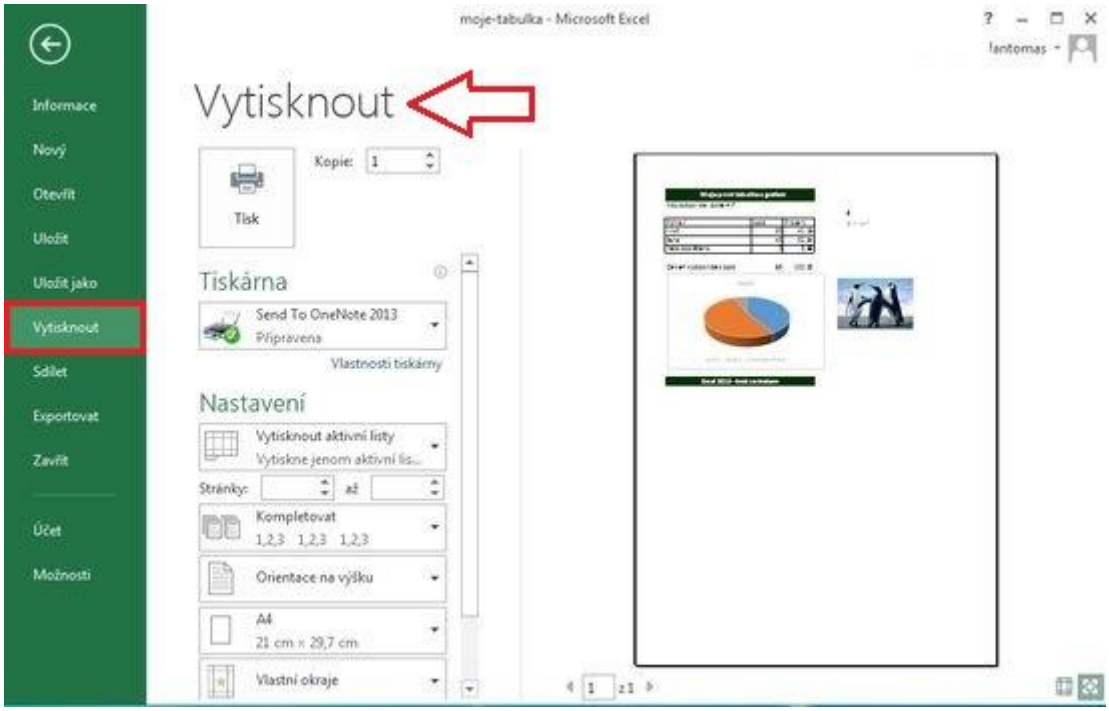
Pro jednoduchost vložte nějakou z předpřipravených rovnic. Na kartě ***Vložení*** v sekci ***Symboly*** vyberete ikonu ***Rovnice*** například ***Povrch kruhu***.

Buňky se symbolem a vzorec mohou vypadat například takto:

D	E	F
	€	
	$A = \pi r^2$	

4.3.1.10. Tisk tabulky s grafem

Na kartě ***Soubor*** v menu ***Vytisknou*** (oproti předchozím verzím, kde bylo pod položkou Tisk) obdržíme okno pro nastavení a přípravu tisku (Vytisknout). **Poznámka:** Pokud máte vybraný graf, bude zobrazen (vytisknut jen tento graf). Co může někdy zmást.



V zobrazeném okně nastavíte požadovanou tiskárnu, orientaci a můžete stisknout tlačítko ***Tisk***.

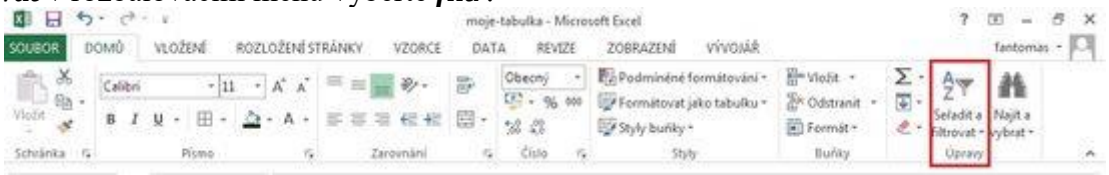
4.3.1.11. Filtry

Přesuňte se na druhý list a vytvořte si podobnou tabulku. Pro jednoduchost je krátká ať na první pohled poznáte, že filtry fungují. Jak si budete s filtry jistí, můžete tabulku rozšiřovat na stovky tisíc řádků.

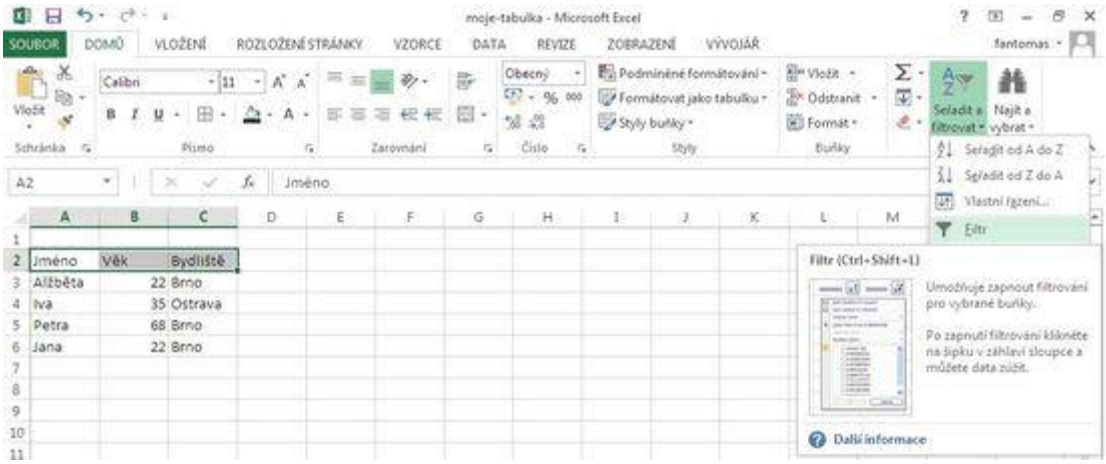
	A	B	C	D
1				
2	Jméno	Věk	Bydliště	
3	Alžběta	22	Brno	
4	Iva	35	Ostrava	
5	Petra	68	Brno	
6	Jana	22	Brno	
7				
8				

4.3.1.11.1. Vložení Filtru

Označte horní řádek tabulky, tj. řádek 2. Na kartě ***Domů*** v sekci ***Úpravy*** ikona ***Seřadit a filtrovat*** v rozbalovacím menu vyberte ***filtr***.



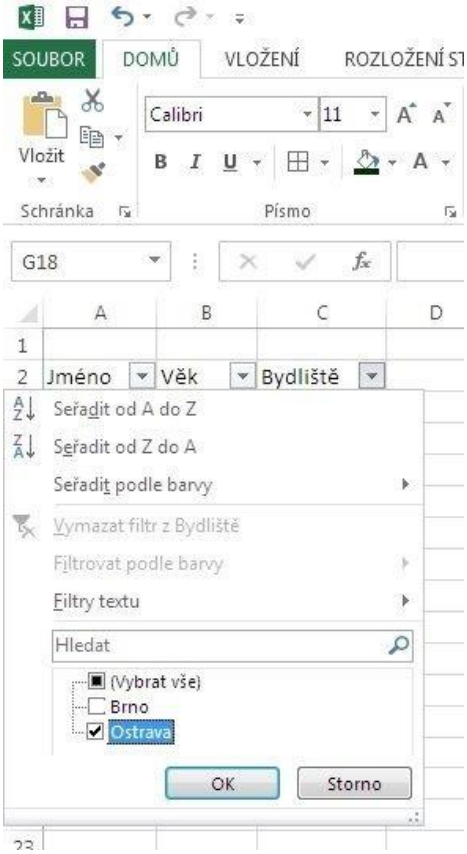
Podrobnější pohled



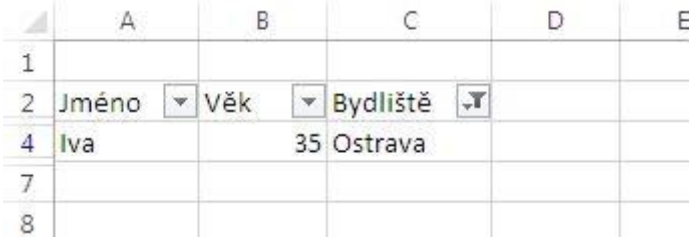
Nadpisy sloupců obdrží filtrovací šipky.

	A	B	C	D
1				
2	Jméno	Věk	Bydliště	
3	Alžběta	22	Brno	
4	Iva	35	Ostrava	
5	Petra	68	Brno	
6	Jana	22	Brno	
7				
8				

Klikem na šipku lze zobrazit menu pro aplikací filtrů. Na daný sloupec se aplikuje požadovaný filtr. V Excelu 2013 lze filtrovat dle mnoha možnosti (větší menší, rovno), můžete zatrhávat příslušné hodnoty, které chcete vyfiltrovat. Navíc můžete filtrovat podle barev :)



Zatrhnete-li u bydliště pouze hodnotu Ostrava, zobrazí se osoby, které splňují tuto podmínku tj. Iva. Navíc čísla řádku, nejdou popořádku (některé chybí - ty které nevyhovují podmínce) a změní se šipka u označení sloupce (ve kterém je proveden výběr).



	A	B	C	D	E
1					
2	Jméno	Věk	Bydliště		
4	Iva	35	Ostrava		
7					
8					

4.3.2. Klávesové zkratky

4.3.2.1. Zkratky pro funkční klávesy

		Alt +	**Ctrl +**	**Shift +**
F1	Nápověda	Vložit graf do listu	Minimalizovat pás karet	–
F2	Úprava aktivní buňky	Uložit jako	Náhled před tiskem	Vložit do buňky komentář
F3	Vložit název	–	Dialogové okno – Správce názvu	Vložit funkci – dialogové okno
F4	Opakovat operaci odkaz – absolutní relativní	Zavřít Excel	Uložit změny?	Přesun na další sloupec
F5	Přijít na	–	Zmenšit okno sešitu	Najít a nahradit
F6	Přepnutí další podokno	–	Další otevřený sešit	Přepnutí předchozí podokno
F7	Kontrola pravopisu	–	Přesun okna sešitu	Zdroje informací
F8	Rozšířit výběr Makro	–	Změnit velikost okna	Přidat k výběru
F9	Výpočet listu	–	Minimalizovat okno sešitu	Přepočet aktuální list
F10	Aktivace pásu karet	–	Maximalizovat okno sešitu	Zobrazit nabídku

		Alt +	Ctrl +	Shift +
F11	Vložit graf nový list	Otevřít Visual Basic	Vložit nový list	Nový list
F12	Uložit jako	–	Otevřít – Dialogové okno	Uložit sešit

4.3.2.2. Zkratky s Ctrl

- **Ctrl + A** Označení všech buněk
- **Ctrl + B** Tučné/netučné písmo (*funguje pro následující psané znaky*)
- **Ctrl + C** Zkopíruje obsah označených buněk do schránky
- **Ctrl + D** Kopírování obsahu a formátu horní buňky do buňky pod ní
- **Ctrl + F** Najít (*otevře dialogové okno*)
- **Ctrl + G** Přejít na (*otevře dialogové okno*)
- **Ctrl + H** Nahradit (*otevře dialogové okno*)
- **Ctrl + I** Písmo kurzivou/nekurzivou (*funguje pro následující psané znaky*)
- **Ctrl + K** Vloží hypertextový odkaz (*otevře dialogové okno*)
- **Ctrl + L** Vytvoří seznam
- **Ctrl + N** Otevře nový sešit
- **Ctrl + O** Otevře soubor
- **Ctrl + P** Vytiskne
- **Ctrl + R** Kopírování obsahu a formátu levé buňky do buňky vpravo
- **Ctrl + S** Uložení souboru
- **Ctrl + S** Uložení souboru
- **Ctrl + T** Zobrazení dialogového okna: ***Vytvořit tabulku***
- **Ctrl + U** Podtrženo/Nepodtrženo
- **Ctrl + V** Vloží do buňky obsah schránky
- **Ctrl + W** Zavře otevřený soubor (*pokud byla provedena změna, možnost uložit*)
- **Ctrl + X** Vyjme obsah označených buněk
- **Ctrl + Y** Opakování posledního příkazu nebo akce (pokud je to možné)
- **Ctrl + Z** Vrácení posledního příkazu zpět nebo odstranění naposledy zadané položky
- **Ctrl + 1** Zobrazení dialogového okna ***Formát buněk***
- **Ctrl + Home** Přesun do buňky A1
- **Ctrl + End** Přesune kurzor na poslední plnou buňku
- **Ctrl + Enter** Do všech označených buněk stejný obsah
- **Ctrl + PageDown** Přepnutí na další list
- **Ctrl + PageUp** Přepnutí na předcházející list
- **Ctrl + Mezerník** Výběr celého(celých) sloupce(ů)
- **Ctrl + Shift + Mezerník** Výběr celého listu
- **Ctrl + F1** Zobrazení/skrytí – pás karet
- **Ctrl + F2** Náhled před tiskem
- **Ctrl + F3** Správce názvu – dialogové *okno*
- **Ctrl+Shift+Enter** Vložení maticového vzorce

4.3.2.3. Zkratky s Alt

- **Alt + Enter** Zahájení nového řádku ve stejné buňce (*vcelku šikovná pomůcka*)
- **Alt + =** Vložení vzorce SubTotal
- **Alt + F1** – Vytvoření grafu do listu
- **Alt + F11** – VBA *spustí se VBA editor*

4.3.2.4. Zkratky se Shift

- **Shift + F2** – Úprava komentáře
- **Shift + F3** – Vložit funkci *dialogové okno*
- **Shift + F8** – tvorba/netvorba nesousedících výběru *viz info stavový panel*
- **Shift + F9** – přepočet aktivního listu
- **Shift + F10** – místní nabídka pro buňku
- **Shift + F11** – vložení nového list
- **Shift + Enter** – přesunete se o buňku nahoru
- Enter – přesunete se o buňku dolů
- **Shift + Tab** – přesunete se o buňku vlevo
- Tab – přesunete se o buňku vpravo

4.3.2.5. Zkratky ostatní

- Tab Přesunutí na další buňku, nebo přesunutí v zamknutém listu mezi odemknutými buňkami
    - **Shift + Tab** Přesunutí o buňku zpět (vlevo)
- **Šipka dolů** Přesun o buňku dolů
- **Šipka nahoru** Přesun o buňku nahoru
- **Šipka vlevo** Přesun o buňku vlevo
- **Šipka vpravo** Přesun o buňku vpravo
- **Home** Přesunutí na začátek řádku

4.3.2.6. Navigace uvnitř listu

- **Alt + Page Down** – Pohyb v listu o jednu obrazovku vpravo
- **Alt + Page Up** – Pohyb v listu o jednu obrazovku vlevo
- **Alt + šipka dolů** – Zobrazení seznamu pro automatický filtr
- **Ctrl + End** – Přesun do poslední buňky s obsahem
- **Ctrl + F** – Zobrazí se dialogové okno ***Najít***
- **Ctrl + H** – Zobrazí dialogové okno ***Nahradit***
- **Ctrl + Home** – Přesun na začátek listu (A1)
- **Ctrl + G** – Zobrazit dialogové okno ***Přejít na*** (také zkratka *F5*)
- **Ctrl + šipky** – Přesun na okraji další oblasti dat
- **Ctrl + šipka vlevo** – Uvnitř buňky: Přesun o jedno slovo doleva
- **Ctrl + šipka doprava** – Uvnitř buňky: Přesun o jedno slovo doprava
- **End** – Uvnitř buňky: přesunout na konec buňky
- **F2** – Kurzor na poslední pozici v buňce
- **F5** – Zobrazit dialogové okno Přejít na (také zkratka *Ctrl + G*)
- **Home** – Přesun na začátek řádku v kterém se nachází kurzor (tj. do sloupce A)
- **Home** – Uvnitř buňky: přesunout na začátek buňky.
- **Page Down** – Přesun o jednu obrazovku dolů
- **Page Up** – Přesun o jednu obrazovku dolů
- **Shift + Tab** – Přesunutí o jednu buňku vlevo
- **Shift + F4** – Opakujte poslední nález
- **Šipky** – Přesun o jednu buňku nahoru, dolů, doleva doprava

- **Tab** – Přesunutí o jednu buňku vpravo

4.3.2.8. Výběr buněk v listu

- **Ctrl + *** ***(hvězdička)*** - Vybere aktuální oblast kolem aktivní buňky (tabulku)
- **Ctrl + A** – Vybere aktuální oblast kolem aktivní buňky (druhý stisk *Ctrl + A* celý list)
- **Ctrl + Mezerník** – Vybere sloupec
- **Ctrl + Shift + End** – Rozšíření výběru k poslední použité buňce
- **Ctrl + Shift + Home** – Rozšíření výběru k začátku listu
- **Ctrl + Shift + Mezerník** – Vybere celý list
- **Ctrl + Shift + O** – Výběr všech buněk s komentáři
- **Ctrl + Shift + PgUp** – Vybere aktuální a předchozí list v sešitu
- **Ctrl + Shift + PgDn** – Vybere aktuální a následující list v sešitu
- **Ctrl + Shift + Šipky** – Rozšíření výběru k okrajové buňce s obsahem
- **Esc** – Zrušit výběr
- **Shift + Home** – Rozšíření výběru k začátku řádku
- **Shift + Mezerník** – Vybere řádek
- **Shift + Šipky** – Rozšíření aktuálního výběru o jednu buňku

4.3.2.9. Práce s aktivním (označeným) výběrem

- **Ctrl + Alt + Šipka vpravo** – Přesunout vpravo mezi nesousedícími oblastmi výběru
- **Ctrl + Alt + Šipka vlevo** – Přesunout vlevo mezi nesousedícími oblastmi výběru
- **Ctrl + .** ***(tečka)*** - Pohyb aktivní buňky po rozích výběru
- **Enter** – Přesun aktivní buňky dolů ve výběru
- **Esc** – Zrušit výběr
- **F8** – Zapnout rozšíření výběru (šipkami se pak zvětší výběr)
- **Shift + Mezerník** – Vybrat pouze aktivní buňky, pokud je vybráno více buněk
- **Shift + Enter** – Přesun aktivní buňky nahoru ve výběru
- **Shift + Tab** – Přesun aktivní buňku vlevo ve výběru
- **Tab** – Přesun aktivní buňku vpravo ve výběru

4.3.2.10. Vkládání speciálních znaků

- **Ctrl+Alt+E** – znak € *(také zkratka AltGr + E)*
- **Ctrl+Alt+Q** – znak \ *(také zkratka AltGr + Q)*
- **Ctrl+Alt+W** – znak | *(také zkratka AltGr + W)*
- **Ctrl+Alt+S** – znak đ *(také zkratka AltGr + S)*
- **Ctrl+Alt+D** – znak Đ *(také zkratka AltGr + D)*
- **Ctrl+Alt+F** - znak [*(také zkratka AltGr + F)*
- **Ctrl+Alt+G** - znak]*(také zkratka AltGr + G)*
- **Ctrl+Alt+L** - znak ł*(také zkratka AltGr + L)*
- **Ctrl+Alt+X** - znak #*(také zkratka AltGr + X)*
- **Ctrl+Alt+C** - znak &*(také zkratka AltGr + C)*
- **Ctrl+Alt+V** - znak @*(také zkratka AltGr + V)*
- **Ctrl+Alt+B** – znak {*(také zkratka AltGr + B)*
- **Ctrl+Alt+N** – znak } *(také zkratka AltGr N)*
- **Ctrl+Alt+,** ***(<)*** - znak < *(také zkratka AltGr + ,)*
- **Ctrl+Alt+.** ***(>)*** - znak > *(také zkratka AltGr + .)*

4.3.3. Data v buňkách – co obsahuje, co lze vkládat

4.3.3.1. Data v buňce

Sešit v Microsoft Excelu se skládá z jednotlivých listu. List se poté skládá z buněk. Každá buňka má svou jedinečnou adresu. Do každé buňky lze umístit informace (hodnotu, formát, text, ...). Dále se podíváme podrobněji.

4.3.3.2. Co buňka obsahuje

Každá jednotlivá buňka, může obsahovat:

- **Hodnota** – číselné, textové, datum čas, ….
  - **Číslo** – číselné hodnoty, což kromě samotného čísla může představovat datum, čas.
  - **Text** – Jakýkoliv text, omezení jen délkou pro jednotlivou buňku.
  - **Datum, čas**
- **Vzorec (funkce)** - v buňce se provede operace, nejen matematické, k dispozici jsou funce statistické, finanční, textové,... Místo správného výpočtu může být obsahem chybová hodnota.
- **Formát (automaticky, vlastní, podmíněný)** - například barva, ohraničení (obsahem buňky může být něco jiného, než je zobrazeno například vzorec, datum čas, ...)
- **Hypertextový odkaz** – například klikatelnýá odkaz na web office.lasakovi.com :)
- **Komentář** – komentář (*klasický žlutý lístek*) můžete upravovat (přiřadit mu tvar barvu)
- **Minigraf** – od Verze Excel 2010 může být v buňce i graf (minigraf).

4.3.3.2.1. Hodnota

Zadávat lze různé hodnoty (číslo, text, datum, čas, procenta). Toto lze povést:

- **Ručně** – zadá se ručně
- **Automaticky – klávesovou zkratkou** – klávesovou zkratkou lze vložit aktuální datum čas (viz dále)
- **Automaticky – makro VBA** – VBA (makro) - může automaticky vkládat údaje do příslušných buněk.
- **Automatická změna formátu** – po zadáni hodnoty data, času ve správném času dojde k automatické změně formátu (= pro vkládání vzorce).
- **Ruční změna formátu** – pomocí menu můžete nastavit příslušné formáty

Ručně

Ruční zadávaní, nejčastější, do kativní buňky zadáte požadovanou hodnotu. Pro specifická zadání se provede automatický formát (*o tom podrobněji v další kapitole tohoto článku*).

Poznámka: - Klávesová zkratka **F2** pro aktivaci zápisu do buňky (pokud nechcete používat myš).

Automaticky klávesovou zkratkou

Existuje několik klávesových zkratek, které ihned zadají do buňky příslušnou hodnotu:

- **Ctrl+;** - zadá aktuální datum
- **ALT+ +** - Vložení vzorce SubTotal

Další oblíbená klávesová zkratka, pokud máte data ve schránce:

- **Ctrl + V** – Ze schránky

Poznámka: Pokud data mají jiný rozměr, může při vkládaní dojít k chybě.

Klávesová zkratka pro vkládaní matic.

Automaticky – makro VBA

Zadávání hodnot (údajů) pomocí VBA a maker je popsáno v samostatných článcích (přesahuje možnosti tohoto článku).

Změna formátu (automatická ruční)

Při vložení některých specifických hodnot dojde k automatické změně formátu, když vkládate datum, čas, procenta. Například: 1.1.2013

Poznámka: Vložená hodnota se automaticky změní na formát datum (a patřičně se naformátuje číslo v buňce). Na první pohled sice stále vidíte ono datum, ale ve skutečnosti je v buňce číslo. Viz další kapitola o formátu.

K další automatické změně dojde při zložení symbolu **= (rozná se)**. Excel automaticky předpokládá vložení funkce (popis funkci následující kapitola).

' apostrof Při použití apostrofu bude Excel ignorovat použití automatického formátu. Takto můžete zobrazit '=A10+A12, bez použíti apostofu se provede výpočet.

Funkce (vzorce) - zadávaní/úprava

Jak jsem psal výše, aby Excel poznal, že jde o vzorec (funkci), musíte jej uvodit znakem rovno =. Poté již můžete vkládat vzorec. Pro vložení funkce (vzorce) máte opět několik možností:

- **Ručně**
- **Myší**
- **ze schránky** – Ctrl + V
- **VBA** – makro

Ručně

Nejrychlejší možnost, pokud znáte příslušnou funkci, můžete začít ihned psát, například:

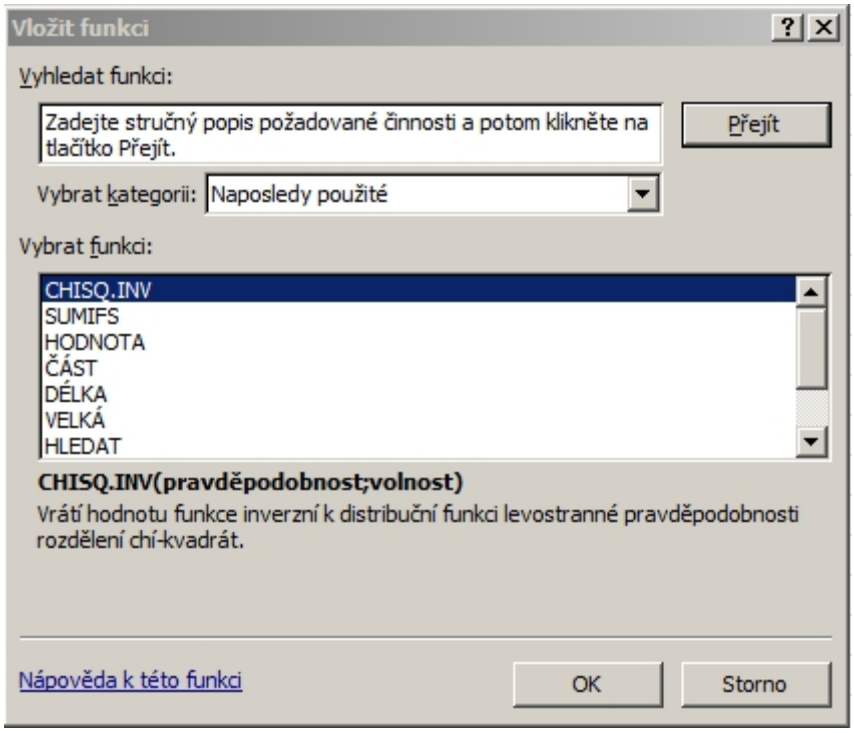
=KDYŽ(A10=1; A3;A5)

Příklady funkcí (vzorců):

- **=A1** – odkaz na buňku A1, pokud v ní bude hodnota 24, bude i v buňce (B2) hodnota 24 (nemateli změněno formátování)
- **=1+1** - nemusíte se vůbec nikam odkazovat a Excel Vám spočítá, kolik je 1+1
- **=A1+A2** – spočtete součet, který dávají čísla v buňkách A1 a A2 (pokud v buňkách nebude číslo, obdržíte chybovou hodnotu)

Myší

Pokud funkci neznáte, můžete pomocí myši vybrat z pasu karet **Vzorce** sekce **knihovna funkcí** ikona **Vložit funkci** v zobrazeném dialogovém okně si již vyberete co potřebujete.



Klávesová zkratka (Ctrl + V). Podobně jako u vložení hodnot, pokud máte ve schránce příslušný funkci (vzorec) tento bude vložen do buňky (buněk).

VBA

' apostrof Při použití apostrofu bude Excel ignorovat použití automatického formátu. Takto můžete zobrazit '=A10+A12, bez použíti apostofu se provede výpočet.

Formátování (včetně podmíněného)

Formátování buňek je nedilnou součásti hodnoty. Každá hodnota je po zadáni automaticky naformátovana. Tento formát pak můžete nastavit (změnit).

Ruční nastavení formátu

Automatické formátování

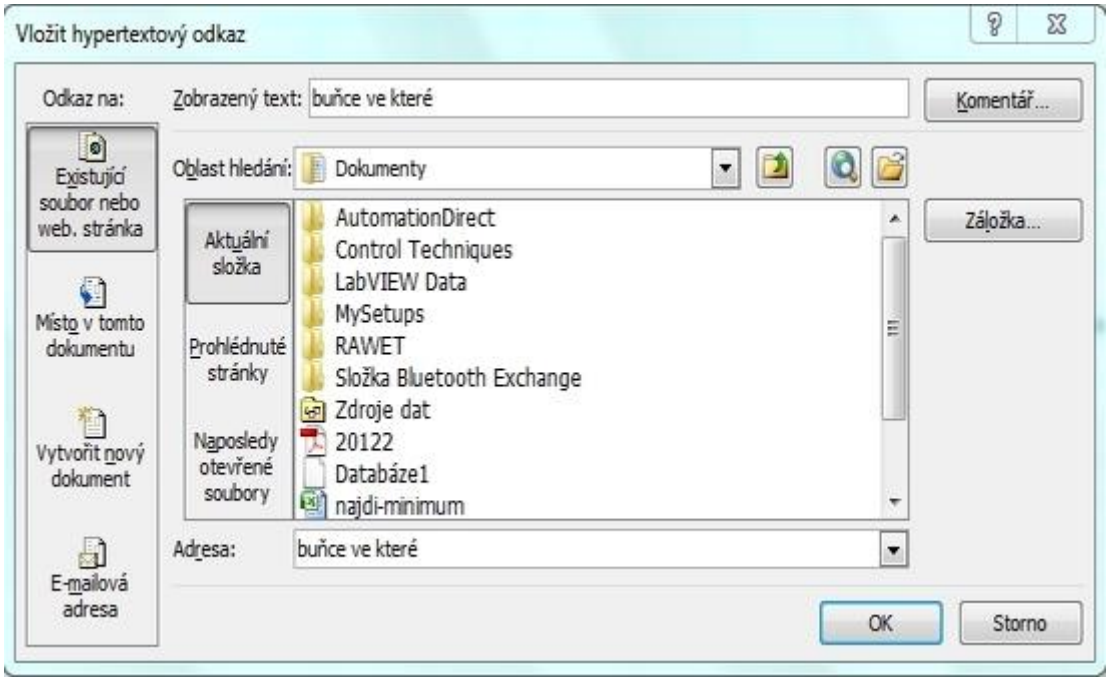
Excel automaticky formátuje zadávané hodnoty (převádí na číslo, datum,) Zkuste zadat a uvidíte co se stane:

- A – zustane tak jak je
- 23 - automaticky se zarovná doprava (jedná se o číslo)
- 12.1.2012 - automaticky se převede na datum
- 12:12 - automaticky se převeden na čas

Poznámka: - při použití apostrofu ' nebude použito automatické formátování, text zůstane tak jak jej napíšete (například vzorec, datum, čas, ...).

Hypertextový odkaz

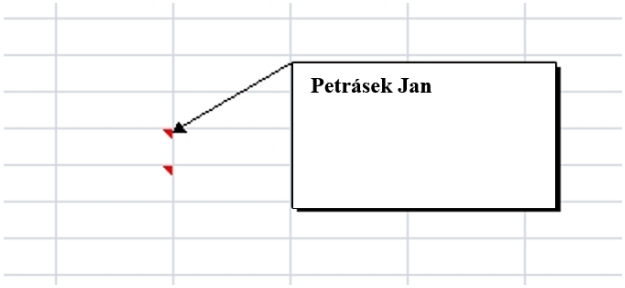
V Excelu se můžete odkazovat na web. Například na tyto stránky :) http://office.lasakovi.com. Nejjednodušeji v buňce, do které potřebujete zadat hypertextový odkaz. Pravým tlačítkem a vyberete hypertextový odkaz.



Do Zobrazený text: zadáte co chcete, aby se zobrazilo v buňce, např: *office.lasakovi.com - nejlepší web o Excelu :)* a do Adresa: zadáte *http://office.lasakovi.com/excel/* kliknete OK a máte hotovo.

Komentáře

Do buňky se dají vkládat komentáře. Nejjednodušeji přes pravý klik na buňce a vložit komentář. Nebo na kartě revize sekce komentáře.



Poznámka: - Buňka, která obsahuje komentář má v pravém horním rohu červený praporek.

Ukázka buňky s miniografem.

	A	B	C	D	E	F	G	H
1								
2		1	2	3	4	5		
3		1	2	3	2	1		
4								
5								

4.3.4. Operátory

V tomto článku jsou představeny různé typy operátoru (znamének), které jsou používany v Microsoft Excelu. Jak se dají tyto operátory dají použít.

Operátory určují typ výpočtu, který chcete s prvky vzorce provést. V aplikaci Microsoft Excel jsou zahrnuty čtyři typy (kategorie): matematické (aritmetické), relační, textové, odkazovací.

4.3.4.1. Vzorec

= Vzorec v Excel vždy začíná znaménkem rovná se (=). Toto znaménko označuje, že následující znaky tvoří vzorec, který Excel vypočte. Po zadání ***=1+2*** se v buňce objeví ***3***

Zobrazit vzorec jako text

Excel automaticky po zadání znaku rovná se začne následující text brát jako vzorec (chce počítat). Obejdete použitím apostrofu:
' ***apostrof*** - co je za apostrofem se bere jako text (tedy i znaménko rovná se), takže '=1+2 neprovede výpočet ale zůstane zobrazeno vše co je za apostrofem tj. ***=1+2***
Poznámka: Pokud chcete zobrazit apostrof musíte jej zadat 2x (ten první se totiž nezobrazí).

4.3.4.2. Aritmetické operátory

Jak již název napovídá jsou určeny k provádění matematických výpočtů (sčítání, odčítání, násobení...).

Operátor	Popis	Význam	Příklad
+	znaménko plus	Sčítání	1+1
–	znaménko minus	Odčítání	5–2
–	znaménko minus	Záporné číslo	–1
*	hvězdička	Násobení	2*2
/	lomítko	Dělení	10/2
%	procento	Procenta	10%
^	„stříška“	Umocnění	2^2

4.3.4.3. Relační operátory

Využití pro porovnávání dvou hodnot, kdy výsledkem je logická hodnota PRAVDA nebo NEPRAVDA.

Operátor	Popis	Význam	Příklad
=	znaménko rovná se	Rovná se	A1=B1
>	znaménko větší než	Větší než	A1>B1
<	znaménko menší než	Menší než	A1<B1
>=	znaménko větší než nebo rovno	Větší než nebo rovno	A1>=B1
<=	znaménko menší než nebo rovno	Menší než nebo rovno	A1<=B1
<>	znaménko nerovná se	Nerovná se	A1<>B1

4.3.4.4. Textové operátory

Například pro spojení (zřetězení) jednoho nebo více textových řetězců. Bude vytvořen jeden dlouhý řetězec.

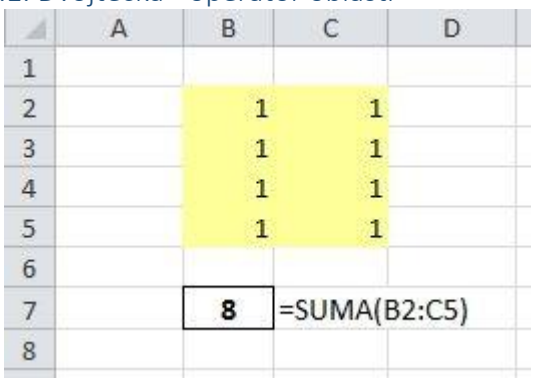
Operátor	Význam	Příklad
&	Spojuje jednotlivé hodnoty do souvislé textové hodnoty.	„Auto“&„mobil“

4.3.4.5. Odkazovací operátory

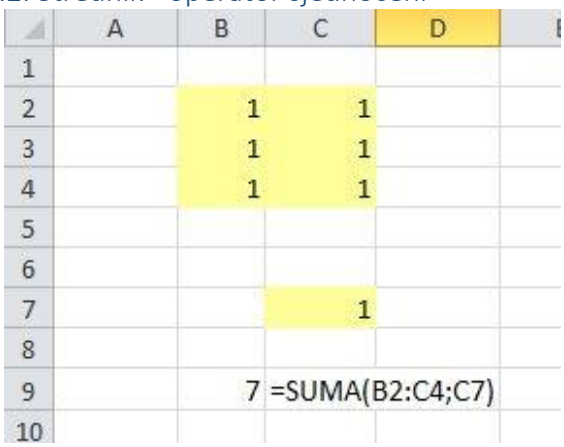
Pro kombinování oblasti buněk, například ve výpočtech.

Operátor	Popis	Význam	Příklad
:	dvojtečka	**operátor oblasti** Jeden odkaz na všechny buňky mezi dvěma odkazy včetně těchto odkazů	(B2:C5)
;	středník	**operátor sjednocení** Kombinuje více odkazů do jednoho odkazu (sjednocuje)	(B2:C4;C7)
	mezera	**operátor průniku** Vytváří jeden odkaz na buňky společné dvěma odkazům (tj. průnik), viz ukázka	(B7:D7 C6:C8)

4.3.4.5.1. Dvojtečka - operátor oblasti

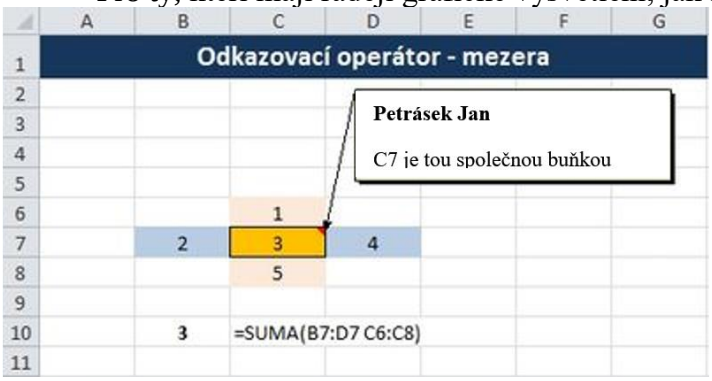


4.3.4.5.2. Středník - operátor sjednocení



4.3.4.5.3. Mezera – operátor průniku

Pro ty, kteří mají raději grafické vysvětlení, jak funguje operátor mezera.



4.3.4.6. Nadřazenost (priorita) operátorů

Zkombinování několika operátorů do jednoho vzorce, se budou operace prováděny v pořadí podle níže uvedených priorit. Pokud je ve vzorci několik stejně nadřazených operatorů (násobení a dělení), jsou operátory vyhodnoceny zleva doprava. Chceteli něco upřednostnit musíte použít závorky =(2+2)*2.

Priorita	Operátor	Popis	Poznámka
1	:	dvojtečka	Např.: A1:A129
2		jedna mezera	
3	;	středník	
4	–	zápor (unární operátor)	Např.: –1
5	%	procenta	Např.: 20%
6	^	umocnění	Např.: 2^128
7	* /	násobení a dělení	Např.: A2*B7
8	+ –	sčítání a odčítání	Např.: A4-28
9	&	spojení (zřetězení) řetězcu	Nappř.: „Pavel“&„ „&„Novák“
10	= < > <= >= <>	srovnání	Např.: A29<>A12

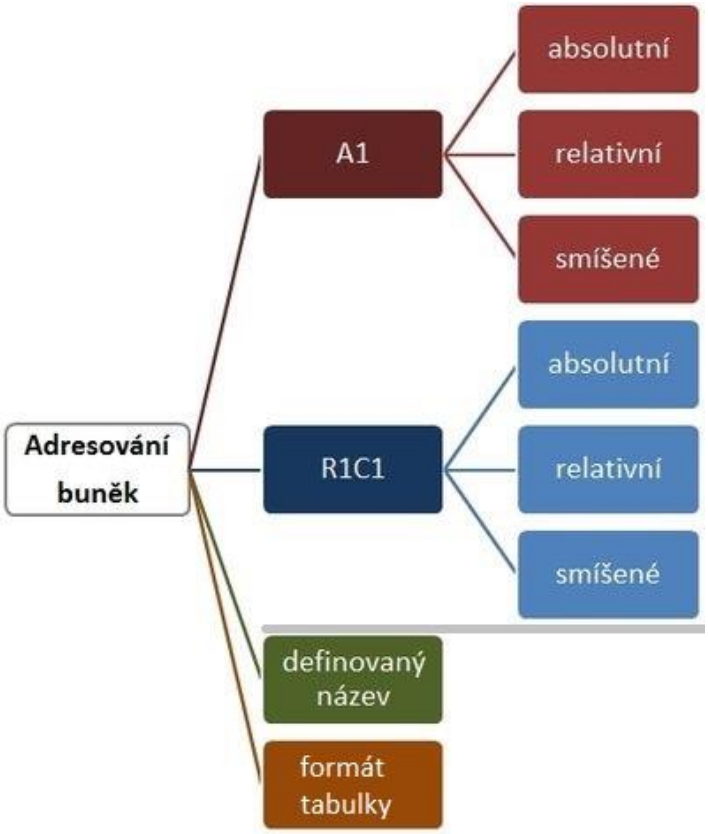
4.3.5. Relativní a absolutní odkazy

V Excelu se bez odkazů neobejdeme. Pokud v Excelu provádíme výpočty (neopisujete otrocky čísla) musíte používat odkazy. Zadáváním do buněk přímo hodnot pro výpočet např. ***=1+1***, je nesmysl. Buňka výpočet provede, ale při požadavku na změnu hodnot (místo 1 budete potřebovat zadat 2) musíte projít jednotlivé buňky a hodnoty ručně opravit ve vzorci (což je zdlouhavé, nepraktické a zbytečné). Navíc na první pohled nevíte z jakých čísel je výpočet proveden.

U mnoha funkcí (např. vyhledávací, logické) si nedovedu představit smysluplný zápis vzorce (funkce) bez odkazů. Proto správné, rychlé, elegantní je, mít v část buněk pro hodnoty a v dalších buňkách příslušné výpočty. Pokud se změní jedna hodnota, změníte příslušnou buňku a ostatní buňky se přepočítají automaticky sami.

Pokud odkazování pochopíte, uznáte, že myšlenka odkazování je geniální a bez odkazu se již neobejdete.

V Excelu se dá odkazovat:



4.3.5.1. Druhy odkazů

V Excelu existují druhy/typy odkazů:

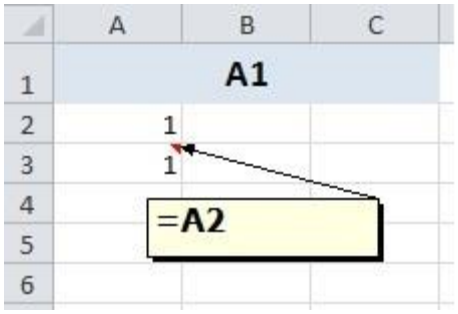
- **A1 – klasickým** - tj. že řádky jsou označeny čísly a sloupce písmeny (nejčastější použití) - tato odkazy navíc mohou být:
  - absolutní
  - relativní
  - smíšení
- **R1C1 – programátorským** - méně používaný, ale pro programátory výhodný (lépe se orientuje v rozsáhlých tabulkách, hlavně pokud programujete ve VBA) i tyto odkazy mohou být:
  - absolutní
  - relativní
  - smíšení

  Navíc se v Excelu lze odkazovat pomocí:
- ***Definované názvy*** – jde vlastně o absolutní odkaz, kdy daná buňka (*oblast buněk*) nese své jedinečné jméno
- ***Formát tabulky*** – pokud na tabulku aplikujete formát, můžete využít odkazy na tabulku s formátem/stylem (*na první pohled vypadá složitě, ale opak je pravdou, styly pomohou zpřehlednit a zjednodušit vaše vzorce*). Tj. na tabulku se aplikuje styl (formát).

Poznámka: Musíte si vybrat buď použití stylu (způsobu) adresování A1, nebo R1C1. Použití definovaných názvů a na buňky, je nezávislé na volbě stylu adresování A1/R1C1.

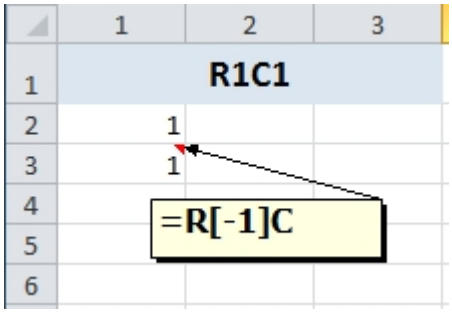
4.3.5.2. A1 – klasický

Nejčastěji používáno pro klasické výpočty, odkazy jsou na první pohled přehlednější. Při prvním spuštění Microsoft Excelu je tento styl odkazu přednastaven (pokud jej někdo nezmění, např. "zákeřný" kolega).



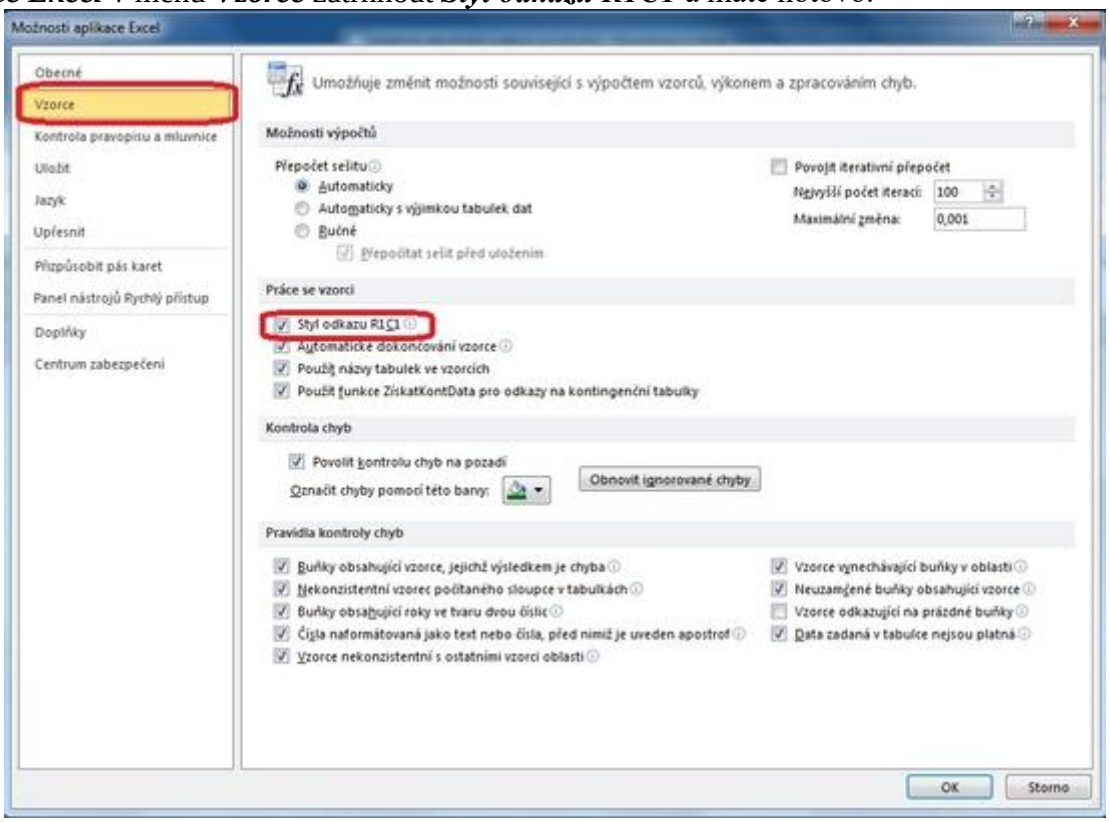
4.3.5.3. R1C1 – programátorský

Oblíbený hlavně programátory. Výhoda při programování ve VBA se nemusíte zabývat písmenky (zda jde o sloupec AA nebo AB?), postě přičtete patřičný počet sloupců.



4.3.5.4. Změnit styl A1 -> R1C1 a zpět R1C1 -> A1

Pro Excel 2013. Na kartě ***Soubor*** – Možnosti v zobrazeném dialogovém okně ***Možnosti aplikace Excel*** v menu ***Vzorce*** zatrhnout ***Styl odkazů R1C1*** a máte hotovo.

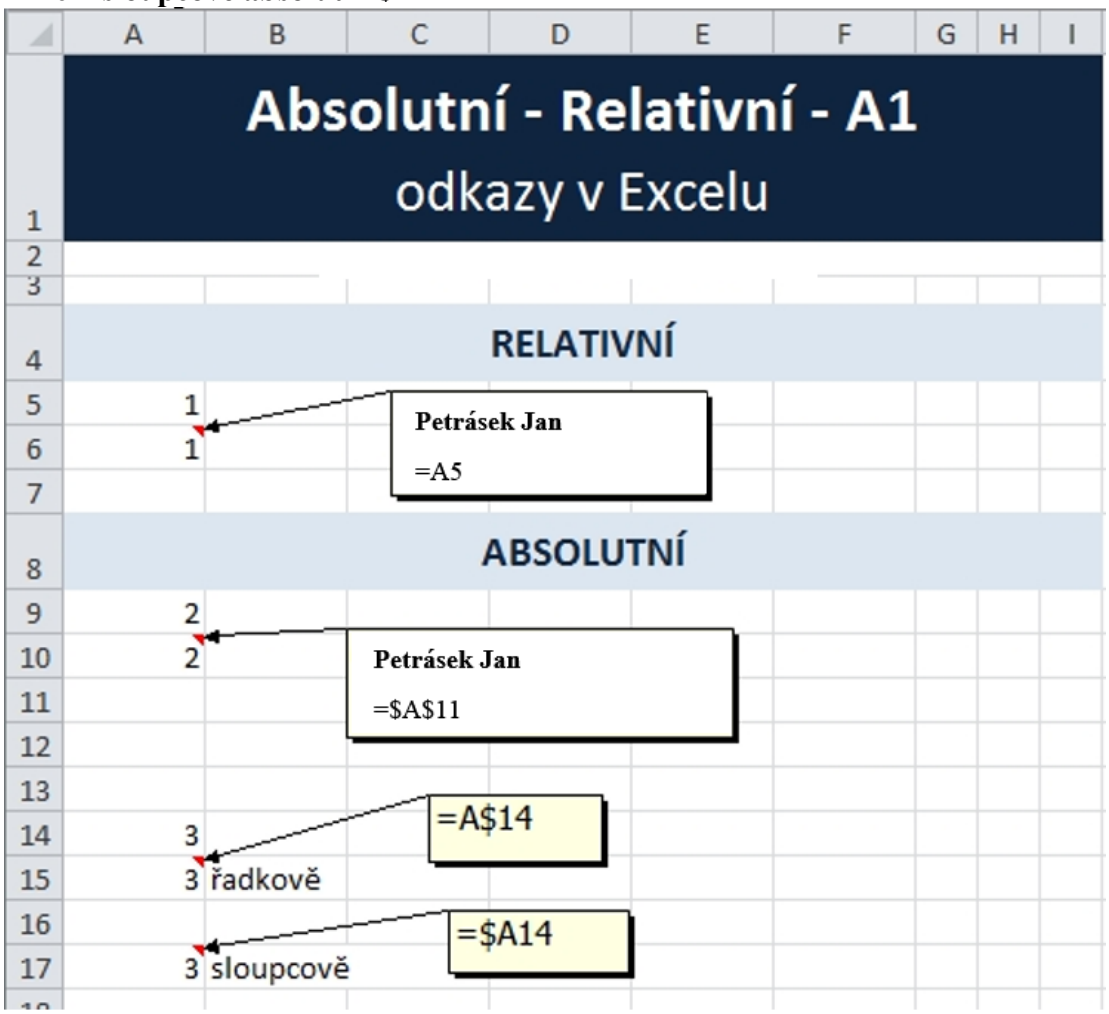


4.3.5.5. Absolutní relativní (smíšené) odkazy

Přepínaní mezi absolutním, relativním, řádkově absolutním a sloupcově absolutním odkazem lze pomocí klávesové zkratky ***F4***.

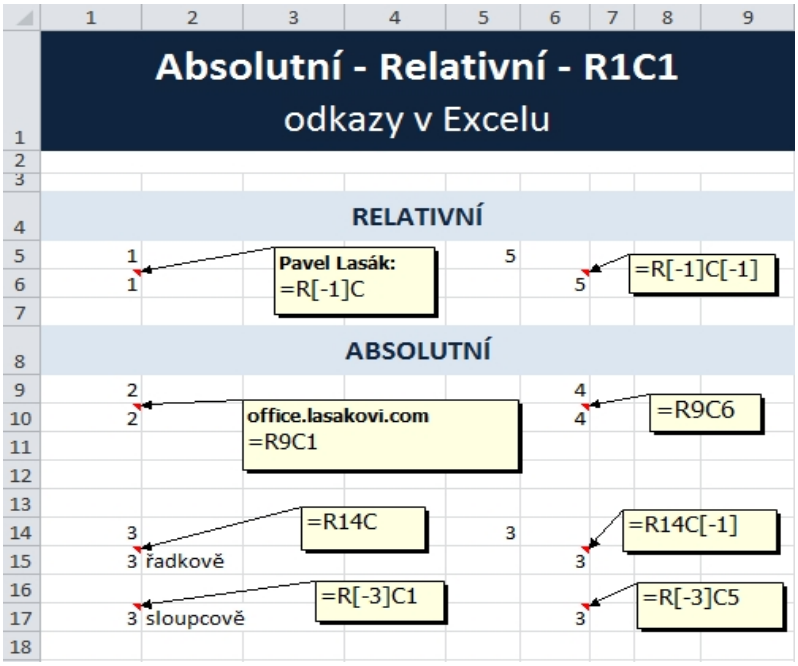
4.3.5.5.1. Styl A1

- **relativní A1** – při kopírování vzorce se dynamicky mění odkaz. Příklady v další kapitole.
- **absolutní $A$1** - při kopírování buňky nedochází je změně odkazu
- **smíšený** – tento odkaz může být buď:
  - **řádkově absolutní A$1**
  - **sloupcově absolutní $A1**



4.3.5.5.2. Styl R1C1

- **relativní R[-1]C[-1]** - pokud ve stejném sloupci řádku bez čísla
- **absolutní R9C6** – při kopírování buňky nedochází je změně odkazu
- **smíšený** – tento odkaz může být buď:
  - **řádkově absolutní R14C[-1]**
  - **sloupcově absolutní R[-1]C5**



Zdroj obrázku: Office.laskovi.com, autorem je Pavel Lasák

4.3.5.6. Pokročilejší odkazy

Odkazovat se nemusíte jen na buňku v daném listě, ale také:

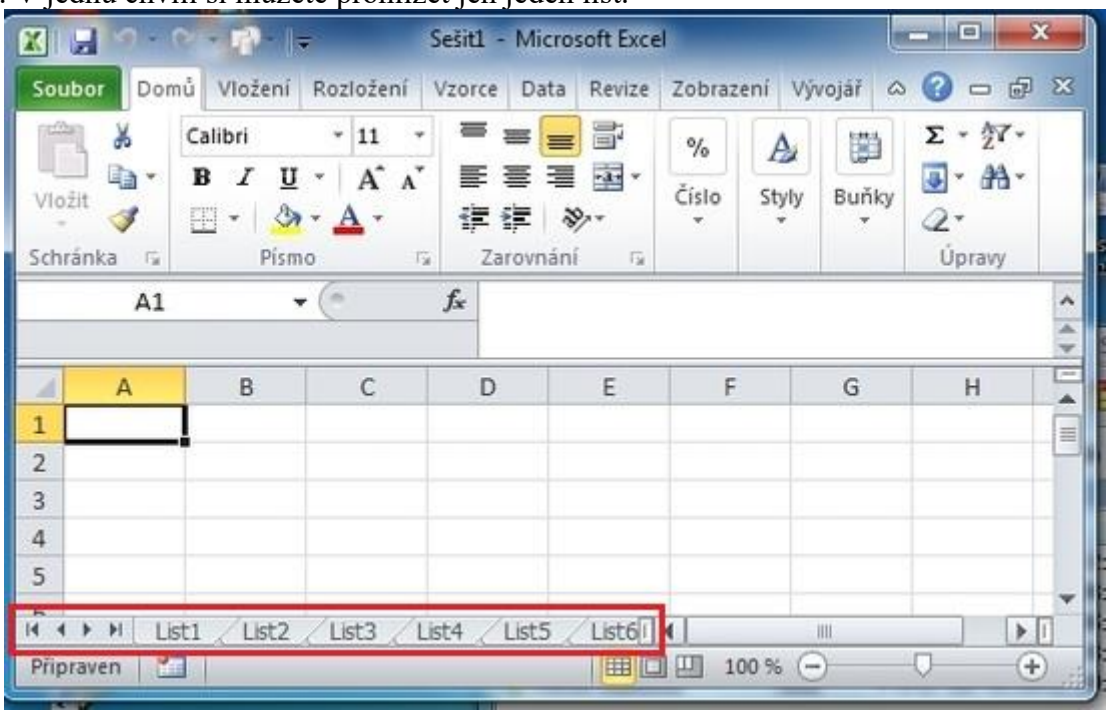
- ***=A1*** - odkaz na buňku ve stejném listu
- ***=List2!A1*** - odkaz na buňku v jiném listu
- ***=[priklad.xls]List1!A1*** - sešit musí být otevřen
- ***='[jiny nazev s mezerami.xls]List1'!A1*** - musí být uvedeno v uvozovkách
- ***='C:\Dokumenty_moje\Excel\[priklad.xls]List1'!A1*** správně nastaveno v možnostech, aby se provedlo automaticky načtení změněných hodnot
- ***=Excel.Sheet.12|'D:\smaz\excel.xlsx'!'!List 1!R12C5'*** - propojení
- ***=[@[MujSloupec]]*** - další způsob odkazů, pokud používáte styly tabulek kde MujSloupec je název sloupce. Podrobnější popis je uveden níže.

Pokud máte odkaz do jiného sešitu, musíte mít nastaveno automatické obnovování v možnostech. Jinak při otevření sešitu se nenačtou automaticky data z těchto externích sešitu.

4.3.6. Práce s listy

4.3.6.1. Výběr (volba) listu

Otevřením sešitu, se Vám zobrazí zároveň jednotlivé listy (v záložkách vidíte jejich jména). V jednu chvíli si můžete prohlížet jen jeden list.

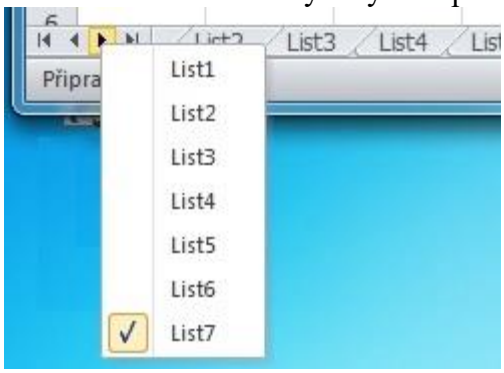


Pokud se chcete přesunout na druhý. Lze to provést klikem na příslušnou záložku, kdy se Vám zobrazí Vámi vybraný list. Pokud je listů více můžete se na další (předchozí dostat klikem na posuvník).



Posuvník umožní přesun na první list, na předchozí list, na další list a na poslední list.

Další možností jak zvolit požadovaný list, je klik pravým tlačítkem myší na jednu z šipek a zobrazí se Vám všechny listy k dispozici (pak si jen vhodný vyberete).



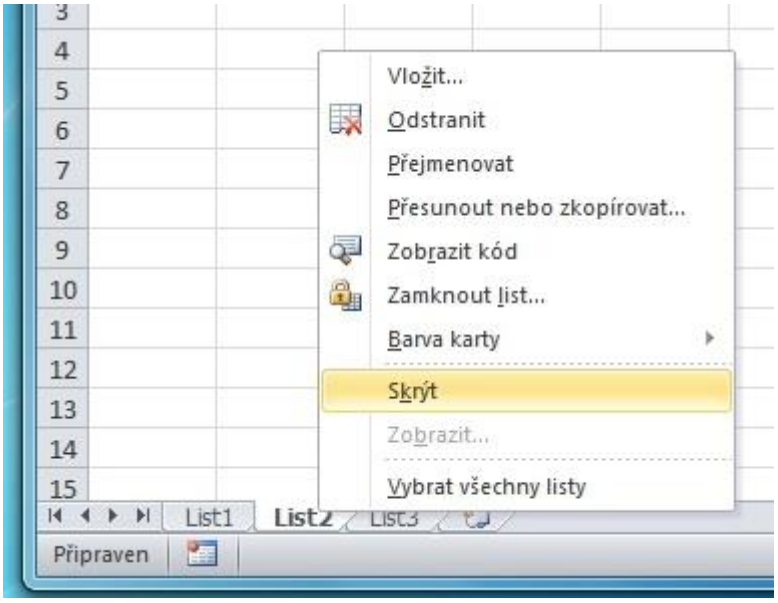
Poznámka: Minimálně jeden list je v každém sešitě viditelný.

4.3.6.2. Skrýt/zobrazit listy

V Excelu můžete skrývat/zobrazovat listy. Máte-li na jednom listě výpočty, které chcete mít na první pohled skryté. Máte dvě možnosti:

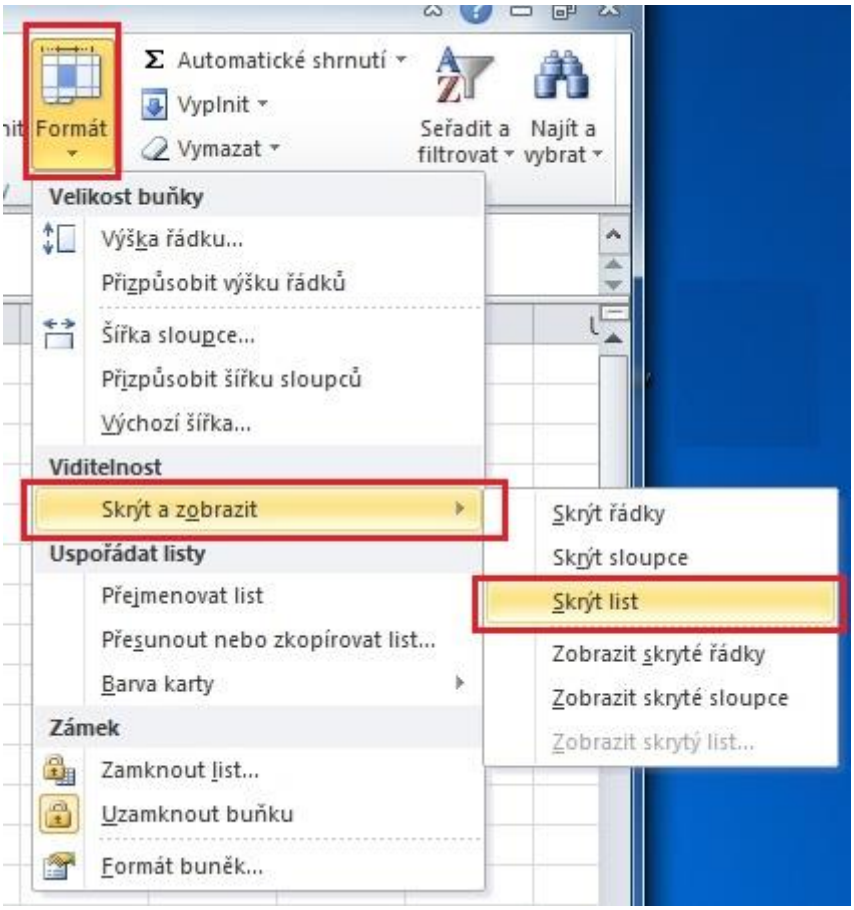
Klik na záložky

V záložkách listu kliknete pravým tlačítkem myší na záložku listu, který chcete skrýt a vyberete ***Skrýt list***



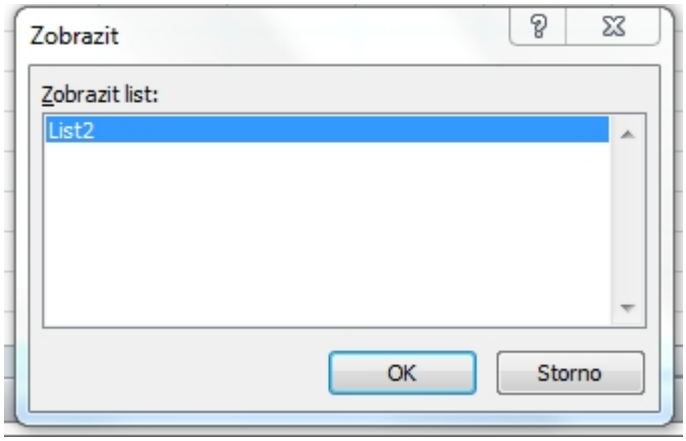
Skrýt list (využitím pásu karet)

V pásu karet ***Domů*** sekce ***Buňky*** ikona ***Formát*** z menu vyberete ***Skrýt a zobrazit – Skrýt list***



Zobrazit list

V pásu karet ***Domů*** sekce ***Buňky*** ikona ***Formát*** z menu vyberete ***Skrýt a zobrazit – Zobrazit skrytý list*** v zobrazeném dialogovém okně si vyberete list, který chcete zobrazit. Pokud je super skrytý list tak jej v přehledu neuvidíte.



Poznámka: Další možnost, jak skrýt list je super skrytý list.

Poznámka: Pokud sešit obsahuje jen jeden list (nelze již vybrat jiný list). Jinak minimálně jeden list musí být zobrazen. Pokud budete chtít odstranit poslední list, obdržíte chybovou hlášku.

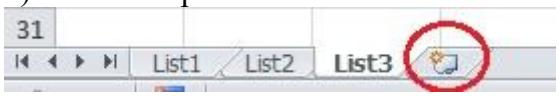


4.3.6.3. Vložit list

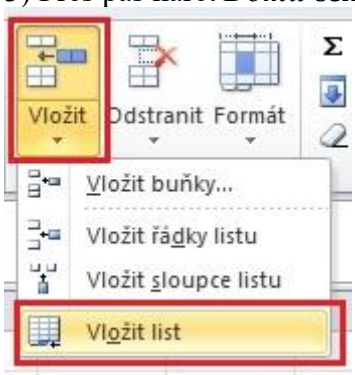
Jak vložit nový list. Pokud Vám jeden list, lze vložit nový list.

1) klávesovou zkratkou ***Shift + F11***

2) Klikem na přidávací tlačítko v liště s názvy listů.



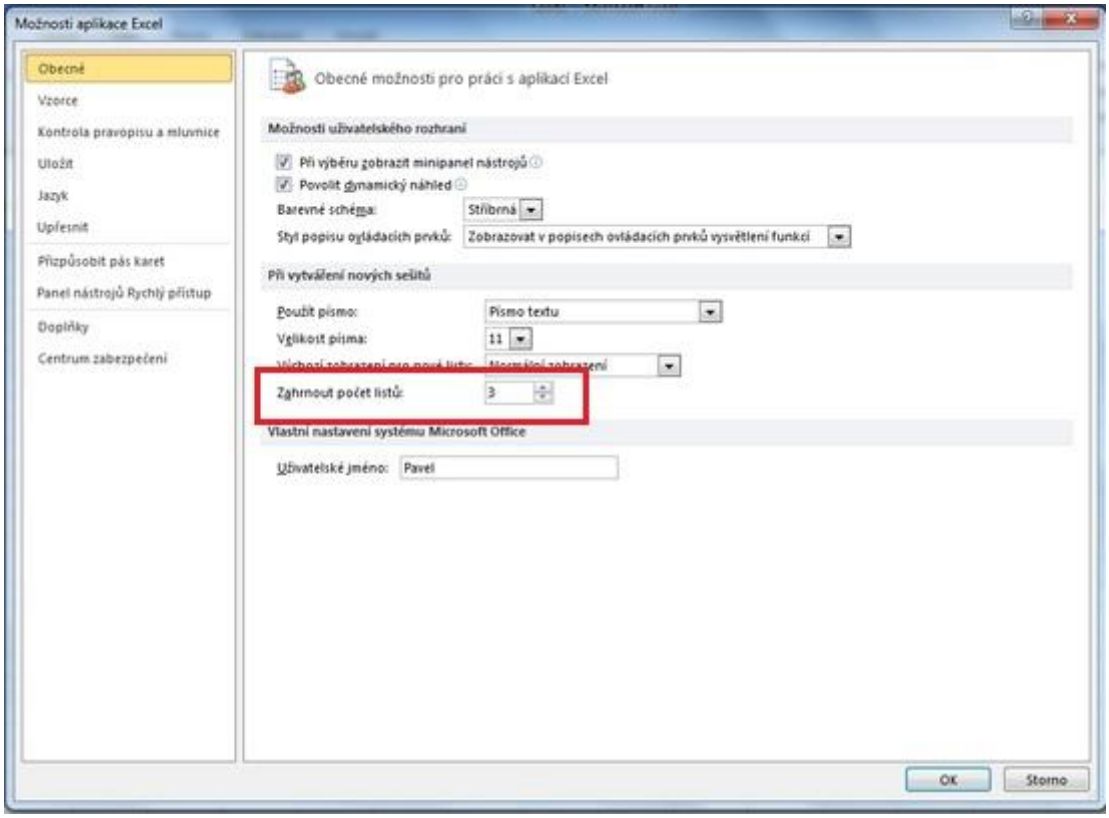
3) Přes pás karet ***Domů*** sekce ***Buňky*** ikona ***vložit – Vložit list***.



Poznámka: Po otevření sešitu v Excelu máte v sešitu jeden list (standartní nastavení Excelu, pokud si změníte, může jich být zobrazen i jiný počet). Toto můžete jednoduše změnit.

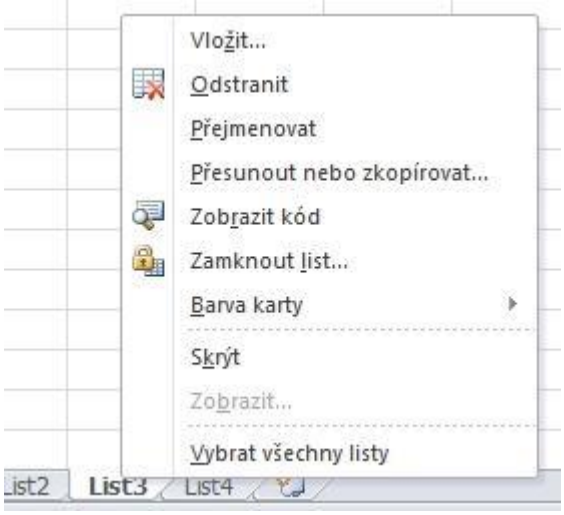
4.3.6.3.1. Změna počtu vložených listu při otevření MS Excelu.

Přes pás karet ***Soubor*** menu ***Možnosti*** v dialogovém oknu změníte. ***Zahrnout počet listů:***

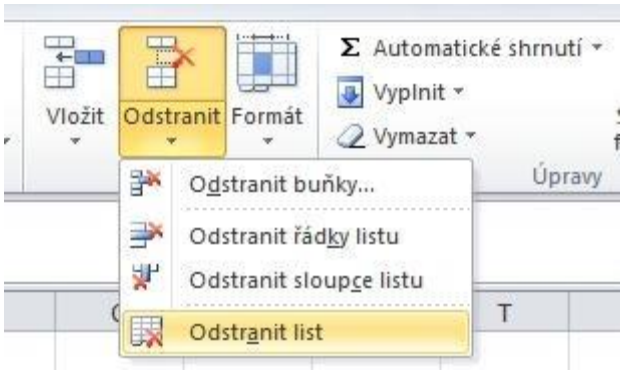


4.3.6.4. Odstranit / Smazat list

1) Klik pravým tlačítkem myší na záložce listu, který chcete odstranit (smazat) a vybrat ***Odstranit***. V případě že list obsahuje data se Excel zeptá, zda opravdu chcete odstranit. Pokud ano klik na ***Odstranit***.



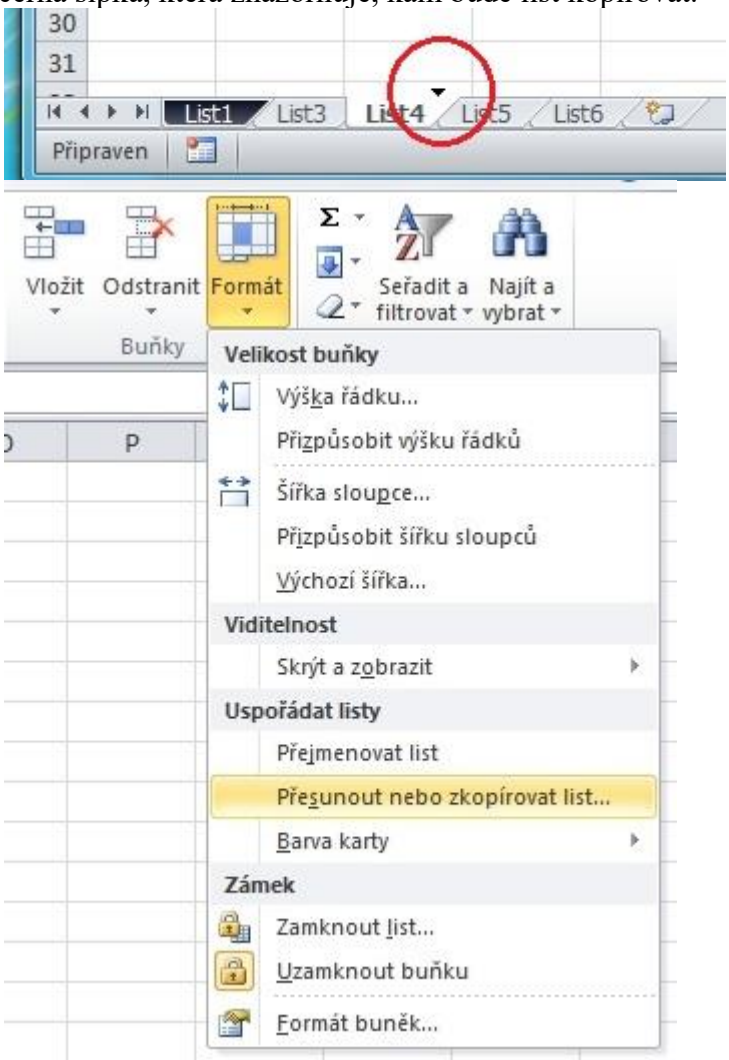
2) Přes pás karet ***Domů*** sekce ***Buňky*** ikona ***Odstranit – Odstranit list***. V případě že list obsahuje data se Excel zeptá, zda opravdu chcete odstranit. Pokud ano klik na ***Odstranit***.



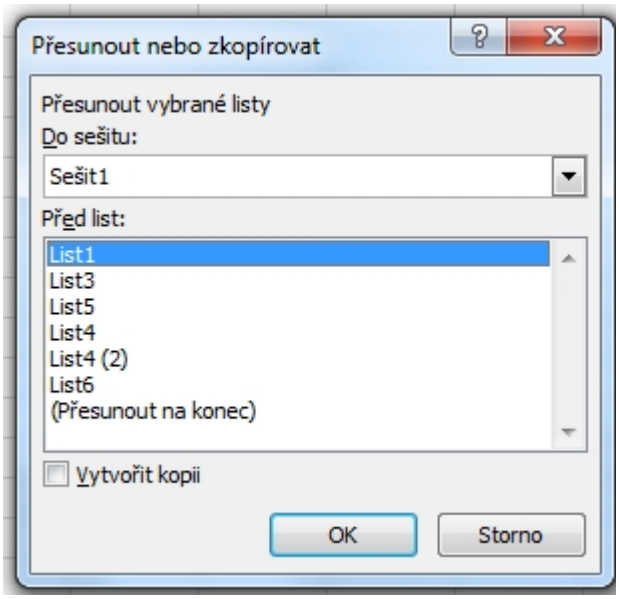
4.3.6.5. Kopírování listu

4.3.6.5.1. Klikem myší na záložku

Stisk levého tlačítka myši na zvolené záložce listu, který chcete zkopírovat a za současného stisku klávesy ***Ctrl*** přesunout na místo kam si přejete vložit kopie listu. U záložek se objeví černá šipka, která znázorňuje, kam bude list kopírovat.



V dialogovém okně vyplníte, kam chcete zkopírovat vybraný list(y). Nesmíte zapomenout **Zatrhnout** ***Vytvořit kopii***

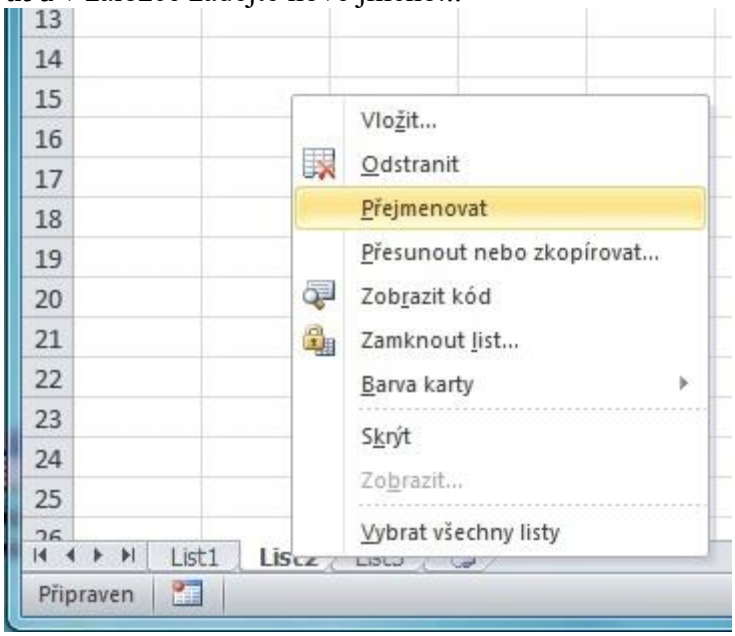


4.3.6.6. Přejmenovat list

Pokud se Vám nelíbí přednastavený název listu "List1" můžete si zvolit vlastní. Volba vlastního jména je vhodná, pokud využíváme více jak jeden list, usnadňuje to orientaci v sešitě a zvyšuje efektivitu práce.

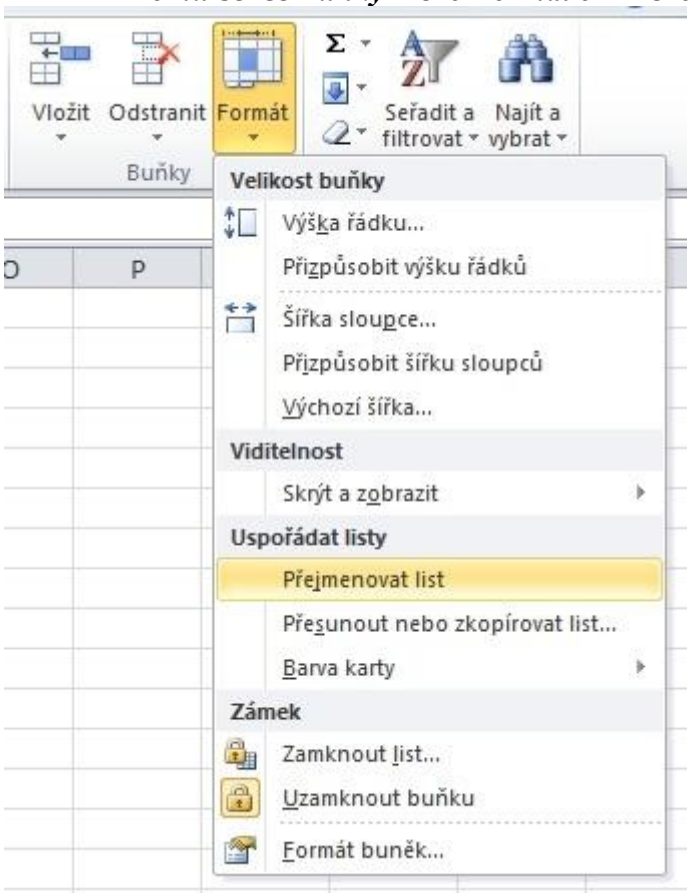
4.3.6.6.1. Klikem myší na záložku

Klik pravým tlačítkem myši na zvolené záložce a z kontextového menu vyberete ***Přejmenovat*** a v záložce zadejte nové jméno...



4.3.6.6.2. Využít pás karet ke změně názvu listu

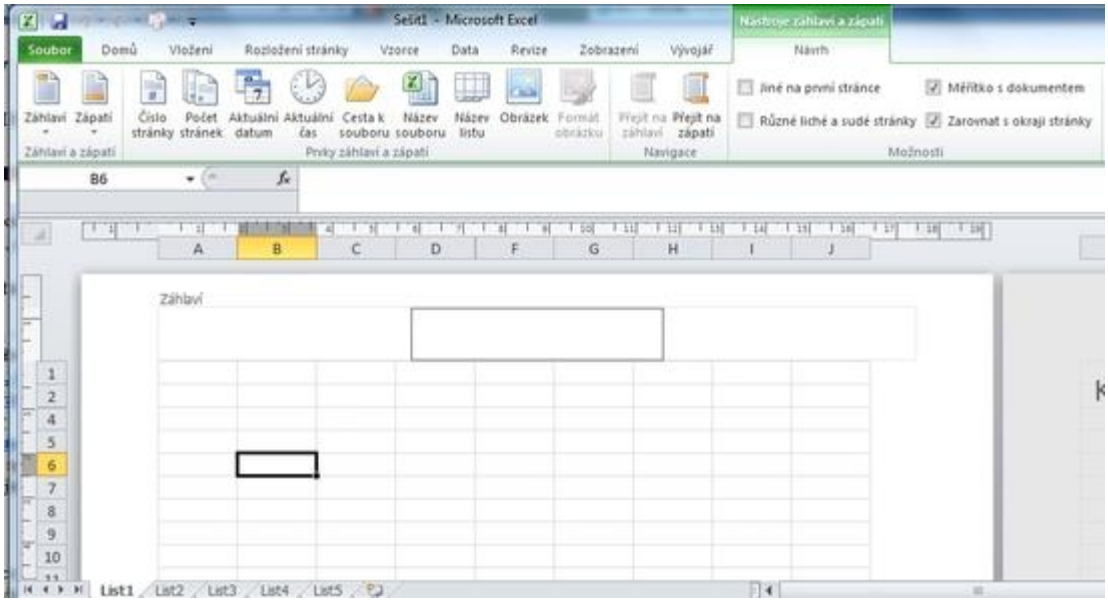
Pás karet ***Domů*** sekce ***Buňky*** ikona ***Formát*** a z menu vyberte ***Přejmenovat list***.



Poznámka: Každý název listu musí být jedinečný, tj. dva listy se nesmí jmenovat stejně.

4.3.6.7. Záhlaví a zápatí

Pás karet ***Vložení*** sekce ***Text*** ikona ***Záhlaví***. Zobrazí se Vám možnost vkládat záhlaví a zápatí.

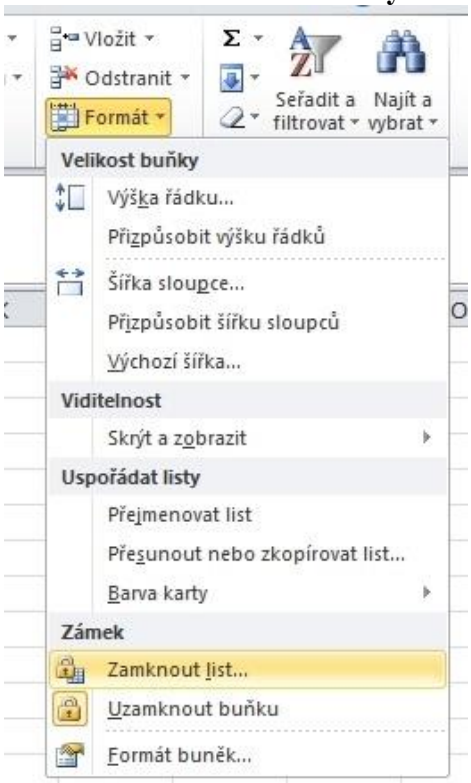


4.3.6.8. Zamykání buněk

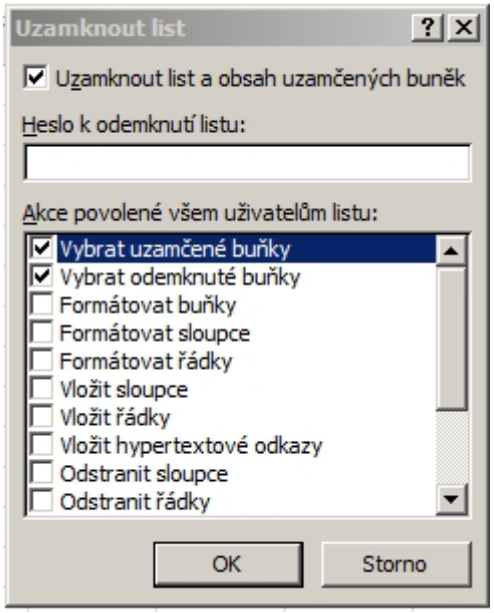
Další velice užitečné je využít zamykání buněk. Jako autor budete moci určit jen ty buňky, které půjdou přepisovat (měnit). Pokud si vytvoříte nějaký populární sešit (fakturu, docházkový list, ...) bude se vám hodit uzamčení buněk. To znamená, že buňky nepůjdou přepsat (až na těch pár, které necháte odemčeny). Nestane se vám, že nechtěně přepíšete nějaký vzorec, soubor uložíte, a příště v domnění, že v dané buňce stále vzorec budete s touto špatnou hodnotou dále počítat.

4.3.6.8.1. Zamknout list

Na kartě **Domů** - sekce **Buňky - Formát** vyberete **Zamknout list**.



V následně zobrazeném dialogovém okně **Uzamknout list** můžete nastavit Heslo (pokud chcete uživateli zamezit odemknutí). Případně dále upravit co uživatel může, nemůže s uzamčenými buňkami dělat.



Pozor: Heslo není nepřekonatelné, na internetu jsou k dispozici prográmky, které ho dokáži zjistit (odstranit). Přitom většina uživatelů si nedokáže odemknout sešit, který je bez hesla.

4.3.6.8.2. Změna zamčené buňky

Pokud je buňka zamčená a vy jí budete chtít změnit, obdržíte dialogové okno, které vás bude informovat o tom, že daná buňka je zamčená.

Buňky, které chci nechat měnit

Pro buňky, kterým chcete nechat možnost zápisu tak v dialogovém okně **Formát buněk** – karta **Zámek** (dialogové okno zobrazíte například klávesovou zkratkou Ctrl + Shift + F) musíte odškrtnout **Uzamknout**

Poznámka: Chápu, že pokud máte sešit uložený, jako šablonu nemusí vás trápit, pokud něco přepíšete. Jen co mám zkušenost, tak správně šablony téměř nikdo nepoužívá.

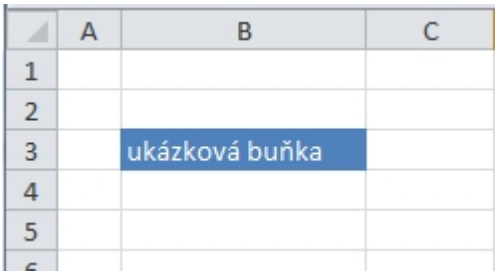
4.3.6.9. Styl buňky

Pokud pro jednu buňku nastavujete nějaké složitější formátování, můžete s výhodou použít styl buněk (místo kopírování formátu). Pod pojmem styl buňky si můžete přestavit nějakou sadu charakteristik (parametrů) buňky jako je například velikost písma, typ písma (kurzíva), barva písma, ohraničení buňky, zarovnání textu, formát čísla, zamykání...). Styl můžete nadefinovat také pro celou tabulku.

Začněme tím nejjednodušším, použijeme přednastavené styly, které jsou v Microsoft Excelu k dispozici. Nejprve vyberete buňku (buňky) kterým chcete přiřadit styl. Přiřazení stylu vybraným buňkám provedete na kartě ***Domů*** sekce ***styly*** ikona ***Styly buňky*** a vybrat požadovaný styl. V mojí ukázce ***Zvýraznění 1***

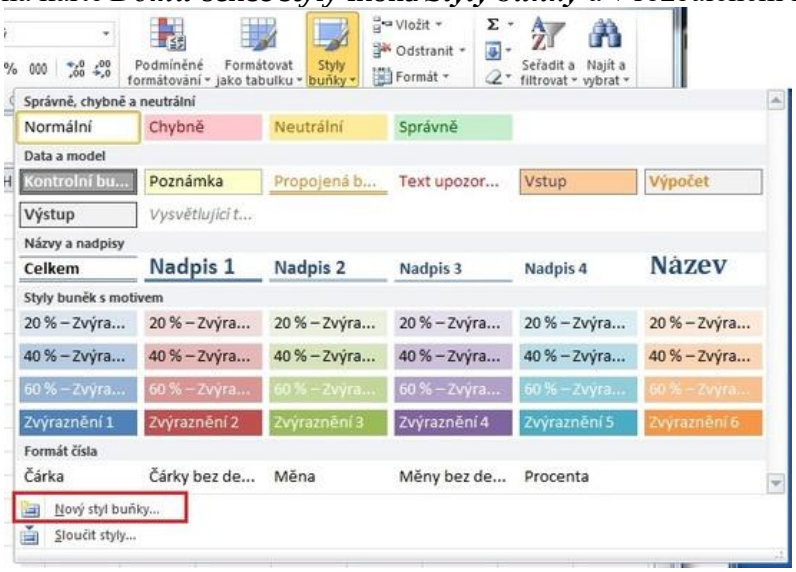


HOTOVO

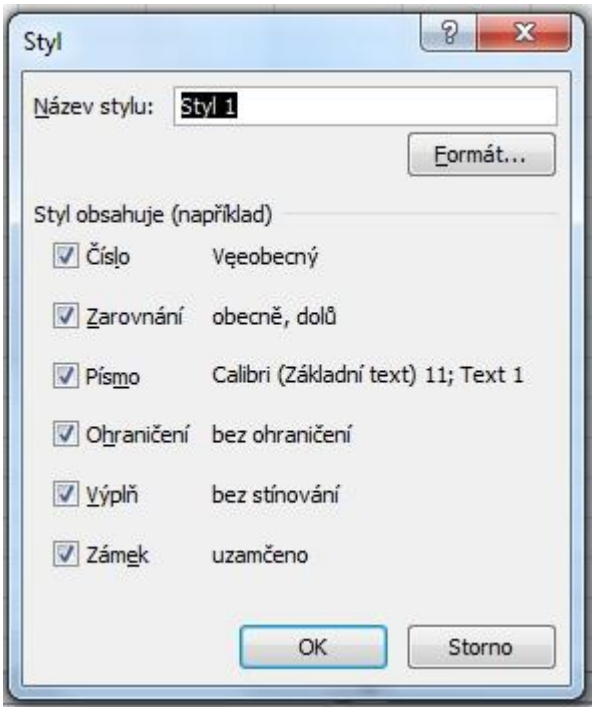


4.3.6.9.1. Vytvořit nový styl buňky

Vytvořit nový styl (přenesou se vlastnosti/parametry právě vybrané/aktivní buňky) provedete na kartě ***Domů*** sekce ***styly*** ikona ***Styly buňky*** a v rozbaleném menu ***Nový styly buňky***.



Obdržíte dialogové okno ***Styl***.



V zobrazeném okně ***Styl*** můžete vyplnit:

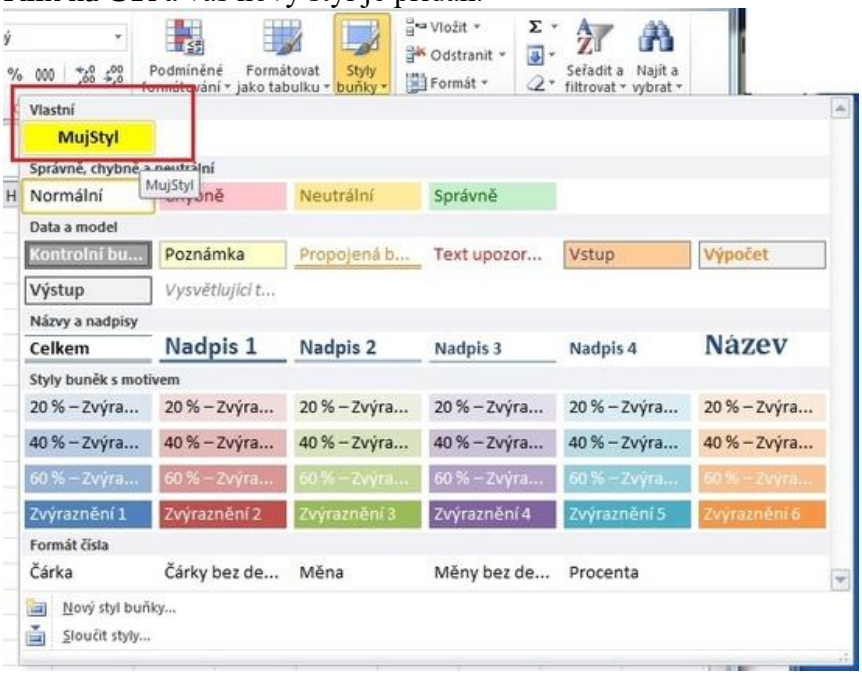
- **Název stylu**

Odškrtnout (tato sada parametru, nebude po aplikaci stylu změněna), která oblast parametrů nebude měněna. K dispozici jsou tyto:

- **Číslo**
- **Zarovnání**
- **Písmo**
- **Ohraničení**
- **Výplň**
- **Zámek**

Klik na **Formát...** můžete ještě dále doplnit, upravit formátování buňky.

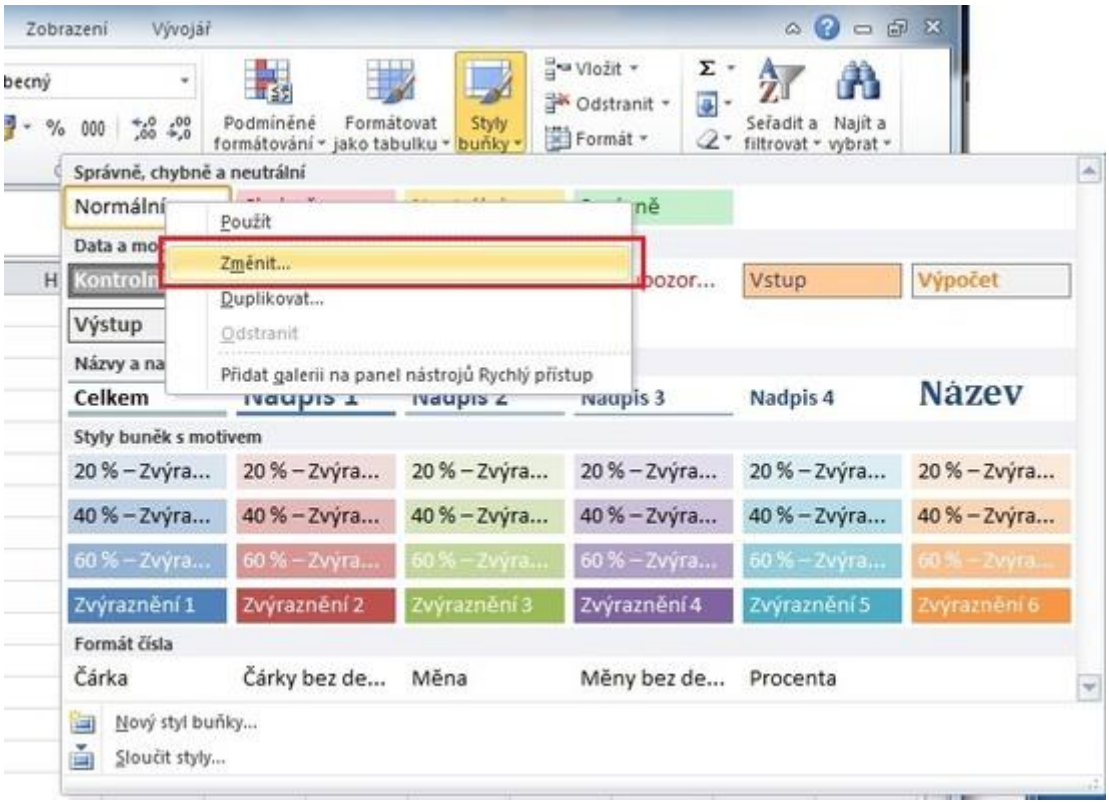
Klik na **OK** a vás nový styl je přidán.



Poznámka: Do nastavení se přenesou charakteristiky (parametry) právě vybrané buňky.

4.3.6.9.2. Změnit upravit styl

Chcete-li hotový styl buňky změnit (vylepšit), provedete opět na kartě ***Domů*** sekce ***styly*** ikona ***Styly buňky*** a v rozbaleném menu vyberete styl, který chcete změnit. Pravé tlačítko myši a z menu ***Změnit***.

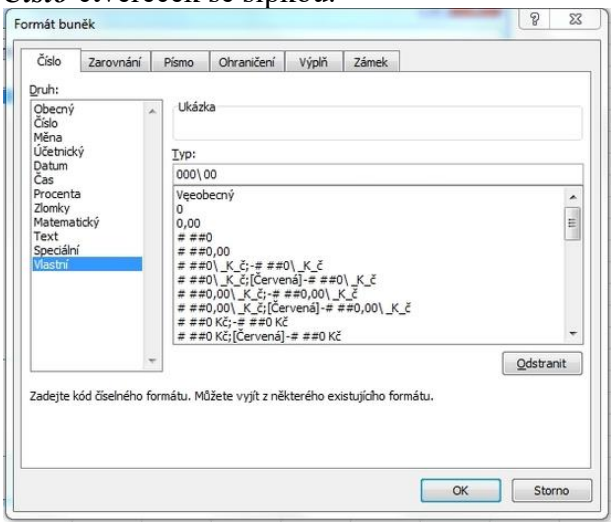


V zobrazeném dialogovém okně ***Styl*** můžete jako v předchozím případě upravovat formáty/parametry buňky.

4.3.6.10. Formát buňky

Dialogové okno ***Formát buněk*** záložka ***číslo*** je k dispozici:

- Pravé tlačítko myší – klik na buňce a poté ***Formát buněk***
- Karta ***Domů*** sekce ***Číslo*** čtvereček se šipkou.



4.3.6.10.1. Trocha teorie o vlastním formátu

- **Znaky** – které se ve vlastním formátu vyskytují (0, @, #, %, ...)
- **Podmínky**
- **Datum čas znaky**
- **Barvy** – jak se definují ve formátu a jaké názvy, nebo číselné označení barvy mají

Poznámka: Pozor na vzájemné kombinace s podmíněným formátováním a vlastních formátů (osobně pro grafiku používám raději podmíněné formátování).

Znaky

Pozor na verzi Excelu (Anglická, česká) - v anglické verzi se používají jiné znaky (čárka, je tečka. atd.), budeme se zabývat MS Excelem v české lokalizaci s českým národním nastavením.

- **@** - zástupný symbol textu v buňce @ "ukázka" @
- **0** - zástupný symbol čísla v buňce – zobrazuje i nuly (lze kombinovat 0,00 nebo 000,0 atd.)
- **#** - zástupný symbol čísla v buňce – Nezobrazuje nuly (# ### ##0)
- **%** - přidá procenta a hodnotu vynásobí 100
- **- + -** zobrazení plus mínus (není nijak vázáno s hodnotou buňky - doporučuji volit s rozmyslem)
- **$** - symbol měny (podobně jako Euro, lze zapsat přímo)
- ***** - musí být doplněna dalším znakem, který vyplní zbytek buňky
- **,** - desetinný oddělovat (pro anglickou verzi je to tečka)
- **""** - pokud obsahuje text tento bude zobrazen (splňuje-li kritéria)
- **.** - tečka oddělení dnů.měsíců.roků (pokud nejde o datum musí se doplnit uvozovkami ".")
- **:** - dvojtečka oddělení hodin:minut (pokud nejde o hodiny, musí se doplnit uvozovkami ":")
- **;** - oddělení sekcí/polí kladné číslo;záporné číslo;nula;text - vid dále barva a příklady
- **?** - otazník
- \ - zpětné lomítko – zobrazí následující znak (\@) - aby se daly zobrazit i znaky co mají specifickou vlastnost
- **[]** - hranaté závorky, lze využít jen tři intervaly (kdy třetí je pro zbývající hodnoty) viz dále
- **_** - dolní podtržítko – mezera
- **Vęeobecný** – Excel píše místo Všeobecný š jako ę

Podmínky

- **[podmínka]** - viz příklady, kde bude lépe pochopitelné
- **[název barva]** - viz další text o barvách
- **[barva číslo]** - viz další text o barvách

Datum čas znaky

Formát data

Pro formát data se používají:

- **d,dd,ddd,dddd**
- **m,mm,mmm,mmmm** měsíce jako římské číslice (I až XII) - problém v české verzi se zobrazením Římských číslic
- **rr,rrrr** - v anglickou (EN) verzi Excelu **yy, yyyy**
- **h,hh**
- **m,mm** - minuty bohužel mají stejný název jako měsíce
- **s,ss**

Případně využít další možnosti (složené závorky, ...)

- **h:mm:ss.00** - zlomky vteřin

- **[mm]** - uplynulý čas v hodinách
- **[h]** - uplynulý čas v minutách
- **h:mm dop./odp.** - zlomky vteřin

4.3.6.10.2. Problém se zobrazováním římských číslic

Excel 2007–2013 má problémy zobrazit římská čísla u měsíců. Proč tomu tak je se mi zatím nepodařilo zjistit. Jak tento problém obejít? Navrhuji použití funkce **ROMAN**:, nebo instalovat Excel 2016, který měsíce čísluje římskými čísly již automaticky.

4.3.6.10.3. Skloňování

I Excel umí skloňovat, jen ho to musíte naučit ;). Jak jsem psal lze využít jen tři intervaly. První omezení, druhé omezení a zbývající hodnoty.

Skloňování slova roky.

[=1]0" rok";[<5]0" roky";0" roků"

Podobně můžete skloňovat další slova (město, tobě sto, řediteli dvě sto :)).

4.3.7. Kontingenční tabulky a grafy

Poslední dobou se zvedá zájem o profesionálnější zpracování dat (jsou potřeba různé analýzy, zpracování a prezentace těchto dat...). Mnoho z tabulek je vytvořeno v Microsoft Excelu. Proč tedy nevyužít jeho možnosti? Například tvorbu kontingenčních tabulek a grafů. Rozhodl jsem se publikovat praktické příklady (a ušetřit si čas odpověďmi na stejné dotazy ;)).

4.3.7.1. Kontingenční tabulka

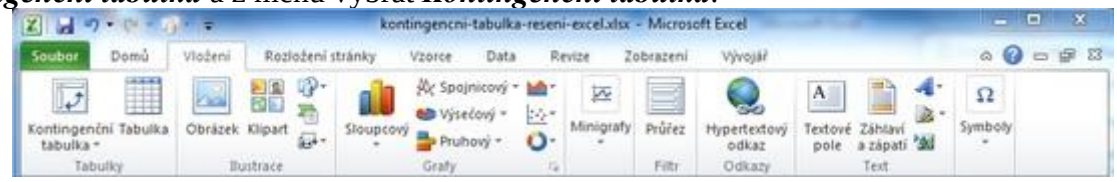
Hledáte-li v čem vytvořit interaktivní report, provést sumarizaci dat z rozsáhlých tabulek, nehledejte. Použijte **kontingenční tabulku**.

Kontingenční tabulka má sice divný název (pro někoho strašidelný), ale zdání klame, její používání je jednodušší, než si myslíte. Jakmile to jednou zkusíte, již bez ní nebudete moci pracovat (tvořit reporty). Pomocí **kontingenční tabulky** můžete data analyzovat, provádět souhrny, třídění, výpočty, tvořit kontingenční grafy. Prezentovaná data budou snáze pochopitelná, a nejen pro prezentování, ale také pro následné rozhodování.

Pro snazší pochopení následujícího je dobré mít zdrojovou tabulku dat, takže tady je:

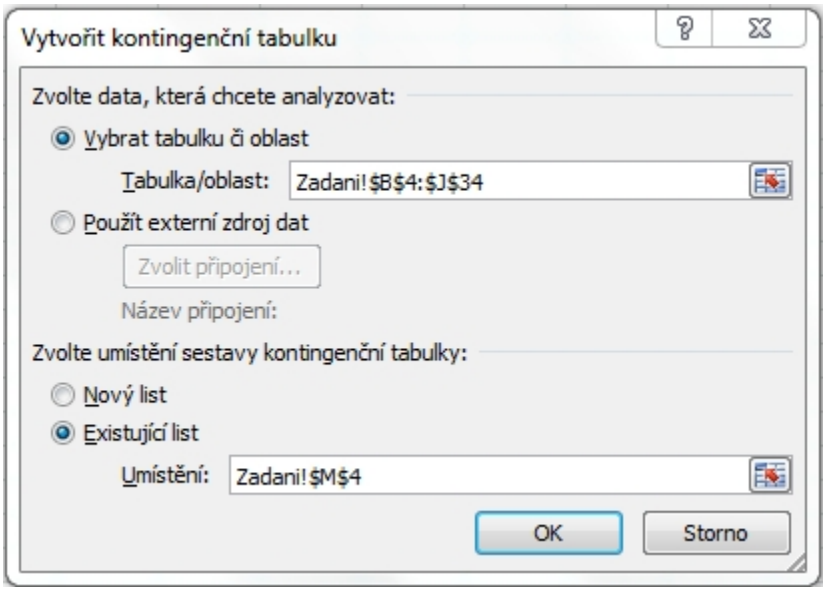
id	Prodejna	Prodejce	Kategorie	Produkt	Datum	Ks	Cena	Prodej
1	Krno	Fantomas	Hokej	Helma	1.2.2013	10	125	1 250
2	Krno	Superman	Golf	Hůl	1.2.2013	60	100	6 000
3	Krno	Rakosníček	Golf	Míček	5.2.2013	45	80	3 600
4	Krno	Bond James	Hokej	Chránče	6.2.2013	33	123	4 059
5	Krno	Wolker Johny	Golf	Boty	6.2.2013	44	456	20 064
6	Krno	Aladin	PinkPong	Lopta	6.2.2013	12	14	168
7	Krno	Bathman	PinkPong	Pálka	14.2.2013	5	147	735
8	Krno	Spiderman	Fotbal	Kopačky	14.2.2013	8	258	2 064
9	Krno	Fantomas	Horolezectví	Karabina	14.2.2013	24	369	8 856
10	Krno	Superman	Horolezectví	Sedák	14.2.2013	32	951	30 432
11	Krno	Rakosníček	Horolezectví	Osma	14.2.2013	24	159	3 816
12	Krno	Bond James	Hokej	Brusle	16.2.2013	20	7	140
13	Krno	Wolker Johny	Hokej	Trenér	16.2.2013	45	873	39 285
14	Krno	Aladin	Fotbal	Kopačák	16.2.2013	45	222	9 990
15	Krno	Bathman	Hokej	Puk	17.2.2013	45	357	16 065
16	Krno	Spiderman	Horolezectví	Lano	21.2.2013	12	12	144
17	Krno	Fantomas	Hokej	Síť	22.2.2013	1	24	24
18	Krno	Superman	Kuželky	Kuželka	22.2.2013	145	64	9 280
19	Krno	Rakosníček	Kuželky	Míč	26.2.2013	10	128	1 280
20	Krno	Bond James	Golf	Míček	3.3.2013	20	1024	20 480
21	Krno	Wolker Johny	Kuželky	Leštidlo	4.3.2013	30	555	16 650
22	Krno	Aladin	Golf	Příručka	5.3.2013	12	666	7 992
23	Krno	Bathman	Horolezectví	Skoba	6.3.2013	15	777	11 655
24	Krno	Spiderman	Fotbal	Rozhočí	7.3.2013	18	888	15 984
25	Krno	Fantomas	Golf	Bagl	8.3.2013	6	999	5 994
26	Krno	Superman	Fotbal	Dres	8.3.2013	12	200	2 400
27	Krno	Rakosníček	Hokej	Chraniče	9.3.2013	49	300	14 700
28	Krno	Bond James	Hokej	Vstupenky	10.3.2013	51	400	20 400
29	Krno	Wolker Johny	Hokej	Brusle	10.3.2013	14	222	3 108
30	Krno	Aladin	PinkPong	Manuál	10.3.2013	87	111	9 657

Ve zdrojové tabulce si vyberte nějakou buňku Na kartě ***Vložení*** sekce ***Tabulky*** sekce ***Kontingenční tabulka*** a z menu vybrat ***Kontingenční tabulka***.



Vytvořit kontingenční tabulku

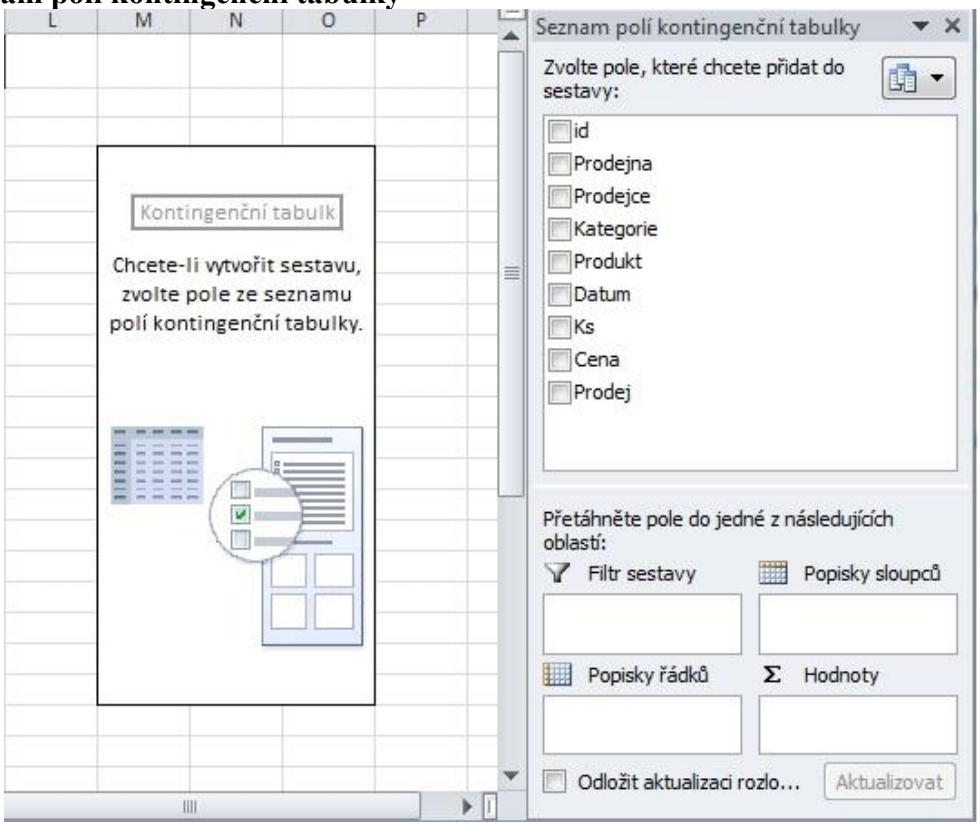
Obdržíte dialogové okno ***Vytvořit kontingenční tabulku***.



V dialogovém okně ***Vytvořit kontingenční tabulku*** nastavíte (něco je nastaveno automaticky)

- **Zvolte data, která chcete analyzovat:** - je zaškrtnuto *Vybrat tabulku či oblast* – toto je přednastaveno automaticky
- **Zvolte umístění sestavy kontingenční tabulky:** - pro přehlednost zaškrtněte *Existující list* a do **Umístění:** zadejte horní levý roh pro kontingenční tabulku *Zadani!$M$4*
- **OK** – potvrdit

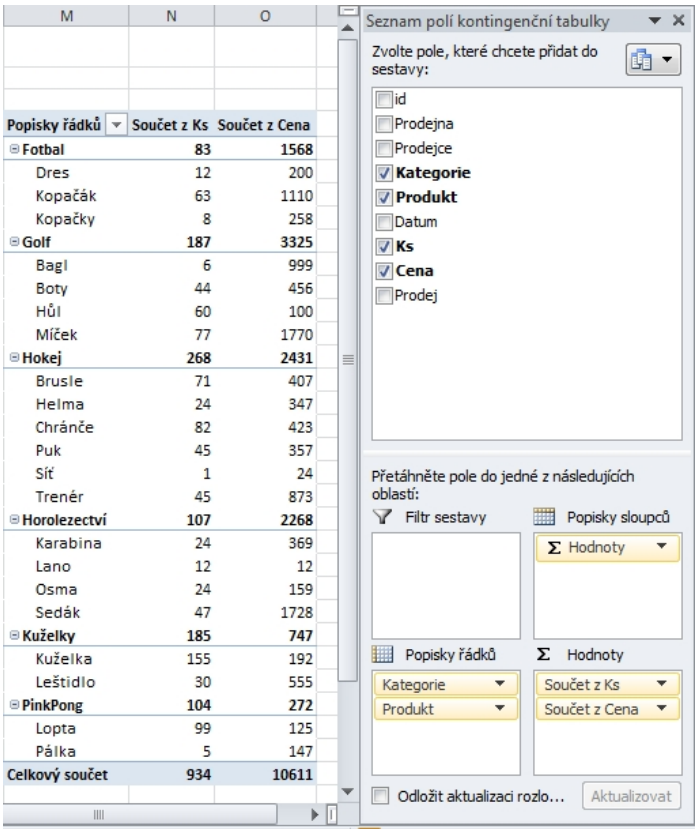
Seznam polí kontingenční tabulky



V seznam polí kontingenční tabulky stačí zaškrtnout tato pole:

- **Kategorie**
- **Produkt**
- **Ks**
- **Cena**

Zaškrtnutím se automaticky tvoří kontingenční tabulka (průběžně jak zaškrtáváte, jde o interaktivní tvorbu)



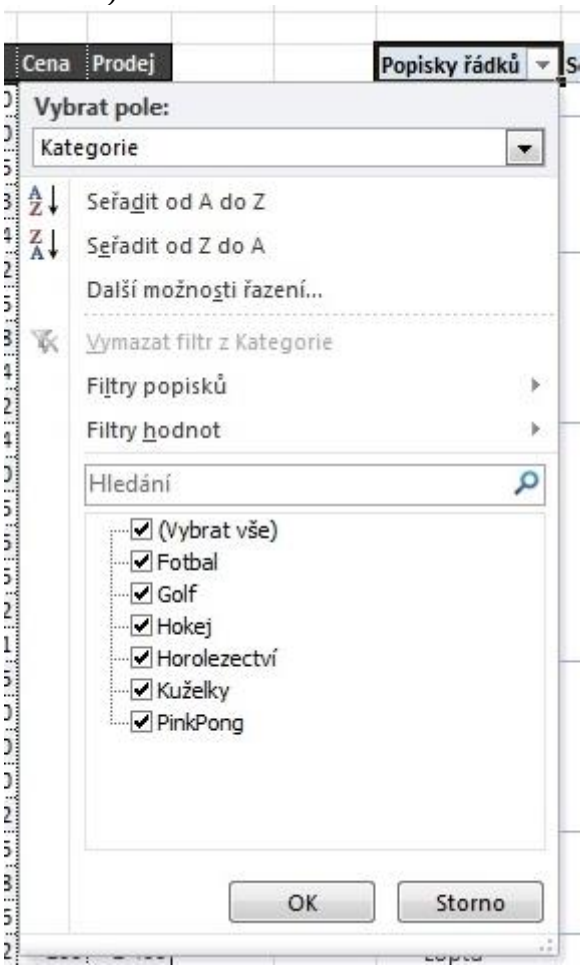
Kontingenční tabulka – hotovo

Kontingenční tabulku máte hotovou.

Popisky řádků	Součet z Ks	Součet z Cena
Fotbal	**83**	**1568**
Dres	12	200
Kopačák	63	1110
Kopačky	8	258
Golf	**187**	**3325**
Bagl	6	999
Boty	44	456
Hůl	60	100
Míček	77	1770
Hokej	**268**	**2431**
Brusle	71	407
Helma	24	347
Chránče	82	423
Puk	45	357
Síť	1	24
Trenér	45	873
Horolezectví	**107**	**2268**
Karabina	24	369
Lano	12	12
Osma	24	159
Sedák	47	1728
Kuželky	**185**	**747**
Kuželka	155	192
Leštidlo	30	555
PinkPong	**104**	**272**
Lopta	99	125
Pálka	5	147
Celkový součet	**934**	**10611**

4.3.7.1.1. Trochu úprav kontingenční tabulky

Protože kontingenční tabulka bude potřeba pro další články, ještě provedeme filtr (ať není tak dlouhá).



V kontingenční tabulce klik na ***Popisky řádků*** a zaškrtnete jen ***Horolezectví*** kontingenční tabulka se vyfiltruje:

Popisky řádků	Součet z Ks	Součet z Cena
⊟Horolezectví	107	2268
Karabina	24	369
Lano	12	12
Osma	24	159
Sedák	47	1728
Celkový součet	107	2268

Dále můžete do kontingenční tabulky přidat další sloupec v **Seznam polí kontingenční tabulky** zaškrtnete *Prodej* kontingenční tabulka se rozšíří o další sloupce.

Popisky řádků	Součet z Ks	Součet z Cena	Součet z Prodej
⊟Horolezectví	107	2268	54903
Karabina	24	369	8856
Lano	12	12	144
Osma	24	159	3816
Sedák	47	1728	42087
Celkový součet	107	2268	54903



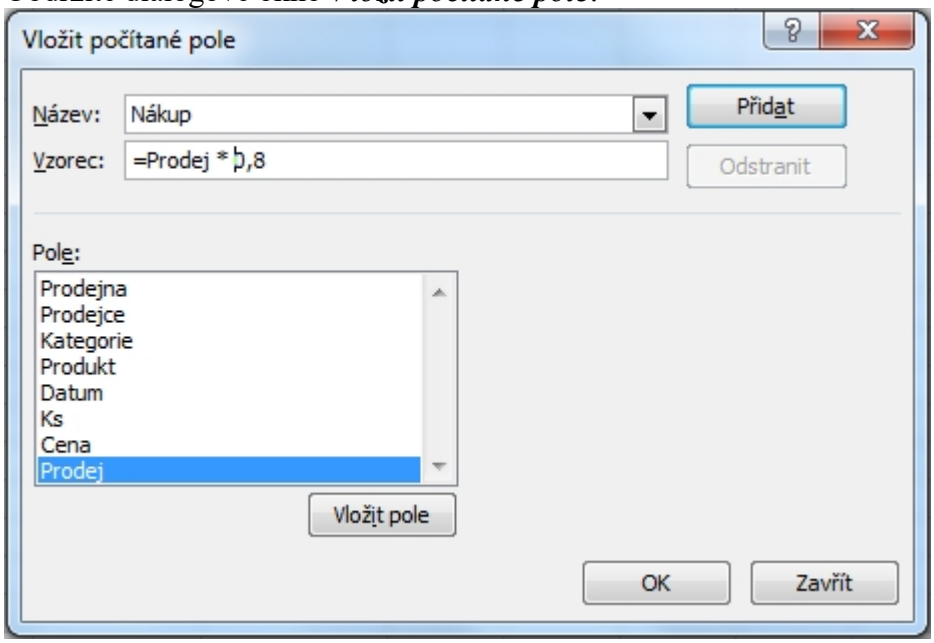
4.3.7.2. Vytvoření výpočtového pole

Cílem je přidat pole Nákup *Prodej * 0,8* a zisk *Prodej – Nákup*.

Na dočasné kartě ***Nástroje kontingenční tabulky*** karta ***Možnosti*** sekce ***Výpočty*** vybrat ***Pole, položky, sady*** a z menu ***Počítané pole***.



Obdržíte dialogové okno ***Vložit počítané pole***.



V tomto okně nastavíte:

- **Název** – místo přednastaveného popisu zadáte ***Nákup***
- **Vzorec** V **Pole:** vybrat *Prodej* klik na **Vložit pole** a k **Vzorec** přidat ** 0,8*
- **Přidat** – pole se automaticky přidá (tlačítku se změní popis z **přidat** na **změnit**)
- **OK** – pokud již nechcete přidávat další pole, stačí jen klik na OK

  Obdobně přidáte Pole **Zisk**.

  Kontingenční tabulka bude doplněna o vaše **počítané pole**
- **Nákup**
- **Zisk**

Popisky řádků	Součet z Ks	Součet z Cena	Součet z Prodej	Součet z Nákup	Součet z Zisk
⊟Horolezectví	107	2268	54903	43 922	10 981
Karabina	24	369	8856	7 085	1 771
Lano	12	12	144	115	29
Osma	24	159	3816	3 053	763
Sedák	32	951	30432	24 346	6 086
Skoba	15	777	11655	9 324	2 331
Celkový součet	107	2268	54903	43 922	10 981

4.3.7.2.1. Změnit/odstranit počítané (výpočtové) pole

Počítané pole již umíte vložit. Jeho změna/odstranění je stejně jednoduché.

Na dočasné kartě ***Nástroje kontingenční tabulky*** karta ***Možnosti*** sekce ***Výpočty*** vybrat ***Pole, položky, sady*** a z menu ***Počítané pole***.

Obdržíte dialogové okno ***Vložit počítané pole***, kde v název vyberete počítané pole, které chcete změnit/odstranit a provedete změnu/odstranění.

4.3.7.2.2. Seznam vzorců

Máte-li v kontingenční více vzorců – je vhodné mít v nich přehled. Tuto akci opět můžete provést automaticky. Na dočasné kartě ***Nástroje kontingenční tabulky*** karta ***Možnosti*** sekce ***Výpočty*** vybrat ***Pole, položky, sady*** a z menu ***Seznam vzorců***.

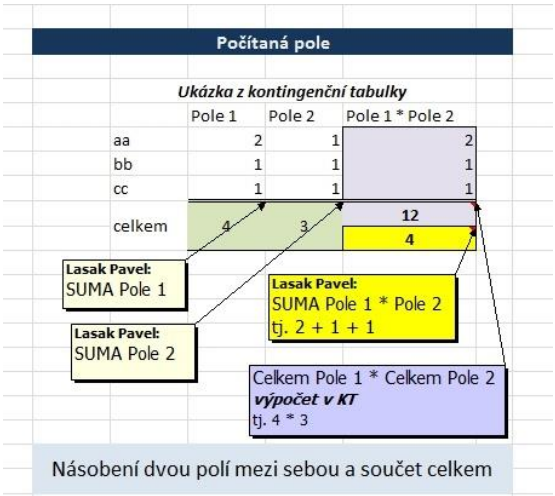
Na dalším listu obdržíte přehled vzorců (funkcí) ve vašich Počítaných polích.

	A	B	C	D	E	F
1	***Počítané pole***					
2	**Pořadí řešení**	**Pole**	**Vzorec**			
3	1	Nákup	=Prodej* 0,8			
4	2	Zisk	=Prodej-Nákup			
5						
6	***Počítaná položka***					
7	**Pořadí řešení**	**Položka**	**Vzorec**			
8						

4.3.7.2.3. Násobení dvou polí

Budete-li v počítaném poli násobit dvě pole ze zdrojové tabulky, možná se vám bude zdát, že Excel počítá nesprávně. Jenže počítá správně, násobí pole * pole, což provádí i pro pole součtová (takže v součtových polích nejde o součet, ale stále o násobení těchto dvou součtových polí ze základní tabulky).

Možná lépe pochopíte na ukázce:



Zdroj Office.laskovi.com, auterm je Pavel Lašák

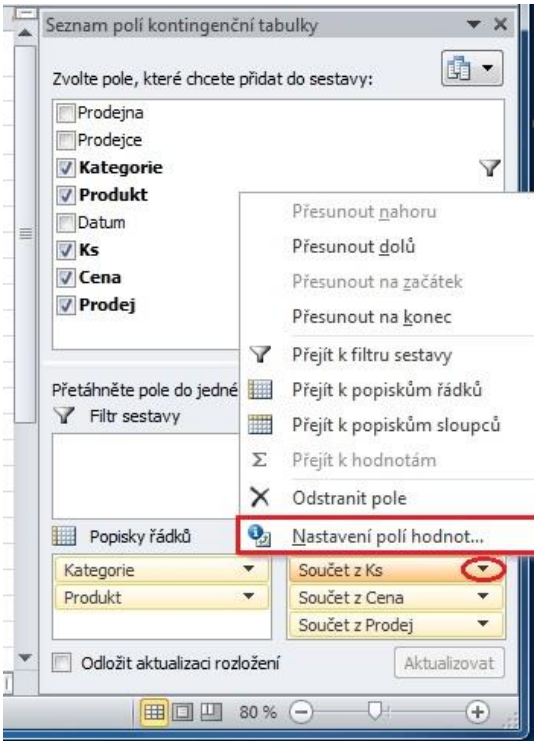
Jak z toho ven?

Například provést toto vynásobení ve zdrojové tabulce a pak se na tento sloupec jen odkázat.

4.3.7.3. Nastavení polí hodnot v kontingenční tabulce

Poznámka: Data v kontingenční tabulce jsou z důvodů přehlednosti filtrovaná.

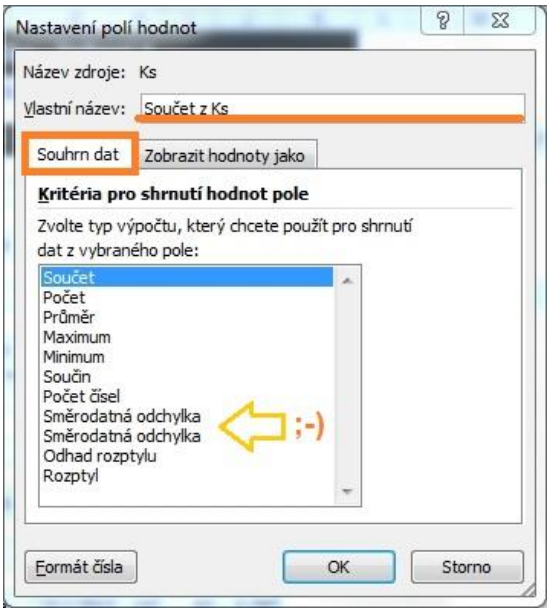
V **Seznam polí kontingenční tabulky** do pole **Hodnoty** (*klik na šipku*) z menu vybrat ***Nastavení polí hodnot ...***



V zobrazeném dialogovém okně ***Nastavení polí hodnot*** Záložka ***Souhrn dat*** lze zvolit kritéria pro shrnutí polí:

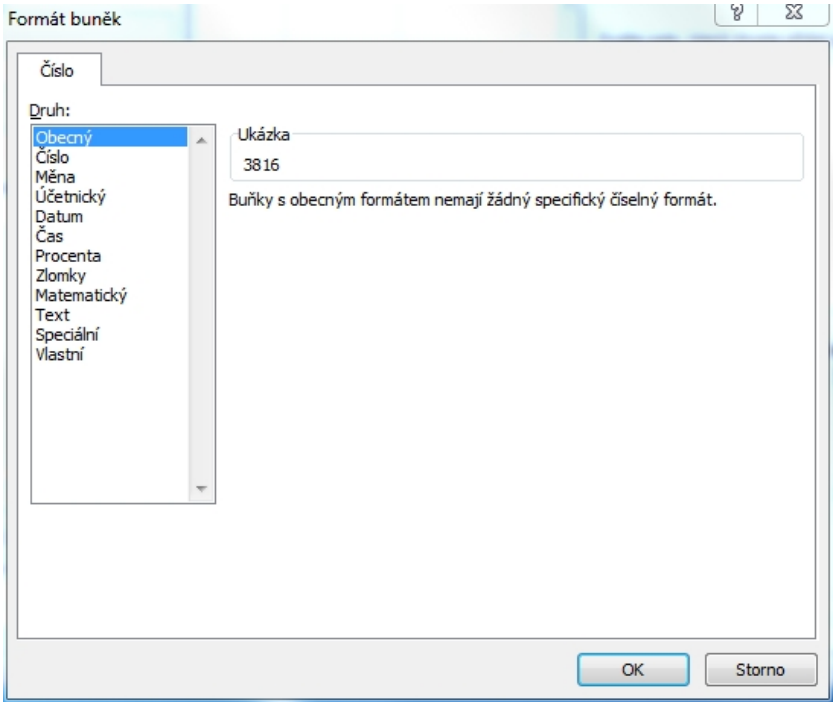
- **Součet**
- **Počet**
- **Průměr**
- **...**

Dále můžete změnit: **Vlastní název**.



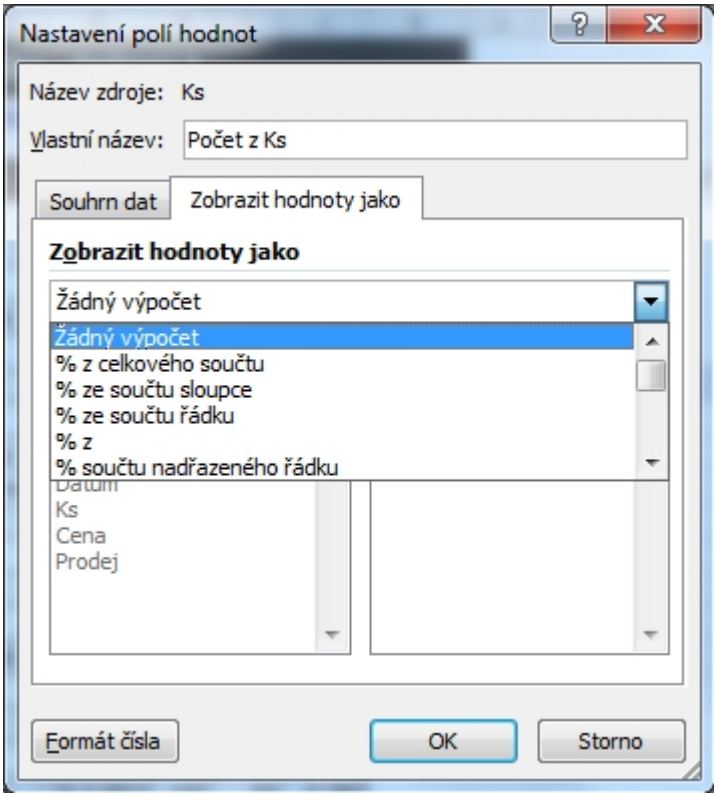
Poznámka: *Směrodatná odchylka* je tam opravdu 2x, ale počítají rozdílně).

Tlačítko **Formát čísla** umožňuje nastavit jak jinak formát čísla.



Záložka **Zobrazit hodnoty jako** umožňuje nastavit zobrazení hodnot jako:

- Žádný výpočet
- % z celkového součtu
- % ze součtu sloupce
- % ze součtu řádků
- % z
- ....



V ukázkovém příklad nastavit

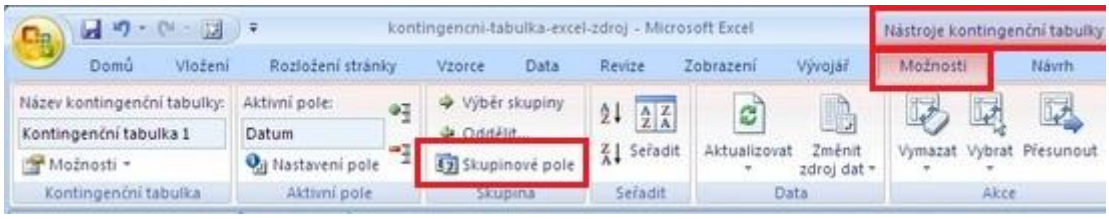
Pro ***Počet z ks*** nastavte *Počet*

Pro ***Součet z ks*** nastavte *% ze součtu sloupce*
Výsledek:

Popisky řádků	Počet z Ks	Součet z Cena	Součet z Prodej
⊟Horolezectví	5	100,00%	54903
Karabina	1	16,27%	8856
Lano	1	0,53%	144
Osma	1	7,01%	3816
Sedák	2	76,19%	42087
Celkový součet	**5**	**100,00%**	**54903**

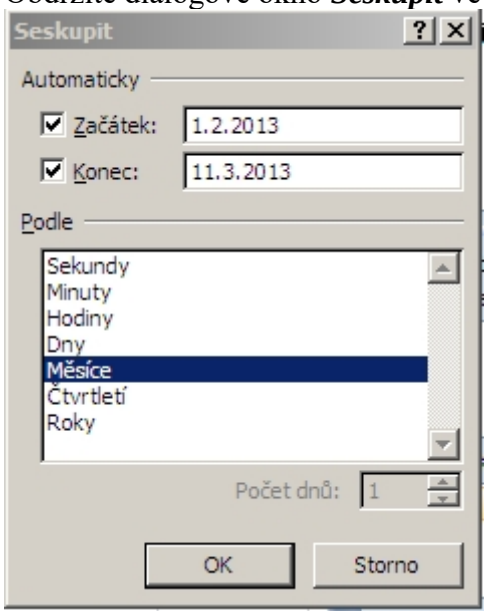
4.3.7.4. Seskupení dat – měsíčně

Klik na některý z datumů v popisu řádku (*potřebná ikona se stane přístupnou*). Na kartě ***Nástroje kontingenční tabulky – Možnosti*** sekce ***Skupina*** ikona ***Skupinové pole***. Obdržíte dialogové okno.



Dialogové okno – Seskupit

Obdržíte dialogové okno ***Seskupit*** ve kterém zatrhnete ***Měsíce*** a potvrdit ***OK***.



Kontingenční tabulka – měsíce - hotovo

V kontingenční tabulce máte popisy řádku seskupené na měsíce.

	Hodnoty		
Popisky řádků	Součet z Cena	Součet z Ks	Součet z Prodej
2	4469	620	157252
3	6142	314	129020
Celkový součet	10611	934	286272

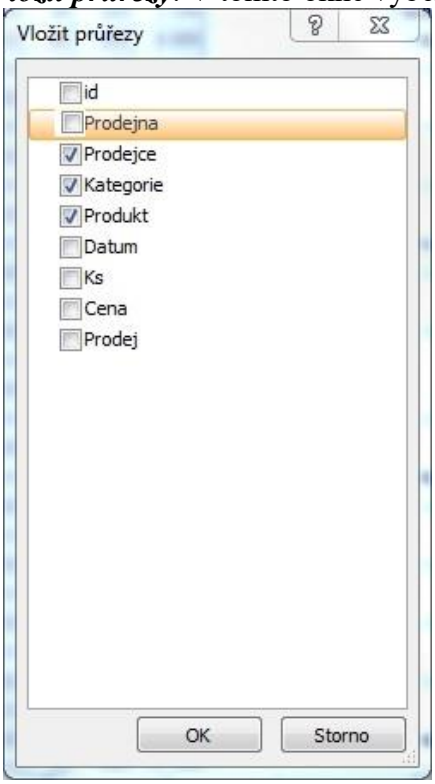
4.3.7.5. Vložení průřezu (Slicers) do kontingenční tabulky

Začínáme Vyběrem kontingenční tabulky.

Na dočasné kartě ***Nástroje kontingenční tabulky – Možnosti*** sekce ***Seřadit a Filtrovat*** ikona ***Vložit průřez*** a z menu vybrat ***Vložit průřez ...***.

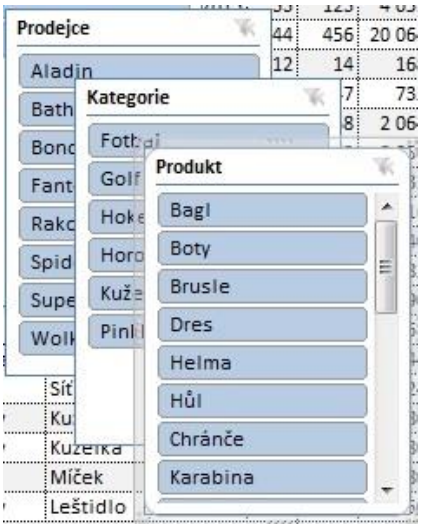


Obdržíte dialogové okno ***Vložit průřezy***. V tomto okně vyberte:



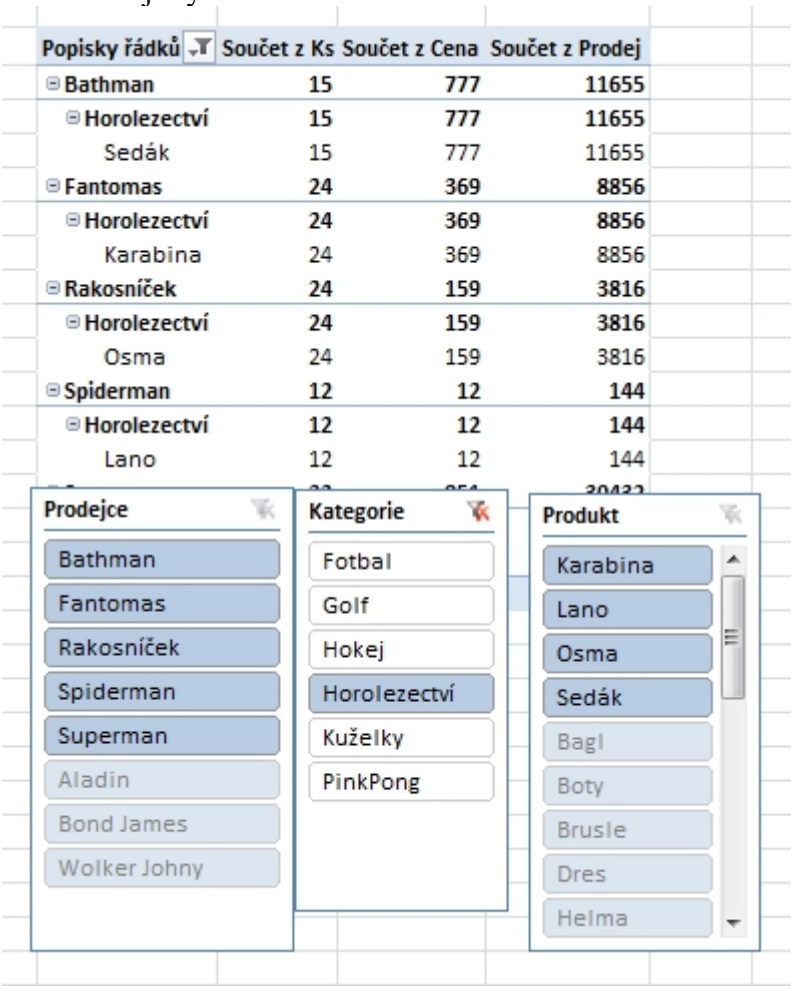
- Prodejce
- Kategorie
- Produkt

Klik na ***OK*** potvrdíte.



Excel automaticky vloží dané kategorie v samostatných oknech.

V jednotlivých dialogových oknech průřezu vyberte požadovaný filtr např. ***Horolezectví***. Dojde ke změně kontingenční tabulky a zároveň vidíte, které položky jsou k dispozici, které nejsou, které se dají vybrat.

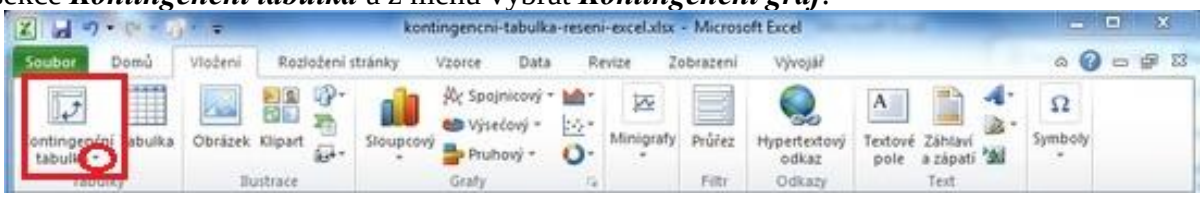


Popisky řádků	Součet z Ks	Součet z Cena	Součet z Prodej
Bathman	15	777	11655
Horolezectví	15	777	11655
Sedák	15	777	11655
Fantomas	24	369	8856
Horolezectví	24	369	8856
Karabina	24	369	8856
Rakosníček	24	159	3816
Horolezectví	24	159	3816
Osma	24	159	3816
Spiderman	12	12	144
Horolezectví	12	12	144
Lano	12	12	144

4.3.8. Kontingenční graf

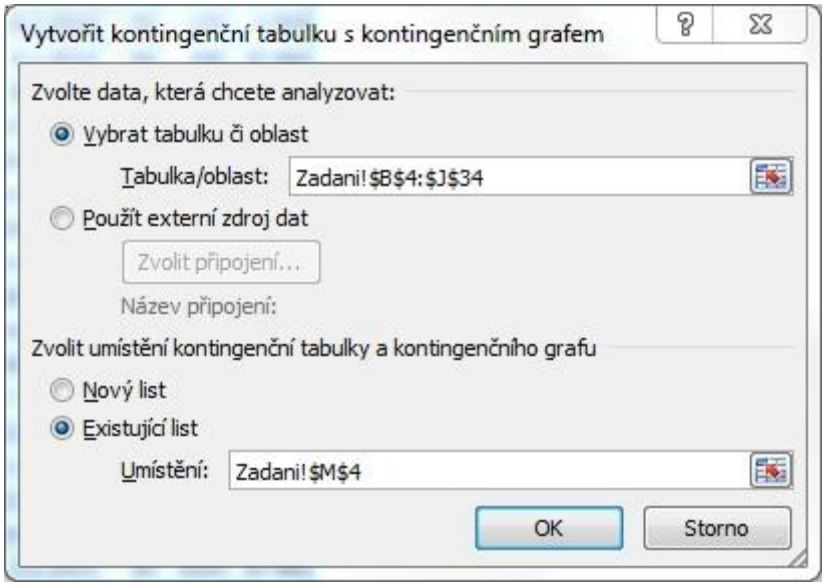
Kontingenční graf jde o divný název (stejně jako kontingenční tabulka), ale zdání klame, tvorba interaktivních kontingenčních grafů je jednoduchá a intuitivní. Můžete rychle a efektně měnit pohled na zobrazená data. Když se k tomu přičte, že při změnách v kontingenčním grafu (tabulce) nezničíte zdrojová data, jde o další nespornou výhodu. Stejně jako u kontingenční tabulky můžete data analyzovat, provádět souhrny, třídění, výpočty, a z těchto dat poté vytvořit kontingenční graf.

Začínáme ve zdrojové tabulce si vyberte nějakou buňku. Na kartě ***Vložení*** sekce ***Tabulky*** sekce ***Kontingenční tabulka*** a z menu vybrat ***Kontingenční graf***.



Vytvořit kontingenční tabulku s kontingenčním grafem

Obdržíte dialogové okno: ***kontingenční tabulku s kontingenčním graf***.



V tomto dialogovém okně nastavíte (něco je nastaveno automaticky)

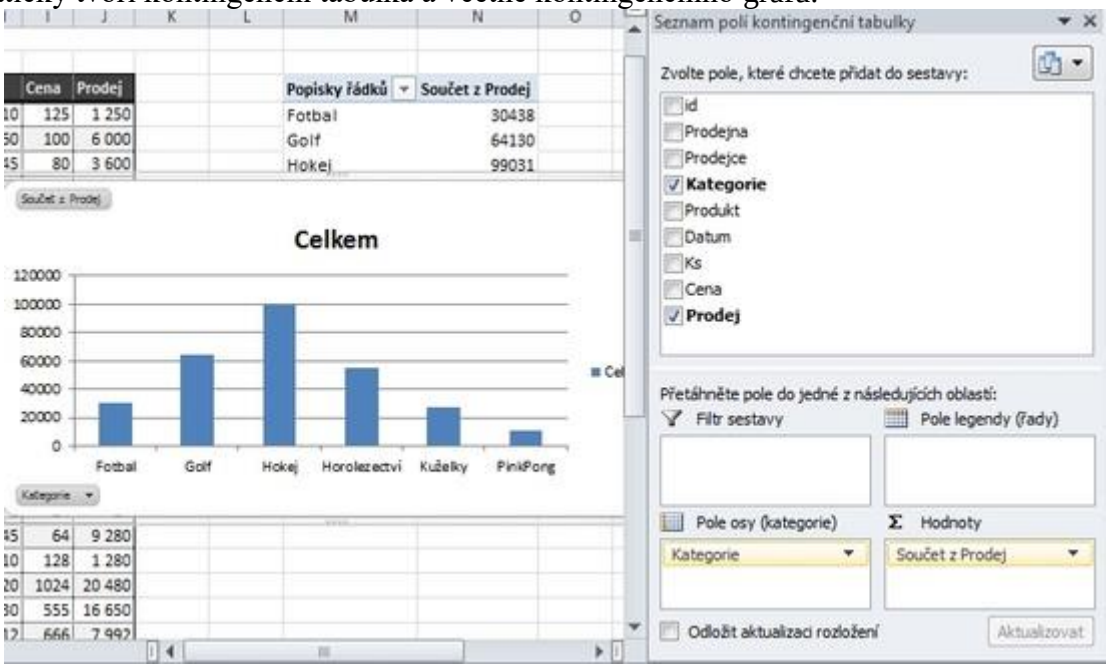
- **Zvolte data, která chcete analyzovat:** - je zaškrtnuto *Vybrat tabulku či oblast* – toto je přednastaveno automaticky
- **Zvolte umístění sestavy kontingenční tabulky:** - pro přehlednost zaškrtněte *Existující list* a do **Umístění:** zadejte horní levý roh pro kontingenční tabulku *Zadani!$M$4*
- **OK** – potvrdit

Seznam polí kontingenční tabulky (grafu)

V seznamu polí kontingenční tabulky stačí zaškrtnout tato pole:

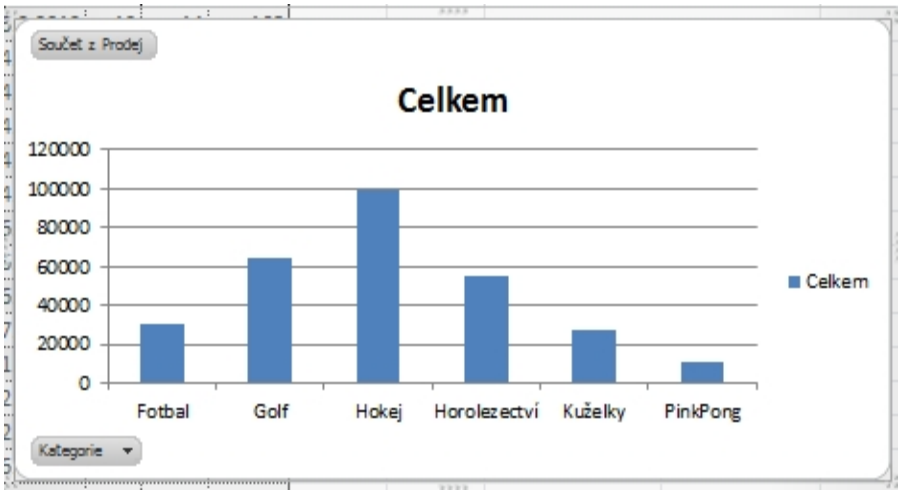
- **Kategorie**
- **Prodej**

Tvoříte graf, tak z důvodu přehlednosti vyberte jen klíčové položky. Zaškrtnutím se automaticky tvoří kontingenční tabulka a včetně kontingenčního grafu.



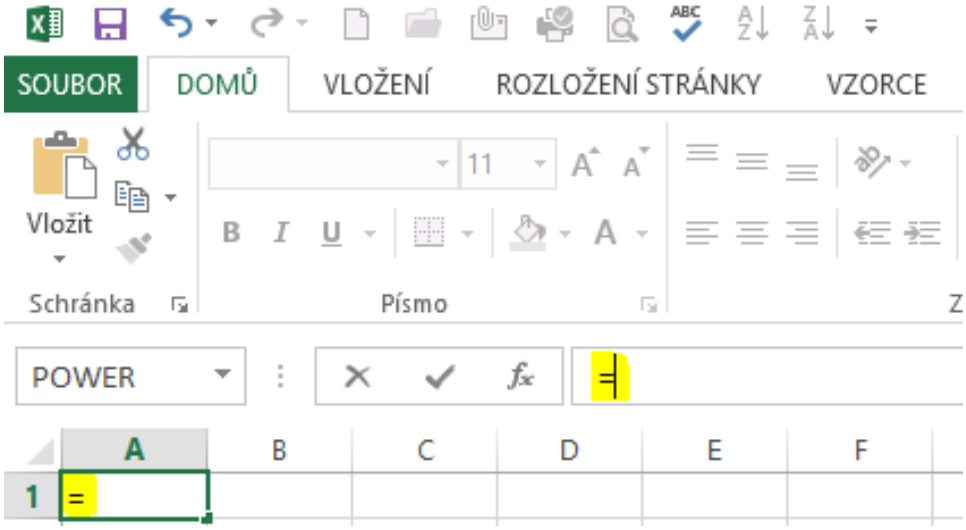
Kontingenční graf – hotovo

Kontingenční tabulku máte hotovou.

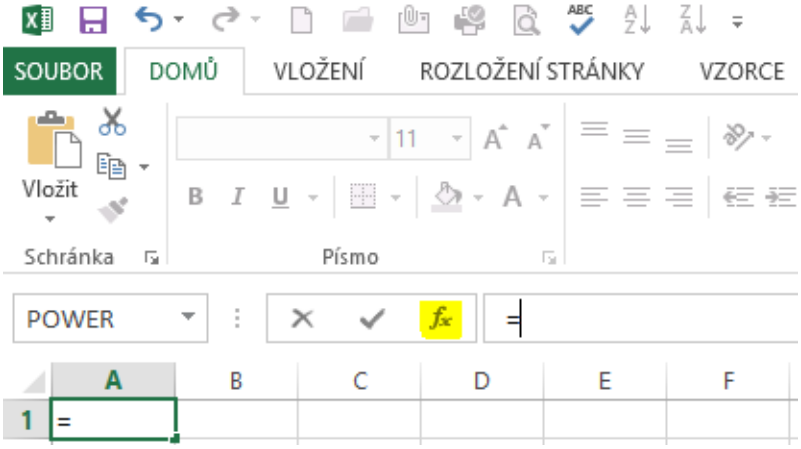


4.3.9. Přehled funkcí

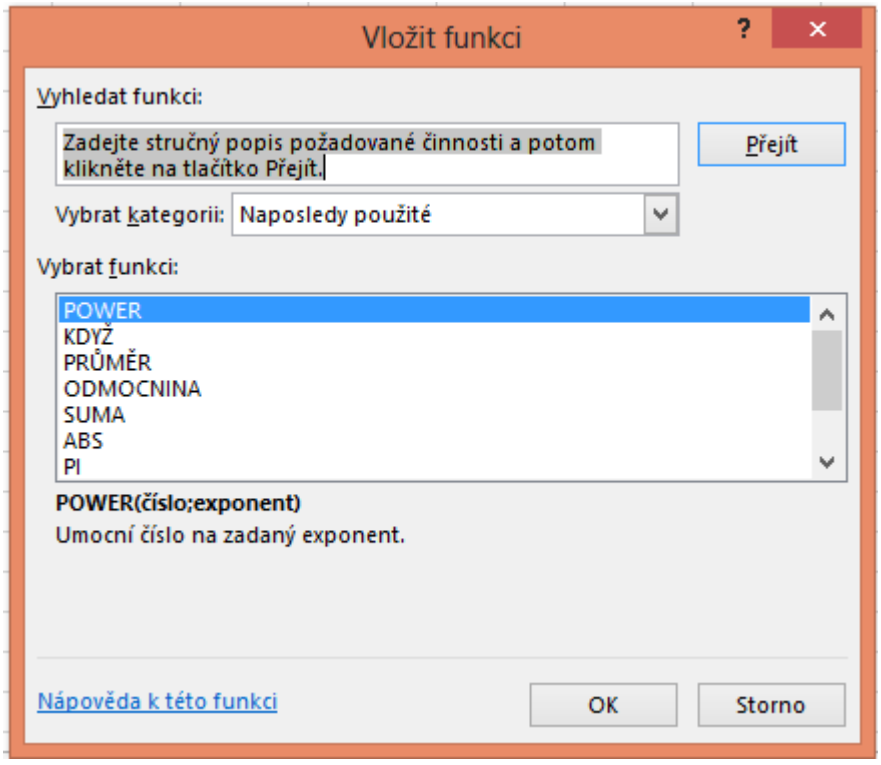
Rozhodl jsem se na tomto místě uvést abecední přehled funkcí, které Excel poskytuje a jejich základní popis, a to včetně ekvivalentů v anglickém jazyce. Tato kapitola nám pomůže, pokud budeme chtít něco spočítat a budeme potřebovat vědět, jak se daná funkce jmenuje v Excelu a ve které kategorii ji nalezneme. Ještě si narychlo řekneme, že pokud vpisujeme vzorec, či funkci, tak pro ruční zadání slouží řádek pro vzorec a zápis začíná vždy symbolem =.



Pokud chceme vložit vzorec pomocí dialogového okna a nabídek, tak užijeme tlačítko f_x



Které vyvolá následující dialog, kde si musíme danou funkci najít a vyvolat její dialog pro vyplnění parametrů.



Nyní již tedy přejdu ke slíbenému seznamu funkcí.

4.3.9.1. Funkce – A

CZ	EN	typ	Popis
A	AND	*logická*	Hodnota PRAVDA, pokud všechny argumenty jsou PRAVDA
ABS	ABS	*matematická*	Absolutní hodnota čísla
ACOT	ACOT	*matematická*	arkustangens čísla (Excel 2013)
ACOTH	ACOTH	*matematická*	hyperbolický arkustangens čísla (Excel 2013)
AMORDEGRC	AMORDEGRC	*finanční*	Lineární amortizace v každém účetním období pomocí koeficientu amortizace
AMORLINC	AMORLINC	*finanční*	Lineární amortizaci v každém účetním období
ARABIC	ARABIC	*matematická*	převede římské číslo na arabské (Excel 2013)
ARCCOS	ACOS	*matematická*	Arkuskosinus čísla
ARCCOSH	ACOSH	*matematická*	Hyperbolický arkuskosinu
ARCSIN	ASIN	*matematická*	Arkussinus čísla
ARCSINH	ASINH	*matematická*	Hyperbolický arkussinus
ARCTG	ATAN	*matematická*	Arkustangens čísla
ARCTG2	ATAN2	*matematická*	Arkustangens x-ové a y-ové souřadnice

CZ	EN	typ	Popis
ARCTGH	ATANH	*matematická*	Hyperbolický arkustangens
AVERAGEA	AVERAGEA	*statistické*	Průměr hodnot v seznamu argumentů

4.3.9.2. Funkce – B

CZ	EN	typ	Popis
BASE	BASE	*matematická*	Převede číslo na text v číselné soustavě o určeném základu. (Excel 2013)
BESSELI	BESSELI	*inženýrské*	Vrátí modifikovanou Besselovu funkci In(x)
BESSELJ	BESSELJ	*inženýrské*	Vrátí modifikovanou Besselovu funkci Jn(x)
BESSELK	BESSELK	*inženýrské*	Vrátí modifikovanou Besselovu funkci Kn(x)
BESSELY	BESSELY	*inženýrské*	Vrátí Besselovu funkci Yn(x)
BETADIST	BETADIST	*statistické*	
BETAINV	BETAINV	*statistické*	
BINOM.DIST.RANGE	BINOM.DIST.RANGE	*statistická*	pravděpodobnost zkušebního výsledku pomocí binomického rozdělení (Excel 2013)
BIN2DEC	BIN2DEC	*inženýrské*	Převede binární číslo na desítkové
BIN2HEX	BIN2HEX	*inženýrské*	Převede binární číslo na šestnáctkové
BIN2OCT	BIN2OCT	*inženýrské*	Převede binární číslo na osmičkové
BINOMDIST	BINOMDIST	*statistické*	
BITAND	BITAND	*inženýrská*	hodnotu bitové operace AND pro dvě čísla (Excel 2013)
BITLSHIFT		*inženýrská*	hodnotu čísla posunutá doleva o požadovaný počet bitů
BITOR	BITOR	*inženýrská*	hodnotu bitové operace OR pro dvě čísla
BITRSHIFT	BITRSHIFT	*inženýrská*	hodnotu čísla posunutá doprava o požadovaný počet bitů

CZ	EN	typ	Popis
BITXOR	BITXOR	*inženýrská*	hodnotu bitové operace Exkluzivní OR pro dvě čísla
BUDHODNOTA	FV	*finanční*	Vrátí budoucí hodnotu investice

4.3.9.3. Funkce – C

CZ	EN	typ	Popis
CELÁ.ČÁST	INT	*matematická*	Zaokrouhlí číslo dolů na nejbližší celé číslo
CEILING.MATH	CEILING.MATH	*matematická*	Zaokrouhlí číslo nahoru na nejbližší celé číslo nebo na nejbližší násobek zadané hodnoty (Excel 2013)
KOMBINACE.A	COMBINA	*matematická*	počet kombinací (s opakováním) pro daný počet položek (Excel 2013)
COMPLEX	COMPLEX	*inženýrské*	Z reálné a imaginární části komplexní číslo
CONCATENATE	CONCATENATE	*textové*	Spojí několik textových položek do jedné
CONFIDENCE	CONFIDENCE	*statistické*	
CONVERT	CONVERT	*inženýrské*	Převede číslo do jiného jednotkového měrného systém
CORREL	CORREL	*statistické*	
COS	COS	*matematická*	Kosinus čísla
COSH	COSH	*matematická*	Hyperbolický kosinus čísla
COT	COT	*matematická*	hyperbolický kosinus čísla (Excel 2013)
COTH	COTH	*matematická*	kotangens úhlu (Excel 2013)
COUNTBLANK	COUNTBLANK	*statistické*	
COUNTIF	COUNTIF	*statistické*	Počet buněk v oblasti odpovídající zadanému kritériu
COUNTIFS	COUNTIFS	*statistické*	Počet buněk v oblastech odpovídající zadaným kritériím
COUPDAYBS	COUPDAYBS	*finanční*	Počet dnů od začátku období placení kupónů do data splatnosti
COUPDAYS	COUPDAYS	*finanční*	Počet dnů v období placení kupónů, které obsahuje den zúčtování

CZ	EN	typ	Popis
COUPDAYSNC	COUPDAYSNC	*finanční*	Počet dnů od data zúčtování do následujícího data placení kupónu
COUPNCD	COUPNCD	*finanční*	Následující datum placení kupónu po datu zúčtování
COUPNUM	COUPNUM	*finanční*	Počet kupónů splatných mezi datem zúčtování a datem splatnosti
COUPPCD	COUPPCD	*finanční*	Předchozí datum placení kupónu před datem zúčtování
COVAR	COVAR	*statistické*	
CRITBINOM	CRITBINOM	*statistické*	
CSC	CSC	*matematická*	kosekans úhlu (Excel 2013)
CSCH	CSCH	*matematická*	hyperbolický kosekans úhlu (Excel 2013)
CUMIPMT	CUMIPMT	*finanční*	Kumulativní úrok splacený mezi dvěma obdobími
CUMPRINC	CUMPRINC	*finanční*	Kumulativní jistina splacena mezi dvěma obdobími půjčky

4.3.9.4. Funkce – Č

CZ	EN	typ	Popis
ČAS	TIME	*datum a čas*	Pořadové číslo času
ČASHODN	TIMEVALUE	*datum a čas*	Převede čas (ve formě textu) na pořadové číslo
ČÁST	MID	*textová*	Vrátí určený počet znaků textu počínaje zadaným místem
ČETNOSTI	FREQUENCY	*statistická*	Počet výskytů hodnot v rámci oblasti
ČISTÁ.SOUČHODNOTA	NPV	*finanční*	Čistá současná hodnota investice

4.3.9.5. Funkce – D

CZ	EN	typ	Popis
DATEDIF	DATEDIF	*datum a čas*	Rozdíl mezi dvěma kalendářními dny (daty)
DATUM	DATE	*datum a čas*	Pořadové číslo data (datumu)
DATUMHODN	DATEVALUE	*datum a čas*	Převede datum ve formě textu na pořadové číslo
DECIMAL	DECIMAL	*matematická*	Převede textový zápis čísla na číslo v desítkové soustavě (Excel 2013)

CZ	EN	typ	Popis
DEC2BIN	DEC2BIN	*inženýrské*	Převede desítkového čísla na dvojkové
DEC2HEX	DEC2HEX	*inženýrské*	Převede desítkové číslo na šestnáctkové
DEC2OCT	DEC2OCT	*inženýrské*	Převede desítkové číslo na osmičkové
DEGREES	DEGREES	*matematická*	Převede radiány na stupně
DÉLKA	LEN	*textová*	Počet znaků textového řetězce
DELTA	DELTA	*inženýrské*	Testuje rovnost dvou hodnot
DEN	DAY	*datum a čas*	Převede pořadové číslo na den v měsíci
DENTÝDNE	WEEKDAY	*datum a čas*	Převede pořadové číslo na denv týdnu
DETERMINANT	MDETERM	*matematická*	Vrátí determinant matice
DEVSQ	DEVSQ	*statistické*	
DISC	DISC	*finanční*	Diskontní sazba cenného papíru
DMAX	DMAX	*databáze*	Maximální hodnota z vybraných položek databáze (tabulky)
DMIN	DMIN	*databáze*	Minimální hodnota z vybraných položek databáze (tabulky)
DNES	TODAY	*datum a čas*	Pořadové číslo dnešního data
DNY	DAYS	*datum a čas*	Vrátí počet dní mezi dvěma daty
DOLLARDE	DOLLARDE	*finanční*	Převede částku v korunách (zlomek) na částku v korunách (desetinné číslo)
DOLLARFR	DOLLARFR	*finanční*	Převede částku v korunách (desetinné číslo) na částku v korunách (zlomek)
DOSADIT	SUBSTITUTE	*textová*	Nahradí v textu zadaný řetězec jiným
DPOČET	DCOUNT	*databáze*	Spočítá buňky databáze (tabulky) obsahující čísla
DPOČET2	DCOUNTA	*databáze*	Spočítá buňky databáze (tabulky), které nejsou prázdné
DPRŮMĚR	DAVERAGE	*databáze*	Průměr z vybraných položek databáze (tabulky)

CZ	EN	typ	Popis
DSMODCH	DSTDEVP	*databáze*	Směrodatná odchylku základního souboru vybraných položek databáze
DSMODCH.VÝBĚR	DSTDEV	*databáze*	Odhadne směrodatnou odchylku výběru vybraných položek databáze
DSOUČIN	DPRODUCT	*databáze*	Vynásobí hodnoty záznamů v databázi, které splňují daná kritéria
DSUMA	DSUM	*databáze*	Sečte čísla buněk v daném sloupci databáze splňující daná kritéria
DURATION	DURATION	*finanční*	Roční dobu cenného papíru s pravidelnými úrokovými sazbami
DVAR	DVARP	*databáze*	Vypočte rozptyl základního souboru vybraných buněk databáze
DVAR.VÝBĚR	DVAR	*databáze*	Odhadne rozptyl výběru vybraných buněk (položek) databáze (tabulky)
DZÍSKAT	DGET	*databáze*	Zobrazí z databáze jeden záznam splňující zadaná kritéria

4.3.9.6. Funkce – E

CZ	EN	typ	Popis
EDATE	EDATE	*datum a čas*	Pořadové číslo data, určený počet měsíců před/po počátečním datu
ENCODEURL	ENCODEURL	*webové*	řetězec kódovaný jako adresa URL (Excel 2013)
EOMONTH	EOMONTH	*datum a čas*	Pořadové číslo posledního dne měsíce před / po zadaném počtu měsíců
ERF	ERF	*inženýrská*	Vrátí chybovou funkci
ERFC	ERFC	*inženýrská*	Vrátí doplňkovou chybovou funkci
EXP	EXP	*matematická*	Základ přirozeného logaritmu umocněný na zadané číslo
EXPONDIST	EXPONDIST	*statistická*	

4.3.9.7. Funkce – F

CZ	EN	typ	Popis
FACTDOUBLE	FACTDOUBLE	*matematická*	Vrátí dvojitý faktoriál čísla
FAKTORIÁL	FACT	*matematická*	Faktoriál čísla
FDIST	FDIST	*statistická*	
FILTERXML	FILTERXML	*webové*	specifická data z obsahu XML pomocí zadaného výrazu XPath (Excel 2013)
FINV	FINV	*statistická*	
FISHER	FISHER	*statistická*	
FISHERINV	FISHERINV	*statistická*	
FLOOR.MATH	FLOOR.MATH	*matematická*	Zaokrouhlí číslo směrem dolů na nejbližší celé číslo nebo na nejbližší násobek (Excel 2013)
FORECAST	FORECAST	*statistická*	
FORMULATEXT	FORMULATEXT	*vyhledávací*	Funkce ze zadaného odkazu zformátovaná jako text
FTEST	FTEST	*statistická*	
FVSCHEDULE	FVSCHEDULE	*finanční*	Budoucí hodnota počáteční jistiny po použití série sazeb složitého úroku

4.3.9.8. Funkce – G

CZ	EN	typ	Popis
GAMMA	GAMMA	*statistická*	hodnotu funkce gama (Excel 2013)
GAMMADIST	GAMMADIST	*statistická*	
GAMMAINV	GAMMAINV	*statistická*	
GAMMALN	GAMMALN	*statistická*	Vrátí přirozený logaritmus gama funkce, Γ(x)
GAUSS	GAUSS	*statistická*	hodnotu standardního normálního kumulativního rozdělení snížená o 0,5 (Excel 2013)
GCD	GCD	*matematická*	Vrátí největší společný dělitel
GEOMEAN	GEOMEAN	*statistická*	Geometrický průměr pole nebo oblasti kladných dat
GESTEP	GESTEP	*inženýrská*	Testuje, zda je číslo větší než mezní hodnota

4.3.9.9. Funkce – H

CZ	EN	typ	Popis
HARMEAN	HARMEAN	*statistická*	
HEX2BIN	HEX2BIN	*inženýrská*	Převede šestnáctkové číslo na binární
HEX2DEC	HEX2DEC	*inženýrská*	Převede šestnáctkové číslo na desítkové
HEX2OCT	HEX2OCT	*inženýrská*	Převede šestnáctkové číslo na osmičkové
HLEDAT	SEARCH	*textová*	Nalezne textovou hodnotu
HODINA	HOUR	*datum a čas*	Převede pořadové číslo na hodinu
HODNOTA	VALUE	*textová*	Převede textový argument na číslo
HODNOTA.NA.TEXT	TEXT	*textová*	Zformátuje číslo a převede ho na text
HYPERTEXTOVÝ.ODKAZ	HYPERLINK	*vyhledávací*	Zástupce/odkaz, který otevře dokument na serveru, intranetu, Internetu.
HYPGEOMDIST	HYPGEOMDIST	*statistická*	

4.3.9.10. Funkce – CH

CZ	EN	typ	Popis
CHIDIST	CHIDIST	*statistická*	
CHIINV	CHIINV	*statistická*	
CHITEST	CHITEST	*statistická*	
CHYBA.TYP	ERROR.TYPE	*informační*	Číslo odpovídající typu chyby

4.3.9.11. Funkce – I

CZ	EN	typ	Popis
IFERROR	IFERROR	*logická*	Pokud chyba, vrátí zadanou hodnotu. V opačném případě vrátí výsledek vzorce
IFNA	IFNA	*logická*	pro #N/A zadaná hodnota jinak výskedek vzorce *od Excel 2013*
IMABS	IMABS	*inženýrská*	Absolutní hodnota (modul) komplexního čísla
IMAGINARY	IMAGINARY	*inženýrská*	Imaginární část komplexního čísla
IMARGUMENT	IMARGUMENT	*inženýrská*	Argument théta, úhel vyjádřený v radiánech

CZ	EN	typ	Popis
IMCONJUGATE	IMCONJUGATE	*inženýrská*	Komplexně sdružené číslo ke komplexnímu číslu
IMCOS	IMCOS	*inženýrská*	Kosinus komplexního čísla
IMCOSH	IMCOSH	*inženýrská*	hyperbolický kosinus komplexního čísla (Excel 2013)
IMCOT	IMCOT	*inženýrská*	kotangens komplexního čísla (Excel 2013)
IMCSC	IMCSC	*inženýrská*	kosekans komplexního čísla (Excel 2013)
IMCSCH	IMCSCH	*inženýrská*	hyperbolický kosekans komplexního čísla (Excel 2013)
IMDIV	IMDIV	*inženýrská*	Podíl dvou komplexních čísel
IMEXP	IMEXP	*inženýrská*	Exponenciální tvar komplexního čísla
IMLN	IMLN	*inženýrská*	Přirozený logaritmus komplexního čísla
IMLOG10	IMLOG10	*inženýrská*	Dekadický logaritmus komplexního čísla
IMLOG2	IMLOG2	*inženýrská*	Logaritmus komplexního čísla při základu 2
IMPOWER	IMPOWER	*inženýrská*	Komplexní číslo umocněné na celé číslo
IMPRODUCT	IMPRODUCT	*inženýrská*	Součin komplexních čísel
IMREAL	IMREAL	*inženýrská*	Reálná část komplexního čísla
IMSEC	IMSEC	*inženýrská*	sekans komplexního čísla (Excel 2013)
IMSECH	IMSECH	*inženýrská*	hyperbolický sekans komplexního čísla (Excel 2013)
IMSIN	IMSIN	*inženýrská*	Sinus komplexního čísla
IMSINH	IMSINH	*inženýrská*	hyperbolický sinus komplexního čísla (Excel 2013)
IMSQRT	IMSQRT	*inženýrská*	Druhou odmocninu komplexního čísla
IMSUB	IMSUB	*inženýrská*	Rozdíl mezi dvěma komplexními čísly
IMSUM	IMSUM	*inženýrská*	Součet dvou komplexních čísel
IMTAN	IMTAN	*inženýrská*	tangens komplexního čísla (Excel 2013)
INDEX	INDEX	*vyhledávací*	Pomocí rejstříku zvolí hodnotu z odkazu nebo matice

CZ	EN	typ	Popis
INTERCEPT	INTERCEPT	*statistická*	
INTRATE	INTRATE	*finanční*	Úrokova sazba plně investova-ného cenného papíru
INVERZE	MINVERSE	*matematická*	Provede inverzi matice
ISEVEN	ISEVEN	*informační*	Jde o sudé číslo?
ISFORMULA	ISFORMULA	*informační*	hodnotu PRAVDA, pokud odkaz obsahuje vzorec (Excel 2013)
ISODD	ISODD	*informační*	Jde o liché číslo?
ISOWEEKNUM	ISOWEEKNUM	*datum a čas*	číslo týdne v roce podle stan-dardu ISO (Excel 2013)
ISPMT	ISPMT	*finanční*	Výše úroku z investice zaplace-ného během určitého období

4.3.9.12. Funkce – J

CZ	EN	typ	Popis
JE.ČISLO	ISNUMBER	*informační*	Jde o číslo?
JE.CHYBA	ISERR	*informační*	Jde o chybu?
JE.CHYBHODN	ISERROR	*informační*	Jde o chybovou hodnotu?
JE.LOGHODN	ISLOGICAL	*informační*	Jde o logickou hodnotu?
JE.NEDEF	ISNA	*informační*	Jde o chybovou hodnotu #N/A
JE.NETEXT	ISNONTEXT	*informační*	Jde o netext?
JE.ODKAZ	ISREF	*informační*	Jde o odkaz?
JE.PRÁZDNÉ	ISBLANK	*informační*	Jde o prázdnou?
JE.TEXT	ISTEXT	*informační*	Jde o text?

4.3.9.13. Funkce – K

CZ	EN	typ	Popis
KČ	DOLLAR	*textové*	Převede číslo na text ve formátu měny
KDYŽ	IF	*logická*	Určí, který logický test má proběhnout
KÓD	CODE	*textové*	Kód prvního znaku zadaného textového řetězce
KOMBINACE	COMBIN	*matematická*	Počet kombinací pro daný počet položek
KURT	KURT	*statistická*	Hodnota špičatosti množiny dat

4.3.9.14. Funkce – L

CZ	EN	typ	Popis
LARGE	LARGE	*statistická*	K-tá největší hodnotu množiny dat
LCM	LCM	*matematická*	Nejmenší společný násobek
LINREGRESE	LINEST	*statistická*	
LINTREND	TREND	*statistická*	
LN	LN	*matematická*	Přirozený logaritmus čísla
LOG	LOG10	*matematická*	Dekadický logaritmus čísla
LOGINV	LOGINV	*statistická*	
LOGLINREGRESE	LOGEST	*statistická*	
LOGNORMDIST	LOGNORMDIST	*statistická*	
LOGZ	LOG	*matematická*	Logaritmus čísla při zadaném základu

4.3.9.15. Funkce – M

CZ	EN	typ	Popis
MALÁ	LOWER	*textová*	Převede text na malá písmena
MAX	MAX	*statistická*	Maximální hodnota ze seznamu argumentů
MAXA	MAXA	*statistická*	Maximální hodnota ze seznamu argumentů včetně čísel, textu a logických hodnot
MDURATION	MDURATION	*finanční*	
MEDIAN	MEDIAN	*statistická*	Střední hodnota zadaných čísel
MĚSÍC	MONTH	*datum a čas*	Převede pořadové číslo na měsíc
MIN	MIN	*statistická*	Minimální hodnotu ze seznamu argumentů
MINA	MINA	*statistická*	Nejmenší hodnota ze seznamu argumentů včetně čísel, textu a logických hodnot
MINUTA	MINUTE	*datum a čas*	Převede pořadové číslo na minutu
MÍRA.VÝNOSNOSTI	IRR	*finanční*	Vnitřní výnosové procento
MOD	MOD	*matematická*	Zbytek po dělení

CZ	EN	typ	Popis
MOD.MÍRA.VÝNOSNOSTI	MIRR	*finanční*	Vnitřní sazba výnosu
MODE	MODE	*statistická*	Hodnotu, která se v množině dat vyskytuje nejčastěji
MROUND	MROUND	*matematická*	Vrátí číslo zaokrouhlené na požadovaný násobek
MULTINOMIAL	MULTINOMIAL	*matematická*	Mnohočlen z množiny čísel
MUNIT	MUNIT	*matematická*	pro zadanou dimenzi vrátí jednotkovou matici (Excel 2013)

4.3.9.16. Funkce – N

CZ	EN	typ	Popis
N	N	*informační*	Hodnotu převedená na číslo
NÁHČÍSLO	RAND	*matematická*	Náhodné číslo mezi 0 a 1
NAHRADIT	REPLACE	*textová*	Nahradí znaky uvnitř textu
NAJÍT	FIND	*textová*	Nalezne textovou hodnotu uvnitř jiné
NE	NOT	*logická*	Logickou negaci argumentu
NEBO	OR	*logická*	Hodnota PRAVDA, pokud alespoň jeden argument je roven PRAVDA
NEDEF	NA	*informační*	Vrátí chybovouhodnotu#N/A
NEGBINOMDIST	NEGBINOMDIST	*statistická*	
NEPRAVDA	FALSE	*logická*	Logickou hodnotu NEPRAVDA
NEPŘÍMÝ.ODKAZ	INDIRECT	*vyhledávací*	Odkaz určený textovou hodnotou
NETWORKDAYS	NETWORKDAYS	*datum a čas*	Počet celých pracovních dní mezi dvěma daty
NETWORKDAYS.INTL	NETWORKDAYS.INTL	*datum a čas*	

CZ	EN	typ	Popis
NOMINAL	NOMINAL	*finanční*	Nominální roční úroková saza
NORMDIST	NORMDIST	*statistická*	
NORMINV	NORMINV	*statistická*	
NORMSDIST	NORMSDIST	*statistická*	
NORMSINV	NORMSINV	*statistická*	
NYNÍ	NOW	*datum a čas*	Pořadové číslo aktuálního data a času
NUMBERVALUE	NUMBERVALUE	*textové*	Převede text na číslo způsobem nezávislým na národním prostředí (Excel 2013)

4.3.9.17. Funkce – O

CZ	EN	typ	Popis
O.PROSTŘEDÍ	INFO	*informační*	Vrátí informace o aktuálním pracovním prostředí
OCT2BIN	OCT2BIN	*inženýrské*	Převede osmičkové číslo na binární
OCT2DEC	OCT2DEC	*inženýrské*	Převede osmičkové číslo na desítkové
OCT2HEX	OCT2HEX	*inženýrské*	Převede osmičkové číslo na šestnáctkové
ODDFPRICE	ODDFPRICE	*finanční*	Cena cenného papíru o nominální hodnotě 100 Kč s odlišným prvním obdobím
ODDFYIELD	ODDFYIELD	*finanční*	Výnos cenného papíru s odlišným prvním obdobím
ODDLPRICE	ODDLPRICE	*finanční*	Cenu cenného papíru o nominální hodnotě 100 Kč s odlišným posledním obdobím
ODDLYIELD	ODDLYIELD	*finanční*	Výnos cenného papíru s odlišným posledním obdobím
ODKAZ	ADDRESS	*vyhledávací*	Textový odkaz na jednu buňku listu
ODMOCNINA	SQRT	*matematická*	Druhá odmocnina
ODPIS.LIN	SLN	*finanční*	Vrátí přímé odpisy
ODPIS.NELIN	SYD	*finanční*	Směrné číslo ročních odpisů aktiva pro zadané období
ODPIS.ZA.INT	VDB	*finanční*	
ODPIS.ZRYCH	DB	*finanční*	Odpis aktiva za určité období
ODPIS.ZRYCH2	DDB	*finanční*	Odpis aktiva za určité období

CZ	EN	typ	Popis
OPAKOVAT	REPT	*textová*	Zopakuje text podle zadaného počtu opakování

4.3.9.18. Funkce – P

CZ	EN	typ	Popis
PEARSON	PEARSON	*statistická*	
PERCENTIL	PERCENTILE	*statistická*	
PERCENTRANK	PERCENTRANK	*statistická*	
PERMUTACE	PERMUT	*statistická*	
PERMUTACE.A	PERMUTATIONA	*statistická*	počet permutací pro daný počet objektů (s opakováním) (Excel 2013)
PDURATION	PDURATION	*finanční*	počet období potřebných k tomu, aby investice dosáhla určité hodnoty (Excel 2013)
PHI	PHI	*statistická*	hodnota funkce hustoty pro standardní směrodatnou odchylku (Excel 2013)
PI	PI	*matematická*	Hodnota čísla pí
PLATBA	PMT	*finanční*	Hodnotu pravidelné splátky anuity
PLATBA.ÚROK	IPMT	*finanční*	Výše úroku z investice zaplaceného během určitého období
PLATBA.ZÁKLAD	PPMT	*finanční*	Hodnota splátky jistiny pro zadanou investici za dané období
POČET	COUNT	*statistické*	Počet čísel v seznamu argumentů
POČET.BLOKŮ	AREAS	*vyhledávací*	Počet oblastí v odkazu
POČET.OBDOBÍ	NPER	*finanční*	Počet období pro investici
POČET2	COUNTA	*statistické*	Počet hodnot v seznamu argumentů
POISSON	POISSON	*statistická*	
POLÍČKO	CELL	*informace*	Vrátí informace o formátování, umístění nebo obsahu buňky
POSUN	OFFSET	*vyhledávací*	Posunutý odkazu od zadaného odkazu
POWER	POWER	*matematická*	Umocní číslo na zadanou mocninu
POZVYHLEDAT	MATCH	*vyhledávací*	Vyhledá hodnoty v odkazu nebo matici.
PRAVDA	TRUE	*logická*	Logickou hodnotu PRAVDA

CZ	EN	typ	Popis
PRICE	PRICE	*finanční*	Cenu cenného papíru o nominální hodnotě 100 Kč, ze kterého je úrok v pravidelných termínech
PRICEDISC	PRICEDISC	*finanční*	Cena diskontního cenného papíru o nominální hodnotě 100 Kč
PRICEMAT	PRICEMAT	*finanční*	Cena cenného papíru o nominální hodnotě 100 Kč, ze kterého je úrok placen k datu splatnosti
PROB	PROB	*statistická*	
PROČISTIT	TRIM	*textová*	Odstraní z textu mezery
PRŮMĚR	AVERAGE	*statistické*	Průměrnou hodnota argumentů
PRŮMOD-CHYLKA	AVEDEV	*statistické*	Průměr absolutních odchylek bodů

4.3.9.19. Funkce – Q

CZ	EN	typ	Popis
QUARTIL	QUARTILE	*statistická*	
QUOTIENT	QUOTIENT	*matematická*	Celá část dělení

4.3.9.20. Funkce – R

CZ	EN	typ	Popis
RADIANS	RADIANS	*matematická*	Převod stupně na radiány
RANDBETWEEN	RANDBETWEEN	*matematická*	Vrátí náhodné číslo mezi zadanými čísly
RANK	RANK	*statistická*	
RSQ	RKQ	*statistická*	
ROK	YEAR	*datum a čas*	Převede pořadové číslo na rok
ROK360	DAYS360	*datum a čas*	Vrátí počet dní mezi dvěma daty na základě roku s 360 dny
ROMAN	ROMAN	*matematická*	Převede arabskou číslice na římskou
ROUNDDOWN	ROUNDDOWN	*matematická*	Zaokrouhlí číslo dolů (k nule)
ROUNDUP	ROUNDUP	*matematická*	Zaokrouhlí číslo nahoru (od nuly)
ŘÁDEK	ROW	*vyhledávací*	Číslo řádku odkazu
ŘÁDKY	ROWS	*vyhledávací*	Počet řádků v odkazu
RRI	RRI	*finanční*	ekvivalentní úrokovou sazbu pro daný růst investic (Excel 2013)

4.3.9.21. Funkce – S

CZ	EN	typ	Popis
SEC	SEC	*matematická*	sekans úhlu
SECH	SECH	*matematická*	hyperbolický sekans úhlu
SEKUNDA	SECOND	*datum a čas*	Převede pořadové číslo na sekundu
SHEET	SHEET	*informační*	číslo odkazovaného listu
SHEETS	SHEETS	*informační*	počet listů v odkazu
SIGN	SIGN	*matematická*	Vrátí znaménko čísla
SIN	SIN	*matematická*	Sinus úhlu
SINH	SINH	*matematická*	Hyperbolický sinus
SKEW	SKEW	*statistická*	
SKEW.P	SKEW.P	*statistická*	nepravidelnost v rozložení na základě zadaných hodnot
SLOPE	SLOPE	*statistická*	
SLOUPCE	COLUMNS	*vyhledávací*	Počet sloupců v odkazu
SLOUPEC	COLUMN	*vyhledávací*	Číslo sloupce odkazu
SMALL	SMALL	*statistická*	K-tá nejmenší hodnota z množiny dat
SMODCH	STDEVP	*statistická*	
SMODCH.VÝBĚR	STDEV	*statistická*	
SOUČHODNOTA	PV	*finanční*	Současná hodnota investice
SOUČIN	PRODUCT	*matematická*	Vynásobí argumenty funkce
SOUČIN.MATIC	MMULT	*matematická*	Součin dvou matic
SOUČIN.SKALÁRNÍ	SUMPRODUCT	*matematická*	Součet součinů odpovídajících prvků matic
SQRTPI	SQRTPI	*matematická*	Vrátí druhou odmocninu výrazu (číslo * pí)
STDEVA	STDEVA	*statistická*	
STDEVPA	STDEVPA	*statistická*	
STEJNÉ	EXACT	*textové*	Jsou dvě textové hodnoty shodné
STEYX	STEYX	*statistická*	
SUBTOTAL	SUBTOTAL	*matematická*	Požadovaný souhrn dat v seznamu nebo v databázi
SUMA	SUM	*matematická*	Sečte argumenty funkce
SUMA.ČTVERCŮ	SUMSQ		
SUMIF	SUMIF	*matematická*	Sečte buňky vybrané podle zadaných kritérií

CZ	EN	typ	Popis
SUMIFS	SUMIFS	*matematická*	Sečte buňky vybrané podle více zadaných kritérií
SUMX2MY2	SUMX2MY2	*matematická*	Vrátí součet rozdílu čtverců odpovídajících hodnot ve dvou maticích
SUMX2PY2	SUMX2PY2	*matematická*	Vrátí součet součtu čtverců odpovídajících hodnot ve dvou maticích
SUMXMY2	SUMXMY2	*matematická*	Vrátí součet čtverců rozdílů odpovídajících hodnot ve dvou maticích
SVYHLEDAT	VLOOKUP	*vyhledávací*	Prohledá první sloupec matice a vrátí hodnotu buňky v udaném sloupci

4.3.9.22. Funkce – T

CZ	EN	typ	Popis
T	T	*textová*	Převede argumenty na text
TBILLEQ	TBILLEQ	*finanční*	Výnos směnky státní pokladny ekvivalentní výnosu obligace
TBILLPRICE	TBILLPRICE	*finanční*	Cena směnky státní pokladny o nominální hodnotě 100 Kč
TBILLYIELD	TBILLYIELD	*finanční*	Výnos směnky státní pokladny
TDIST	TDIST	*statistická*	
TG	TAN	*matematická*	Tangens čísla
TANH	TGH	*matematická*	Hyperbolický tangens čísla
TINV	TINV	*statistická*	
TRANSPOZICE	TRANSPOSE	*vyhledávací*	Vrátí transponovanou matici
TRIMMEAN	TRIMMEAN	*statistická*	
TTEST	TTEST	*statistická*	
TYP	TYPE	*informační*	Číslo označující datový typ hodnoty

4.3.9.23. Funkce – U

CZ	EN	typ	Popis
UNICHAR	UNICHAR	*textová*	znak unicode, na který odkazuje daná číselná hodnota (Excel 2013)
UNICODE	UNICODE	*textové*	číslo (bod kódu), které odpovídá k prvnímu znaku textu (Excel 2013)
ÚROKOVÁ.MÍRA	RATE	*finanční*	Úroková sazba za období anuity
USEKNOUT	TRUNC	*matematická*	Zkrátí číslo na celé číslo

4.3.9.24. Funkce – V

CZ	EN	typ	Popis
VAR	VARP	*statistická*	
VAR.VÝBĚR	VAR	*statistická*	
VARA	VARA	*statistická*	
VARPA	VARPA	*statistická*	
VELKÁ	UPPER	*textová*	Převede text na velká písmena
VELKÁ2	PROPER	*textová*	Převede první písmeno každého slova textu na velké
VPRAVO	RIGHT	*textová*	Vrátí první znaky textu umístěné nejvíce vpravo (ZPRAVA)
VLEVO	LEFT	*textová*	Vrátí první znaky textu umístěné nejvíce vlevo (ZLEVA)
VVYHLEDAT	HLOOKUP	*vyhledávací*	Prohledá první řádek matice a vrátí hodnotu buňky v udaném řádku
VYČISTIT	CLEAN	*textové*	Odebere z textu všechny netisknutelné znaky
VYHLEDAT	LOOKUP	*vyhledávací*	Vyhledá hodnoty ve vektoru nebo matici

4.3.9.25. Funkce – W

CZ	EN	typ	Popis
WEBOVÁSLUŽBA	WEBSERVICE	*webová*	Data z webové služby (Excel 2013)
WEEKNUM	WEEKNUM	*datum a čas*	Číslo představující číselnou pozici týdne v roce
WEIBULL	WEIBULL	*statistická*	Hodnota Weibullova rozdělení
WORKDAY	WORKDAY	*datum a čas*	Pořadové číslo data před nebo po zadaném počtu pracovních dní

4.3.9.26. Funkce – X

CZ	EN	typ	Popis
XIRR	XIRR	*finanční*	Čista současná hodnota pro harmonogram peněžních toků, který nemusí být nutně periodický
XNPV	XNPV	*finanční*	Výnos cenného papíru, ze kterého je úrok placen v pravidelných termínech
XOR	XOR	*logická*	Hodnotu logické operace XOR *od Excel 2013*

4.3.9.27. Funkce – Y

CZ	EN	typ	Popis
YEARFRAC	YEARFRAC	*datum a čas*	Část roku vyjádřená zlomkem. Počet dní mezi počátečním a koncovým datem
YIELD	YIELD	*finanční*	Výnos cenného papíru, ze kterého je úrok placen v pravidelných termínech
YIELDDISC	YIELDDISC	*finanční*	Roční výnos diskontního cenného papíru
YIELDMAT	YIELDMAT	*finanční*	Roční výnos cenného papíru, ze kterého je úrok placen k datu splatnosti

4.3.9.28. Funkce – Z

CZ	EN	typ	Popis
ZAOKR.DOLŮ	FLOOR	*matematická*	Zaokrouhlí číslo dolů, směrem k nule
ZAOKR.NAHORU	CEILING	*matematická*	Zaokrouhlí na nejbližší celé číslo
ZAOKROUHLIT	ROUND	*matematická*	Zaokrouhlí číslo na zadaný počet číslic
ZAOKROUHLIT.NA.LICHÉ	ODD	*matematická*	Zaokrouhlí číslo nahoru na nejbližší celé liché číslo
ZAOKROUHLIT.NA.SUDÉ	EVEN	*matematická*	Zaokrouhlí číslo nahoru na nejbližší celé sudé číslo
ZAOKROUHLIT.NA.TEXT	FIXED	*textová*	Zformátuje číslo jako text s pevným počtem desetinných míst
ZLEVA	LEFT	*textová*	(VLEVO) Vrátí první znaky textu umístěné nejvíce vlevo (Excel 2010)
ZNAK	CHAR	*textové*	Znak určený číslem
ZPRAVA	RIGHT	*textová*	Vrátí první znaky textu umístěné nejvíce vpravo (VPRAVO)
ZTEST	ZTEST	*statistická*	Jednostranná P-hodnota z-testu
ZVOLIT	CHOOSE	*vyhledávací*	Zvolí hodnotu ze seznamu hodnot

4.3.9.30. Funkce nové v Excel 2013 a 2016

Následující tabulka obsahuje nové funkce verze 2016

Funkce	Popis
CONCAT	Text: Sloučí text z více oblastí a/nebo řetězce, ale nenabízí možnost oddělovače nebo IgnoreEmpty argumenty. Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.
FORECAST.LINEAR	V aplikaci Excel 2016 je tato funkce nahrazená funkcí FORECAST.LINEAR v rámci nových prognostických funkcích, ale je pořád dostupná kvůli kompatibilitě s předchozími verzemi.
FORECAST.ETS	Statistické funkce: Vrátí budoucí hodnotu na základě existujících (historických) hodnot pomocí verze AAA algoritmu exponenciální vyrovnání (ETS). Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.
FORECAST.ETS.CONFINT	Statistické funkce: Vrátí interval spolehlivosti pro předpovídanou hodnotu v zadaném cílovém datu. Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.
FORECAST.ETS.SEASONALITY	Statistické funkce: Vrátí délku opakujícího se vzorku, který Excel zjistí pro zadanou časovou řadu. Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.
FORECAST.ETS.STAT	Statistické funkce: Vrátí statistickou hodnotu jako výsledek předpovídání časové řady. Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.
FORECAST.LINEAR	Statistické funkce: Vrátí budoucí hodnotu na základě existujících hodnot Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.
IFS	Logické: Zkontroluje, zda jsou splněny jedné nebo více podmínek a vrátí hodnotu, která odpovídá první pravdivé podmínce. Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.
MAXIFS	Statistické funkce: Vrátí maximální hodnotu z buňky vybrané podle danou sadu podmínek nebo kritérií. Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.
MINIFS	Statistické funkce: Vrátí minimální hodnotu mezi buňky vybrané podle danou sadu podmínek nebo kritérií. Tato funkce není dostupná v aplikaci Excel 2016 pro Mac.

V následují tabulce jsou nové funkce společné pro vezre 2013 a 2016

Funkce	Popis
ACOT	Matematické a trigonometrické funkce: Vrátí arkuskotangens čísla.
ACOTH	Matematické a trigonometrické funkce: Vrátí hyperbolický arkuskotangens čísla.
ARABIC	Matematické a trigonometrické funkce: Převede římské číslo na arabské.
BINOM.DIST.RANGE	Statistické funkce: Vrátí pravděpodobnost zkušebního výsledku pomocí binomického rozdělení.
BITAND	Inženýrské funkce: Vrátí hodnotu bitové operace AND pro dvě čísla.
BITLSHIFT	Inženýrské funkce: Vrátí hodnotu čísla posunutou doleva o počet bitů zadaný v operátoru velikost_posunu.
BITOR	Inženýrské funkce: Vrátí hodnotu bitové operace OR pro dvě čísla.
BITRSHIFT	Inženýrské funkce: Vrátí hodnotu čísla posunutou doprava o počet bitů zadaný v operátoru velikost_posunu.
BITXOR	Inženýrské funkce: Vrátí hodnotu bitové operace Exkluzivní OR pro dvě čísla.
CEILING.MATH	Matematické a trigonometrické funkce: Zaokrouhlí číslo nahoru na nejbližší celé číslo nebo na nejbližší násobek zadané hodnoty.
COMBINA	Matematické a trigonometrické funkce: Vrátí počet kombinací (s opakováním) pro daný počet položek.
COT	Matematické a trigonometrické funkce: Vrátí hyperbolický kosinus čísla.
COTH	Matematické a trigonometrické funkce: Vrátí kotangens úhlu.
CSC	Matematické a trigonometrické funkce: Vrátí kosekans úhlu.
CSCH	Matematické a trigonometrické funkce: Vrátí hyperbolický kosekans úhlu.
DAYS	Funkce data a času: Vrátí počet dní mezi dvěma daty.
DBCS	Textové funkce: Změní znaky s poloviční šířkou (jednobajtové) v řetězci znaků na znaky s plnou šířkou (dvoubajtové).
DECIMAL	Matematické a trigonometrické funkce: Převede textový zápis čísla o určitém základu na číslo v desítkové soustavě.
ENCODEURL	Webové funkce: Vrátí řetězec kódovaný jako adresa URL. Tato funkce není v aplikaci Excel Online dostupná.
FILTERXML	Webové funkce**:** Vrátí specifická data z obsahu XML pomocí zadaného výrazu XPath. Tato funkce není v aplikaci Excel Online dostupná.

Funkce	Popis
FLOOR.MATH	Matematické a trigonometrické funkce: Zaokrouhlí číslo směrem dolů na nejbližší celé číslo nebo na nejbližší násobek zadané hodnoty.
FORMULATEXT(funkce)	Vyhledávací funkce a funkce pro odkazy: Vrátí funkci ze zadaného odkazu, zformátovanou jako text.
GAMMA	Statistické funkce: Vrátí hodnotu funkce gama.
GAUSS	Statistické funkce: Vrátí hodnotu standardního normálního kumulativního rozdělení sníženou o 0,5.
IFNA	Logické funkce: Pokud je výsledkem vzorce chybová hodnota #NENÍ_K_DISPOZICI, vrátí tato funkce hodnotu, kterou zadáte. V opačném případě vrátí výsledek vzorce.
IMCOSH	Inženýrské funkce: Vrátí hyperbolický kosinus komplexního čísla.
IMCOT	Inženýrské funkce: Vrátí kotangens komplexního čísla.
IMCSC	Inženýrské funkce: Vrátí kosekans komplexního čísla.
IMCSCH	Inženýrské funkce: Vrátí hyperbolický kosekans komplexního čísla.
IMSEC	Inženýrské funkce: Vrátí sekans komplexního čísla.
IMSECH	Inženýrské funkce: Vrátí hyperbolický sekans komplexního čísla.
IMSINH	Inženýrské funkce: Vrátí hyperbolický sinus komplexního čísla.
IMTAN	Inženýrské funkce: Vrátí tangens komplexního čísla.
ISFORMULA	Informační funkce: Vrátí hodnotu PRAVDA, pokud odkaz vede na buňku, která obsahuje vzorec.

4.3.10. Funkce Když

Vrátí zadanou hodnotu, pokud je zadaná podmínka vyhodnocena jako PRAVDA, a jinou hodnotu, pokud je zadaná podmínka vyhodnocena jako NEPRAVDA.

4.3.10.1. Syntaxe v češtině

=KDYŽ(Podmínka; Ano; Ne)

4.3.10.2. Popis parametru:

- ***Podmínka*** – libovolná hodnota (výraz), který může nabývat hodnotu PRAVDA nebo NEPRAVDA
- ***Ano*** – je vrácená zadaná hodnota, je-li hodnota argumentu ***podmínka*** PRAVDA.
- ***Ne*** – je vrácená hodnota, je-li hodnota argumentu ***podmínka*** NEPRAVDA.

Poznámka: Při použití ve VBA nebo při hledání v anglických textech je tato funkce pod názvem **IF** - tj. pokud budete hledat u strejdy googla ;-) Navíc v anglické verzi Excelu je oddělovačem čárka ne středník ajko v české.

IF(Podmínka, Ano, Ne)

4.3.10.3. Parametr – Podmínka – může být:

- = je rovno
- <> není rovno
- > je větší než
- < je menší než
- >= je větší nebo rovno
- <= je menší nebo rovno

Teorií máte za sebou a jdeme na praktické příklady.

4.3.10.4. KDYŽ – obrácena logika - jedna podmínka

Pokračujeme jednoduchým úkolem. Kontrolujeme zadané číslo, které nesmí být jedna. Pokud hodnota v požadované buňce ***A21*** je jedna ***1*** (zadána číslem) potřebujete vypsat: ***Přepiš na jiné číslo!*** Pokud je různá od jedné vypíšete ***Správně***

Řešení:

=KDYŽ(A21<>1;"Správně";"Přepiš na jiné číslo!")

Logiku můžete obrátit, a i toto je správné řešení:

=KDYŽ(A21=1;"Přepiš na jiné číslo!";"Správně.")

Obě řešení jsou správná. Záleží na Vás, co se Vám více líbí a jak plánujete funkci KDYŽ rozšiřovat (jelikož podmínek může být více). Význam toho pochopíte dále na dalších příkladech.

4.3.10.4. KDYŽ – použití dvou podmínek – jak vnořovat funkce

Do jedné buňky ***A26*** se má zadat jednička ***1*** a do buňky ***A27*** zadejte slovo ***test***. Pokud je zadána jen jednička obdržíte info ***Zadána jen jednička.***, pokud jen slovo test ***zadáno jen slovo test***, pokud jednička i slovo test ***Zadána jednička a slovo test.*** pokud nic ***Nezadáno nic, co je požadováno.***

Řešení:

=KDYŽ(A26=1;KDYŽ(A27="test";"Zadána jednička a slovo test.";"Zadána jen jednička.");KDYŽ(A27="test";"zadáno jen slovo test";"Nezadáno nic co je požadováno."))

Řešení další – lze využít i dalších logických funkcí jako **A** a **NEBO** což ukáži na dalších příkladech.

Poznámka: Funkce můžete vnořovat více. Od MS Excel 2007 (do verze 2003 je vnoření pouze 7) vnoření může být až 64! Nejvíce jsem použil 10 vnoření, ale třeba někdy v budoucnu...

4.3.10.5. KDYŽ – ještě více podmínek

Potřebujete k číselnému rozmezí, které bude zadáno v buňce ***A34*** přiřadit písmeno (například školní výsledky testů). Požadované přiřazení je:

- **A** větší než 89
- **B** 80–89
- **C** 70–79
- **D** 60–69
- **E** menší než 60

Řešení:

=KDYŽ(A34>89;"A";KDYŽ(A34>79;"B";KDYŽ(A34>69;"C";KDYŽ(A34>59;"D";"E"))))

Řešení 2

Pokud buňku, do které zadáváte hodnotu pojmenujete ***průměr***

=KDYŽ(průměr>89;"A";KDYŽ(průměr>79;"B";KDYŽ(průměr>69;"C";KDYŽ(průměr>59;"D";"E"))))

Poznámka: Tento úkol je možné řešit i využitím jiné funkce, například pomocí SVYHLEDAT.

4.3.11. Komplexní čísla a Excel

Komplexní čísla vzniknou rozšířením reálných čísel. Tak že komplexnímu číslu (reálné složce) se přidá druhá (imaginární). Zapisuje se nejčastěji jako a + bi (případně a + bj), kde i(j) znamená imaginární složku komplexního čísla. Komplexní čísla lze interpretovat geometricky (osa x – reálná, osa y imaginární).

Komplexní čísla vám na první pokus nemusí v Excelu fungovat. Může se stát, že nemáte nainstalovanou jejich podporu. Podporu komplexních čísel provedete:

1.Karta Vývojář – Doplňky

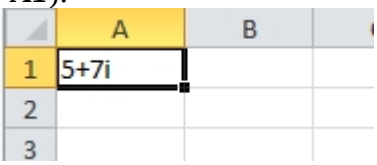
2.V zobrazeném okně zatrhněte Analytické nástroje, a potvrdit OK

4.3.11.1. Zadání

Tímto si ověříte, zda vám podpora komplexních čísel funguje správně:

=COMPLEX(5;7)

Pokud funguje vše správně obdržíte následující (*pokud je výše uvedená funkce zapsána v buňce **A1***):



Další možnost zadání komplexního čísla (pokud se vám nelíbí "i")

=COMPLEX(5;7;"j")

4.3.11.2. Základní operace s komplexními čísly

Součet dvou komplexních čísel

=IMSUM("3+4i";"5-3i")

=IMSUM(A1;A2)

Rozdíl dvou komplexních čísel

=IMSUB("13+4i";"5+3i")

=IMSUB(A1;A2)

Součin komplexních čísel

=IMPRODUCT("1+2i";30)

=IMPRODUCT("3+4i";"5-3i")

podíl dvou komplexních

=IMDIV("-28+240i";"10+24i")

4.3.11.3. Funkce pro práci s komplexními čísly

Pro práci s komplexními čísly jsou v Excelu k dispozici tyto funkce:

Český název funkce	Anglický název funkce
COMPLEX	*COMPLEX*
IMABS	*IMABS*
IMAGINARY	*IMAGINARY*

Český název funkce	Anglický název funkce
IMARGUMENT	*IMARGUMENT*
IMCONJUGATE	*IMCONJUGATE*
IMCOS	*IMCOS*
IMDIV	*IMDIV*
IMEXP	*IMEXP*
IMLN	*IMLN*
IMLOG10	*IMLOG10*
IMLOG2	*IMLOG2*
IMPOWER	*IMPOWER*
IMPRODUCT	*IMPRODUCT*
IMREAL	*IMREAL*
IMSQRT	*IMSQRT*
IMSUB	*IMSUB*
IMSUM	*IMSUM*

4.3.12. Matice

Matice (*nemyslím tu, co patří k šroubu a slovenskou už vůbec ne*) jako matematický pojem. Matice je skupina hodnot, se kterou se pracuje jako s celkem. Každý prvek matice má svůj řádkový a sloupcový index.

Teorii, jak se pracuje s maticemi, naleznete v každé lepší učebnici pro střední školy, či matematických skriptech pro VŠ. V dalších textu předpokládám základní matematickou znalost maticového počtu. Dále se budeme zabývat, jak aplikovat matice v Microsoft Excelu

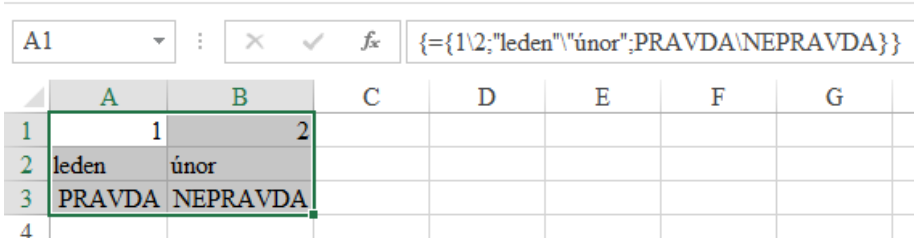
4.3.12.1. Zápis matice

Tady nastává první kámen úrazu, musíte vědět, kterou verzi Excelu máte a pro ni při zápisu použít vhodné formátovací znaky. Protože Microsoft systém zápisu mění (není jednotný napříč verzemi) je v tom trochu zmatek. Proto se asi maticový zápis moc nepoužívá (přitom si jich využitím ušetříte mnoho práce). Když už překonáte nástrahy zápisu, nesmíte zapomenout na klávesovou zkratku Ctrl+Shift+Enter. Pro verze 2010 a 2013 do první buňky vložíme:

={1\2;"leden"\"únor";PRAVDA\NEPRAVDA}

A následně označíme rozměr matice, tedy v našem případě 2x3 buňky. Když máme takto připraveno, tak zmáčkneme pravý ctrl+ pravý shift + hlavní (nikoli numlockový) enter.

Výsledkem bude toto:

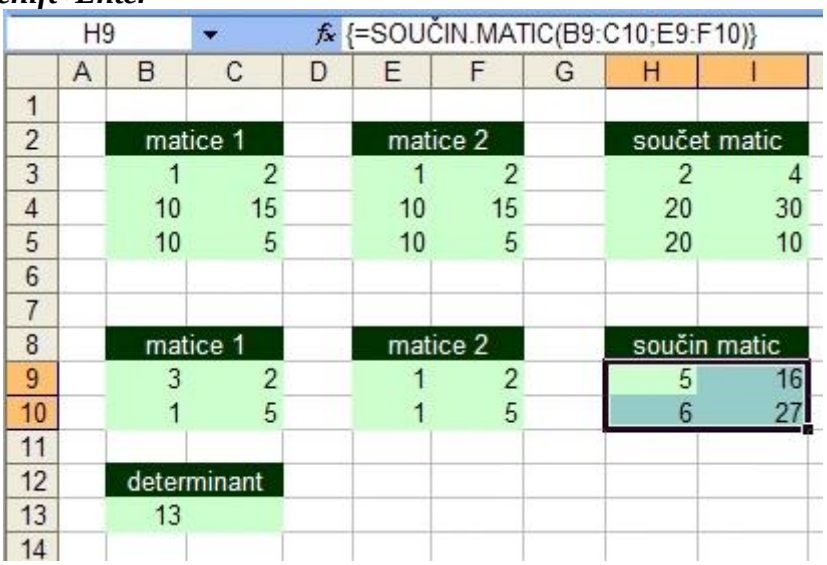


4.3.12.2. Operace s Maticemi

Nebudu se zabývat teorii matic (*Nemyslím třídění matiček a šroubku :)*). Předpokládám, že víte, jak se s maticemi pracuje, jaké matice se dají sčítat násobit. Jak se vypočte determinant.

4.3.12.2.1. Součet matic

Tady stačí jednoduchý vzorec A+B. Tj. v našem případě: =B3:C5+E3:F5
Výsledek můžete vidět na obrázku. Pozor při zápisu nezapomenout na klávesovou zkratku ***Ctrl+Schift+Enter***



4.3.12.2.2. Součin matic

Tady využijeme již předpřipravenou funkci SOUČIN.MATIC. Takže výsledek pro náš ukázkový příklad vypadá: =SOUČIN.MATIC(B9:C10;E9:F10)

4.3.12.2.3. Determinant

Pro výpočet determinantu, lze opět použít hotovou funkci (*nebo si naprogramovat vlastní*). Já využil to první:

=DETERMINANT(B9:C10)

4.3.12.3. Proč matice?

Když už máme základní ponětí, co jsou matice tak si rozebereme výhody a nevýhody.

Výhody

- jsou odolné vůči přepsaní
- lze uplatnit u složitějších výpočtu a výběru
- pro automatizované výběry nemusí být pomocná tabulka
- vcelku rychlé výpočty i u složitějších programů

Nevýhody

- na počátku chvílí trvá, než je člověk pochopí
- musí se pamatovat na trojklik :) Ctrl+Schift+Enter
- změny – Microsoft mění deklarace (člověk pak zbytečně hledá chybu jinde)

Neutrální

- **velikost – Ctrl+/** (/ z numerické klávesnice)

4.3.12.3. Transpozice

Mnohdy se hodí přehodit řádky za sloupce a sloupce za řádky v Microsoft Excelu. Jak by matematikové řekly, je potřeba provést transpozici matice. Excel zvládne transformovat nejen číslo, ale také text, datum, logické hodnoty atd.

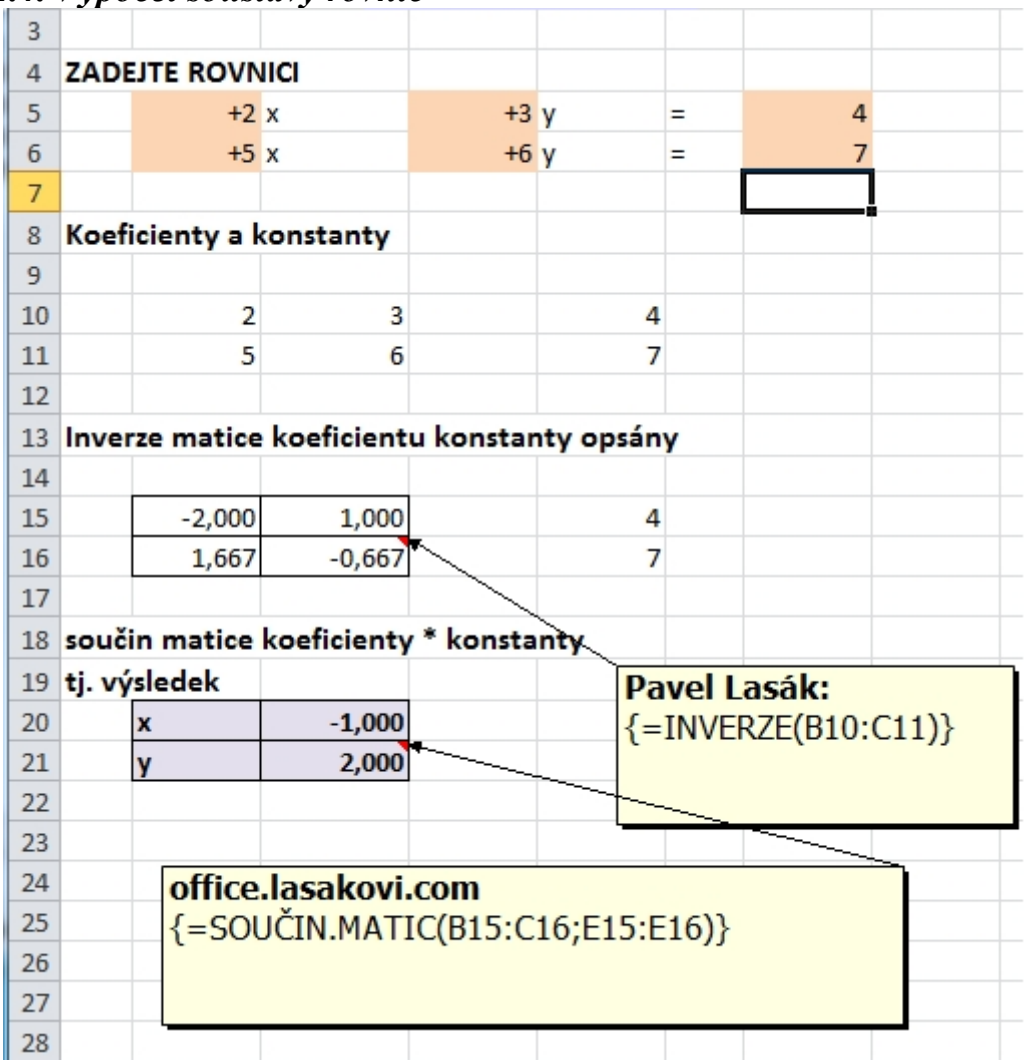
=TRANSPOZICE(pole) – je oblast kterou budeme transponovat.

Pro správnou funkci se musí funkce vložit jako matickový vzorec. Proto před vložením funkce je potřeba označit oblast, kam bude transponovaná matice vložena (oblast musí mít

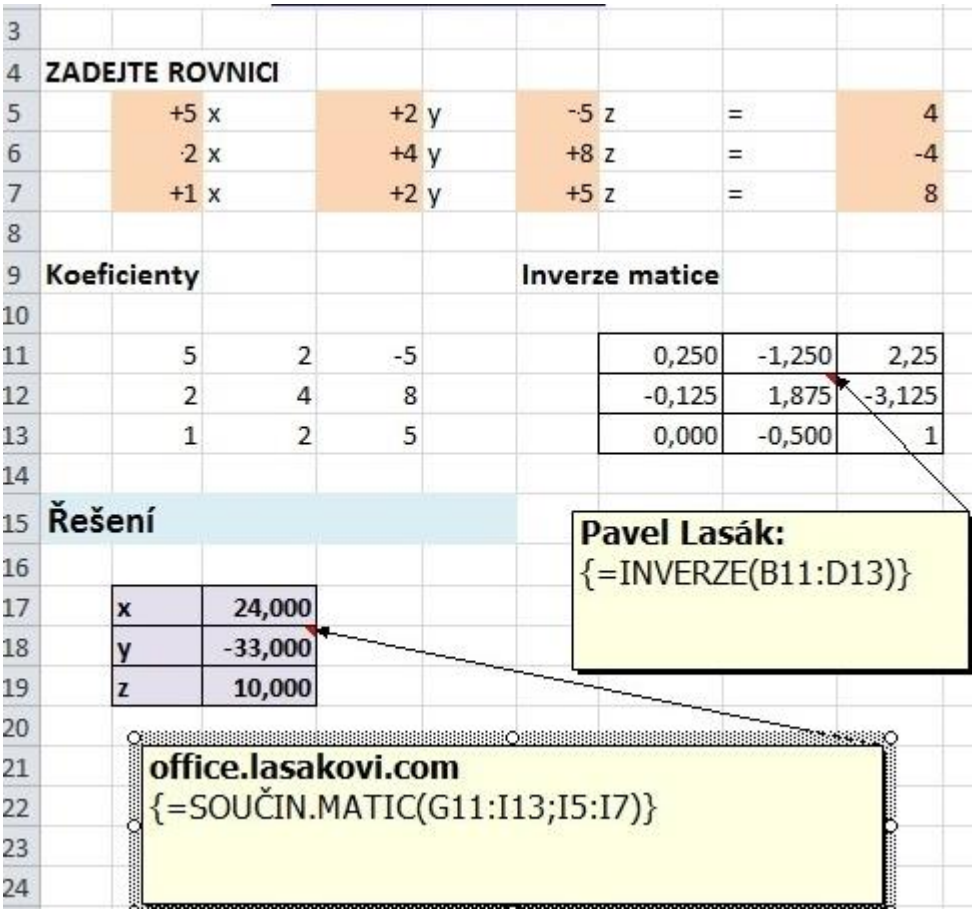
správný rozměr). Pak jen pokud je oblast označena zapíšete funkci ***=TRANSPOZICE*** a do závorky uvedete oblast původní matice. Pokud je hotovo nezapomenou stisknout klávesovou kombinaci ***Ctrl+ Schift + Enter*** a Microsoft Excel zařídí vše za nás (včetně složených závorek {}).

Poznáka: Pokud máme obdélníkovou oblast, nesmíme zapomenout, že počet řádku bude počet sloupců a naopak.

4.3.12.4. Výpočet soustavy rovnic



Zdroj Office.lasakovi.com, autor Pavel Lasák



Zdroj Office.lasakovi.com, autor Pavel Lasák

4.3.13. Grafy

Pro správnou a názornou prezentaci dat je nutné zvolit vhodný graf. Číselné údaje jsou snáze pochopitelné (alespoň většinu lidí, osobně mám raději tabulku).

4.3.13.1. Sloupcový

Sloupcový graf v angličtině ***Column Chart*** se mimo jiné hodí pro zobrazení datových hodnot v závislostí na časovém období (*stoupající cena akcií či zisku*), nebo kategorie (*například počet obyvatel v krajských městech atd.*). K vizuálnímu zobrazení hodnot, které porovnáváte (*dva a více výrobků*).



Sloupcové grafy zobrazují hodnoty jako řadu svislých sloupců, které jsou seskupených podle kategorie/kategorií. Porovnáváte skupiny údajů. Jelikož je sloupcový graf uveden jako první, předpokládá Microsoft nejčastější použití.

Na vodorovné ose je skupina - čas, kategorie, výrobky. Na svislé příslušné hodnoty (cena, velikost, účinnost...). Podobný typ grafu naleznete i pod názvem **pruhový**. Ze sloupcového grafu dají vytvořit odvozeniny, například **Vodopádový graf**.

4.3.13.1.1. Sloupcový graf - je vhodný

- **Obsahuje jednu nebo více řad.** - mající nějakou souvislost (prodeje za časové období)
- **Kategorie mohou obsahovat záporné hodnoty, nuly** – oproti některým grafům nevadí, pokud jsou data záporná

Sloupcový graf je vhodný, například ke grafickému zobrazení změn v čase. Případně pro srovnání, jaký vliv mají tyto položky v čase. Zobrazení hodnot za určité období (prodeje, zisky, ztráty, zákazníci). Můžete efektně porovnávat, například strukturu zákazníku (muži versus ženy), strukturu prodeje dle jednotlivých poboček, atd. Sloupcový graf je podobný pruhovému grafu. Sloupcový zobrazuje hodnoty svisle (pruhový vodorovně). Tady záleží na vašem grafickém cítění, který graf použijete (sloupcový nebo pruhový). A zda vyberete 2D verzi (obdélníky), nebo 3D verzi (hranoly, válce, kužely, jehlany).

4.3.13.1.2. Sloupcový graf – není vhodný

Použití sloupcového grafu není vhodné pokud:

- **Obsahuje desítky řad.** - stává se nepřehledným

4.3.13.1.3. Druhy sloupcových grafů

Pro více než čtyři řady (skupiny), doporučuji zvolit skládaný nebo prostorový graf. Pro více než 10 řad (skupin), bych doporučoval rozdělení do více grafů. Zpracovat data pro prezentaci jinak (zobrazení bude přehlednější). Vybírat můžete z:



- **Dvourozměrný graf**
  - **Skupinový sloupcový**
  - **Skládaný sloupcový**
  - **100% Skládaný sloupcový**
- **Prostorový sloupcový**
  - **Prostorový skupinový sloupcový**
  - **Skládaný sloupcový v prostorovém provedení**
  - **100% Skládaný sloupcový v prostorovém provedení**
  - **Prostorový sloupcový**
- **Válcový**
  - **Skupinový válcový**
  - **Skládaný válcový**
  - **100% Skládaný válcový**
  - **Prostorový válcový**
- **Kuželový**
  - **Skupinový kuželový**
  - **Skládaný kuželový**
  - **100% Skládaný kuželový**
  - **Prostorový kuželový**
- **Jehlanový**
  - **Skupinový jehlanový**
  - **Skládaný jehlanový**
  - **100% Skládaný jehlanový**
  - **Prostorový jehlanový**

4.3.13.2. Výsečový graf

Výsečový grafy zobrazí velikost jednotlivých položek datových řady úměrně k součtu položek do kruhové výseče. To znamená, že jsou sečteny všechny položky v řadě a tyto procentuálně rozděleny z celého součtu. Tj. procenta jsou podle velikosti jednotlivých položek.

Výsečový graf se podobá prstencovému grafu. Ale na rozdíl od prstencového může zobrazit data jedné řady.

Příklad: Z dotazníku máme odpovědi na otázku muž nebo žena? V odpovědi máme dvě položky muž (52 hlasů), žena (48 hlasů). Celkem 100 hlasů. V grafu bude připadat 52% na odpověď můž a 48% na odpověď žena. Tyto hodnoty můžeme mít z roku 2010. No a v dotazníku z roku 2011 můžeme mít počet žen 110 a mužů 60.

Poznámka: Samozřejmě počet odpovědí se nemusí rovnat 100, může být 14.328. Jen jsem uvedl ukázku pro názornost, ať k výpočtu nemusím mít kalkulačku.

4.3.13.2.1. Výsečový graf – je vhodný

Použití výsečového grafu je vhodné pro:

- **Výsledky voleb** – znáte z TV jako koláčové grafy

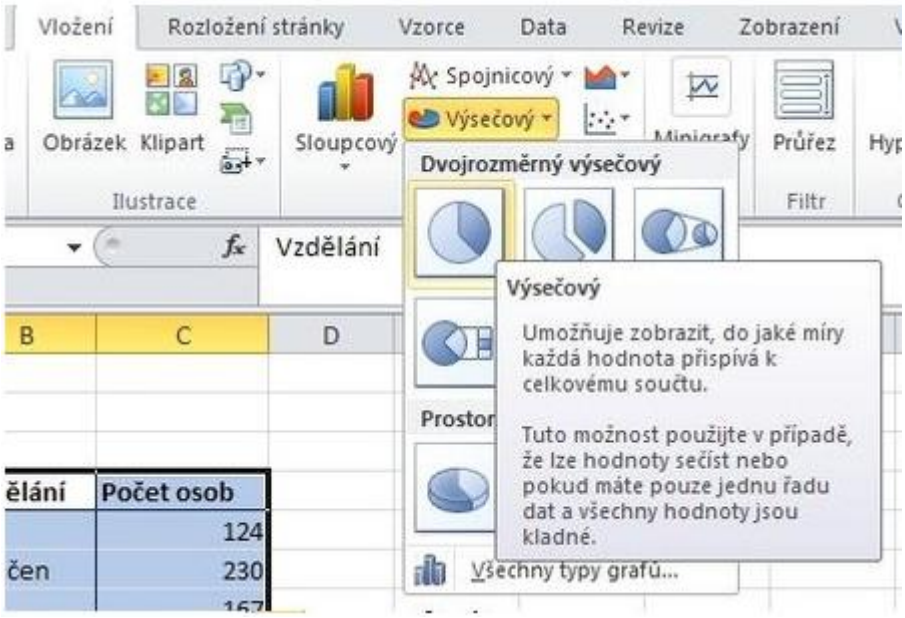
4.3.13.2.2. Výsečový graf – není vhodný

Použití prstencového grafu není vhodné pokud:

- **Obsahuje záporné hodnoty či nulu**
- **Kategorie není kompletní části grafů**
- **Obsahuje více řad.**

4.3.13.2.3. Vložení

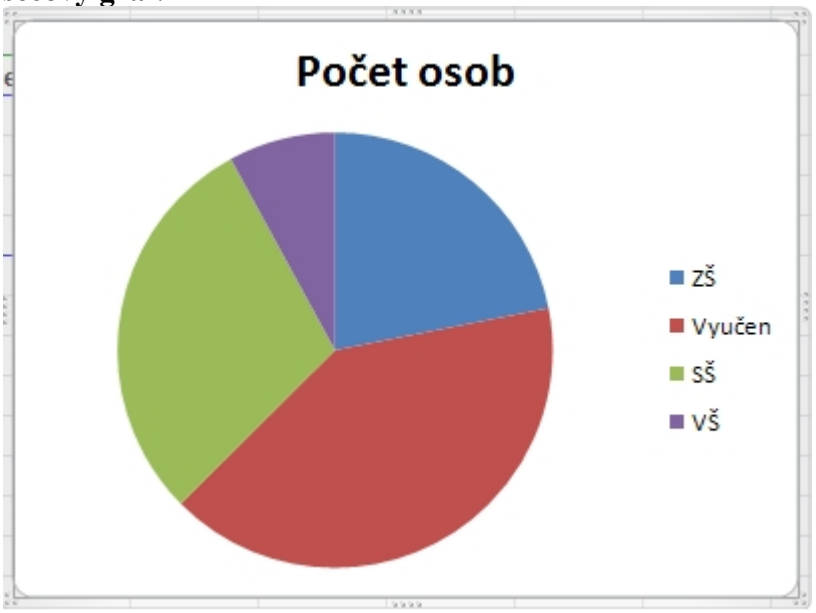
Vložení grafu je jednoduché, označíme připravenou tabulku a z karty **Vložení** ze skupiny **Grafy**. Skupina grafu **Výsečový**



Na výběr máme:

- **Výsečový**
- **Výsečový rozložený**
- **Výsečový s dílčí výsečí**
- **Výsečový s dílčími pruhy**
- **Výsečový s prostorovým efektem**
- **Výsečový rozložený s prostorovým efektem**

Vyberte **Výsečový graf**.



4.3.13.3. Prstencový graf

Prstencový grafy zobrazí velikost jednotlivých položek datových řady úměrně k součtu položek do prstence. Každý prstenec znázorňuje jednu datovou řadu, což znamená, že jsou sečteny všechny položky v řadě a jejich procentuální rozdělení z celého součtu. Tj. procenta jsou podle velikosti jednotlivých položek.

Prstencový graf se podobá výsečovému grafu. Ale na rozdíl od výsečového může zobrazit data z několika řad.

Příklad: Z dotazníku máme odpovědi na otázku muž nebo žena? V odpovědi máme dvě položky muž (52 hlasů), žena (48 hlasů). Celkem 100 hlasů. V grafu bude připadat 52% na odpověď muž a 48% na odpověď žena. Tyto hodnoty můžeme mít z roku 2010. No a v dotazníku z roku 2011 můžeme mít počet žen 110 a mužů 60.

Poznámka: Samozřejmě počet odpovědí se nemusí rovnat 100, může být 14.328. Jen jsem uvedl ukázku pro názornost ať k výpočtu nemusím mít kalkulačku.

4.3.13.3.1. Užití

Použití prstencového grafu vhodné pro:

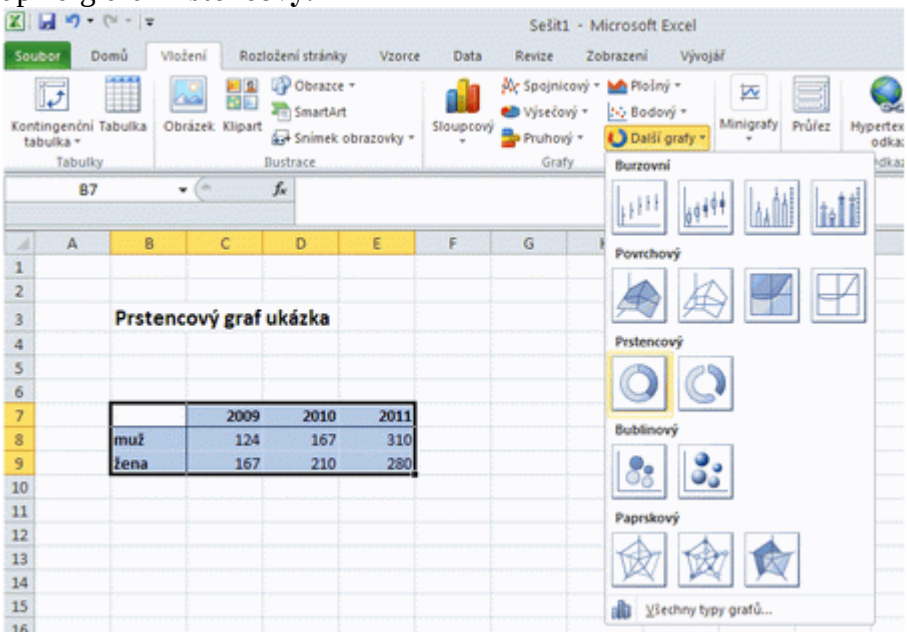
- **Výsledky voleb** – znáte z TV jako koláčové grafy
- **Výsledky sčítaní lidu** – porovnáni s daty před 10 lety
- **Příjmy a výdaje dle let** – možno použít i jiný graf záleží, jak chcete prezentaci dat pojmout

Použití prstencového grafu není vhodné pokud:

- **Obsahuje záporné hodnoty či nulu**
- **Kategorie není kompletní části grafů**

4.3.13.3.2. Vložení

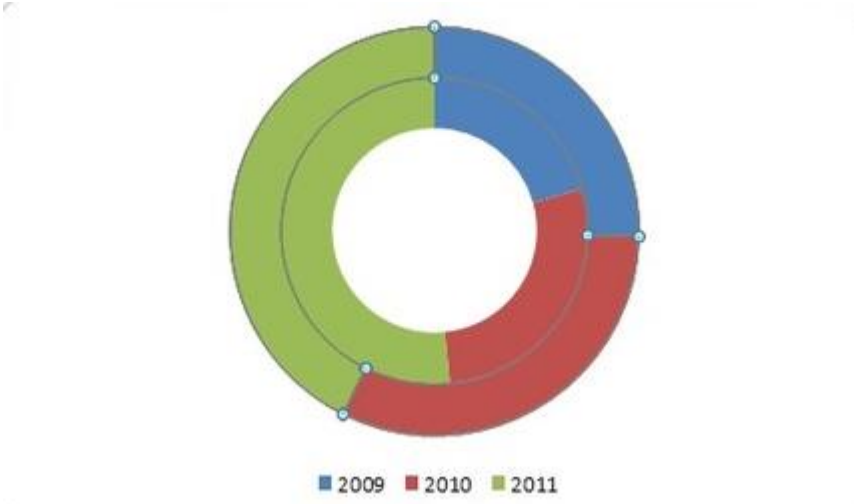
Vložení grafu je jednoduché, označíme připravenou tabulku a z karty **Vložení** ze skupiny **Grafy**. Skupina grafu **Prstencový.**



Na výběr máme:

- **Prstencový**
- **Prstencový s rozložením**

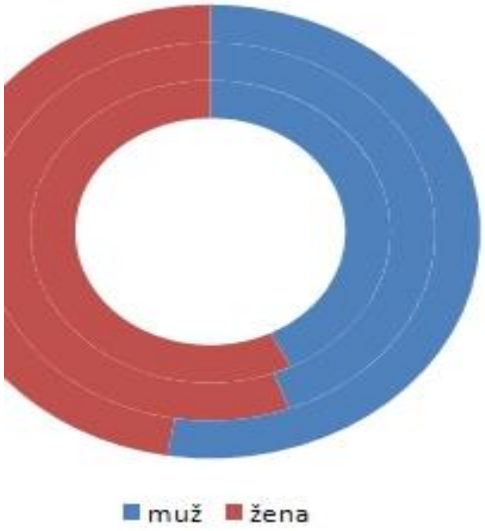
Vyberte **Prstencový graf**.



První volbou, ale Excel zvolil jako data sloupce, což nevyhovuje, jelikož naše data jsou v řádcích. Není nic jednoduššího než na nové skupině karet **Nástroje grafu**. Vyberte kartu **Návrh** ve skupině **Data** zvolíte **Zaměnit řádek za sloupec**.



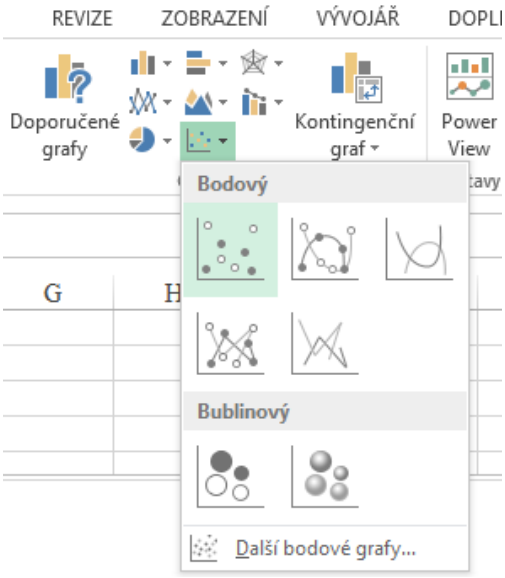
Graf se přepočítá.



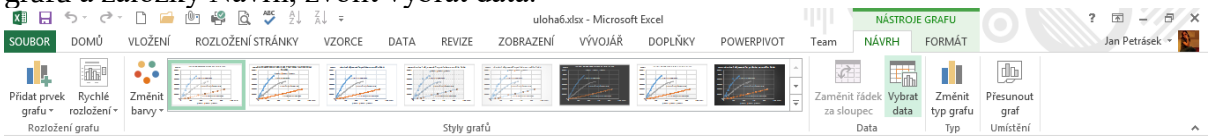
4.3.13.4. Bodový graf

Tento graf ocení především fyzici, matematici a také technici. Hodí se pro zobrazení složitých a reálných průběhů, které se dají zaznamenat v souřadnicích x,y. Tedy ideální je pro zobrazení závislosti něčeho na něčem, kdy body jdou následně prokládat pomocí různých aproximačních křivek a také je možné u těchto grafů provést predikci dat.

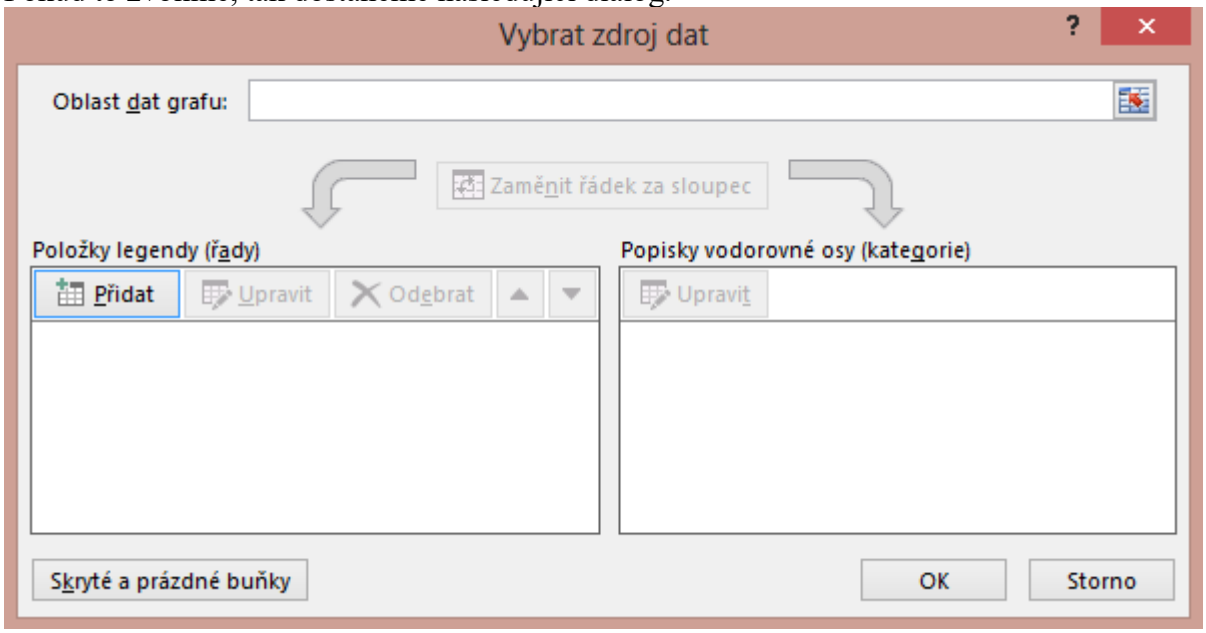
Graf vyložíme pomocí karty vložení a oblasti grafy (platí pro Excel 2013).



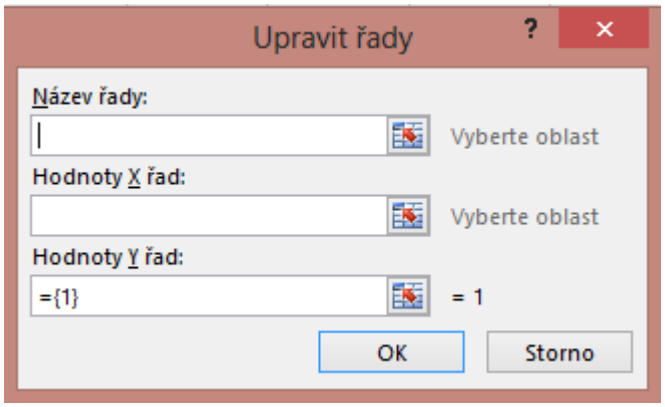
Doporučuji graf vkládat prázdný, bez vybrané oblasti a potom pomocí karty Nástroje grafu a záložky Návrh, zvolit vybrat data.



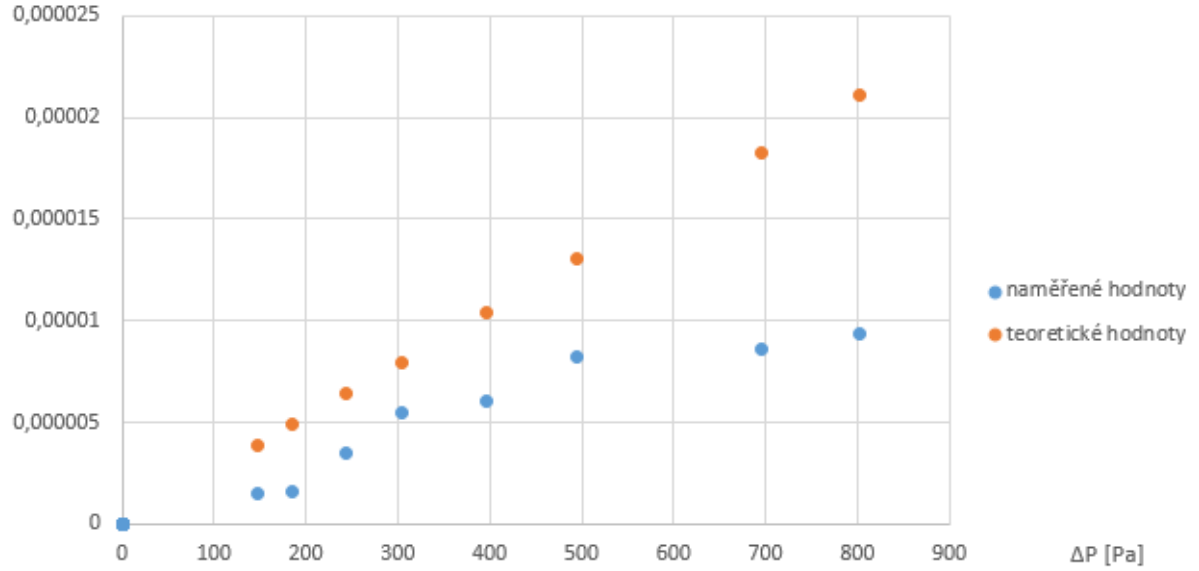
Pokud to zvolíme, tak dostaneme následující dialog.



Kde zvolíme přidat

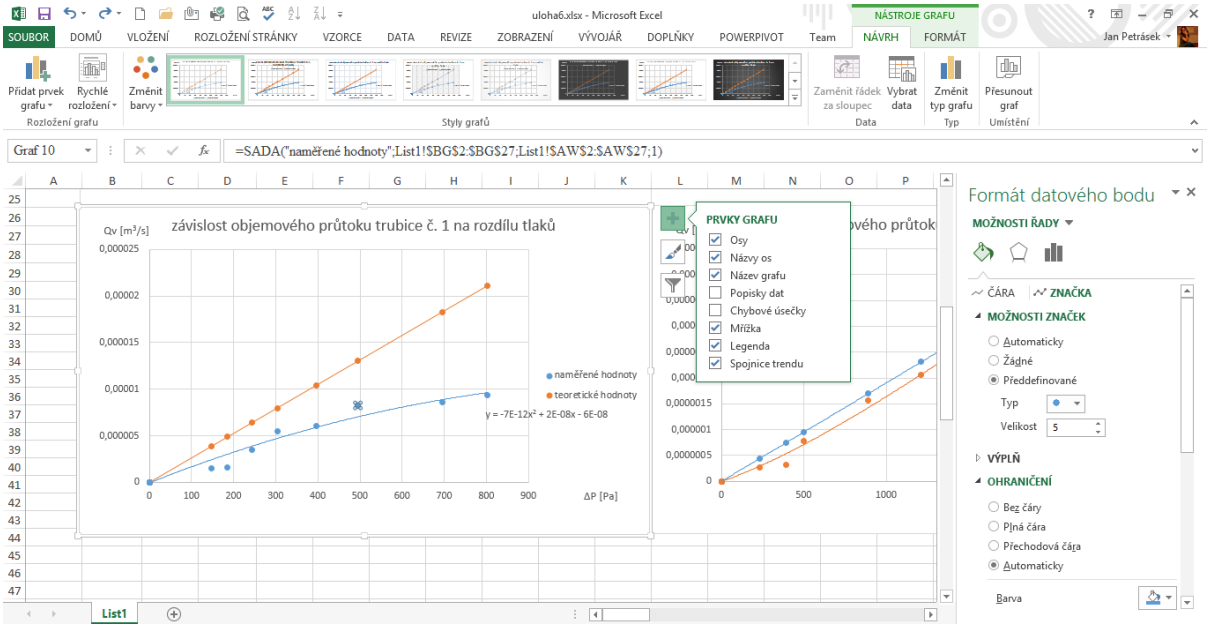


Vyplníme název řady a klikneme do položky hodnoty X řad, následně označíme hodnoty, které mají být na Xové ose grafu. U hodnoty Y řad musíme vymazat obsah a následně označit hodnoty, které patří na osu y. Takto získáme body pro jenu křivku, postup opakujeme pro všechny křivky, které chceme mít v grafu. Pro dvě řady může graf vypadat například takto:

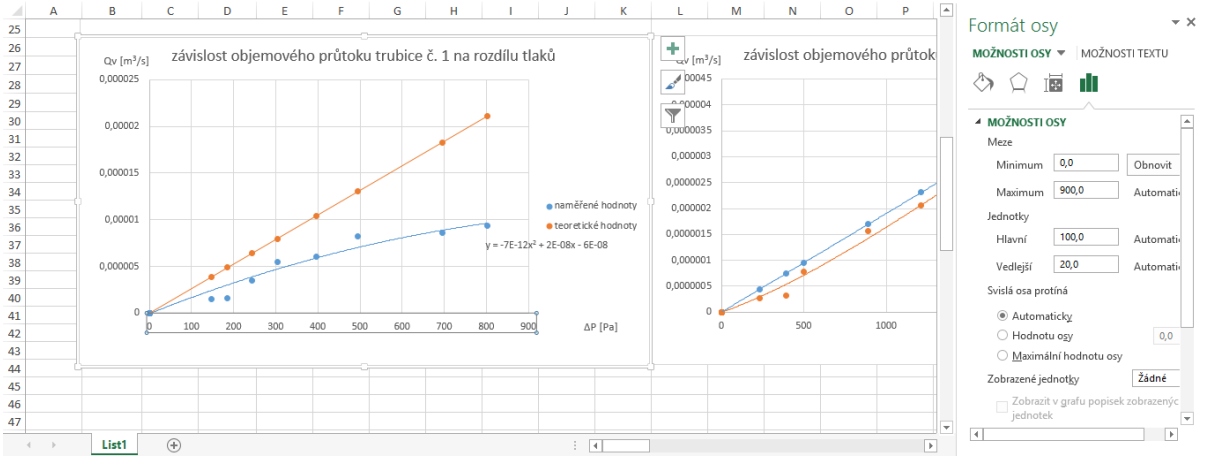


Pomocí kliknutí na zelené plus přímo u grafu, zvolíme názvy os, název grafu, spojnice trendu (aproximační křivky), popisky dat pod.

Dvojklikem na bod vyvoláme vlastnosti řady, kde můžeme měnit tvar a velikost bodu, jeho barvu a spoustu dalších vlastností.



Příklad výsledku měření laminárního proudění je vidět víše. Dvojklikem na čísla osy dostaneme formát osy a tam si nastavíme rozsah, citlivost atd. možností je strašně velké množství a je na každém, aby si ty možnosti vyzkoušel.



Tyto grafy mohou člověka i náramně pobavit, třeba jako v tomto článku: http://office.lasakovi.com/excel/grafy/graf-xy-bodovy-hodiny/

4.3.13.5. Polární graf

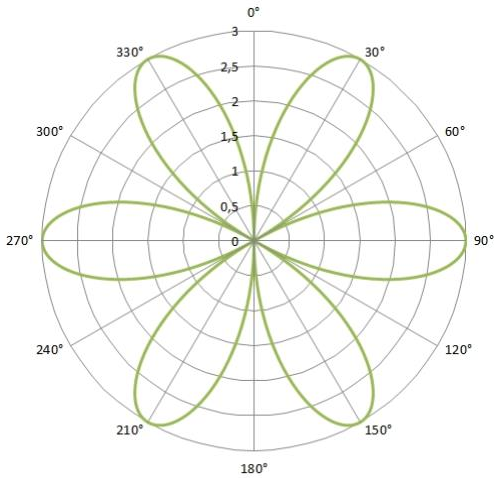
Pro některé prezentace získaných dat se nejlépe hodí polární grafy. Polární graf je závislost dané hodnoty (veličiny) na úhlu (0-360°). Čehož se hojně využívá nejen v elektrotechnice (například vyzařovací diagramy reproduktoru, charakteristiky svítivosti atd.).

Polární graf (diagram) jde speciální zobrazení hodnot, které mohou vykazovat cykličnost, nebo u nich záleží na směru. Při konstrukci polárního grafu jsou pro zobrazení jednoho bodu potřeba dva údaje úhel a velkosti od počátku (středu).

Bohužel Excel nemá přímo k dispozici polární graf. Lze sice využít paprskový, ale jeho úprava je pracná, proto v tomto článku ukáži, jak využít bezplatného doplňku PolarPlotter.xlam, který pomůže polární graf vytvořit efektivně. Doplněk lze bezplatně stáhnout na adrese: http://www.andypope.info/charts/polarplot.htm - stáhnete si doplněk **PolarPlotter.xlam**, který stačí spustit. Po spuštění se a v kartě **Vložit** objeví nová sekce **Polar Plot** s ikonou **Create**.



Označíte tabulku s daty a klikem na ikonu Create obdržíte dialogové okno, kde můžete provést vhodná nastavení, ale nemusíte, stačí klik na **OK** a vytvoříte Polární graf.

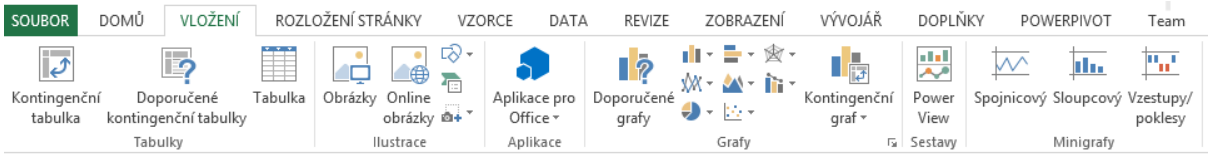


4.3.13.6. Mini graf

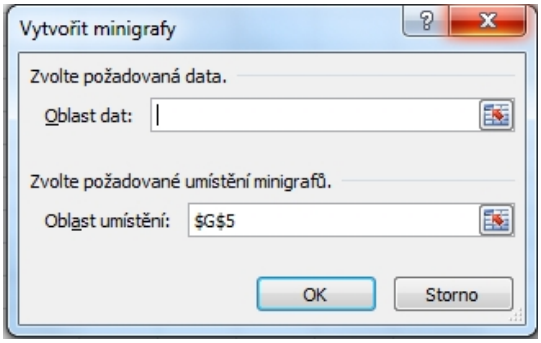
Potřebujete-li rychlý graf, můžete v Excelu využít minigrafů. Neboli vložit minigraf do buňky (od Microsoft Excel 2010). Jde o velice jednoduchý graf, který nemá označené a popsané osy. Využití nalezne, pokud potřebujete vytvořit rychlý náhled na data v tabulce. K dispozici jsou tři druhy minigrafů:

- Spojnicový
- Sloupcový
- Vzestupy/poklesy

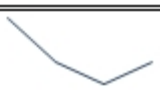
Označíte buňku, do které chcete umístit minigaf. Na pásu karet ***Vložení*** v sekce ***Minigrafy*** vyberte požadovaný typ minigrafu, v mojí ukázce jsem vybral ***Spojnicový***.



Obdržíte dialogové okno: **Vytvořit minigrafy**



Do pole *Oblast dat:* zadáte odkaz, na buňky s daty (ve kterých máte k dispozici data pro minigraf). Klikem na ***OK*** se ve vybrané buňce zobrazí minigraf.

Prodej Q ks / Výrobek	1	2	3	4	Miniraf
Pomeranč	120	100	90	100	
Banán	120	50	10	80	
Kiwi	25	35	30	28	
Rajče	70	110	115	80	

4.4. Závěrem

V této poněkud obsáhlé kapitole jsme si ukázali, jak vytvořit trojici nejužívanějších dokumentů. Je naprosto jasné, že Word bude stěžejní program, který nás všechny bude provázet po celý život. Některé programy jsou intuitivní, jako třeba PowerPoint, ale bohužel nejvíce místa si vyžádal Excel, protože je těžký na zvládnutí a nejen studenti, ale i já jej moc nemusím. Excel je ale program, který umí ušetřit hodně práce a připravuje vstupní data pro PowerPoint i pro Word. Pokud jste dali na mou radu a zkusili jste si sami udělat to, o čem jsme se tady bavili, tak byste bez problémů měli napsat diplomovou práci nebo i jiný oficiální dokument, který byste měli umět obhájit s užitím prezentace.

5. Práce s uživatelským uložištěm

Nejen aplikace a OS potřebují ukládat data trvale, ale především my uživatelé. Trvale ukládaná data jsou uložená na HDD, které může být rozděleno souborovým systémem na více logických jednotek. Souborový systém (ve Windows je to NTFS) umožňuje tvorbu adresářů a ukládání souborů. My jako uživatelé, kromě logické struktury se musíme starat i souborový systém. V této kapitole si řekneme, jak ušetřit místo bez mazání souborů, defragmentovat a tím urychlit práci, ale hlavně zálohovat a vypalovat data.

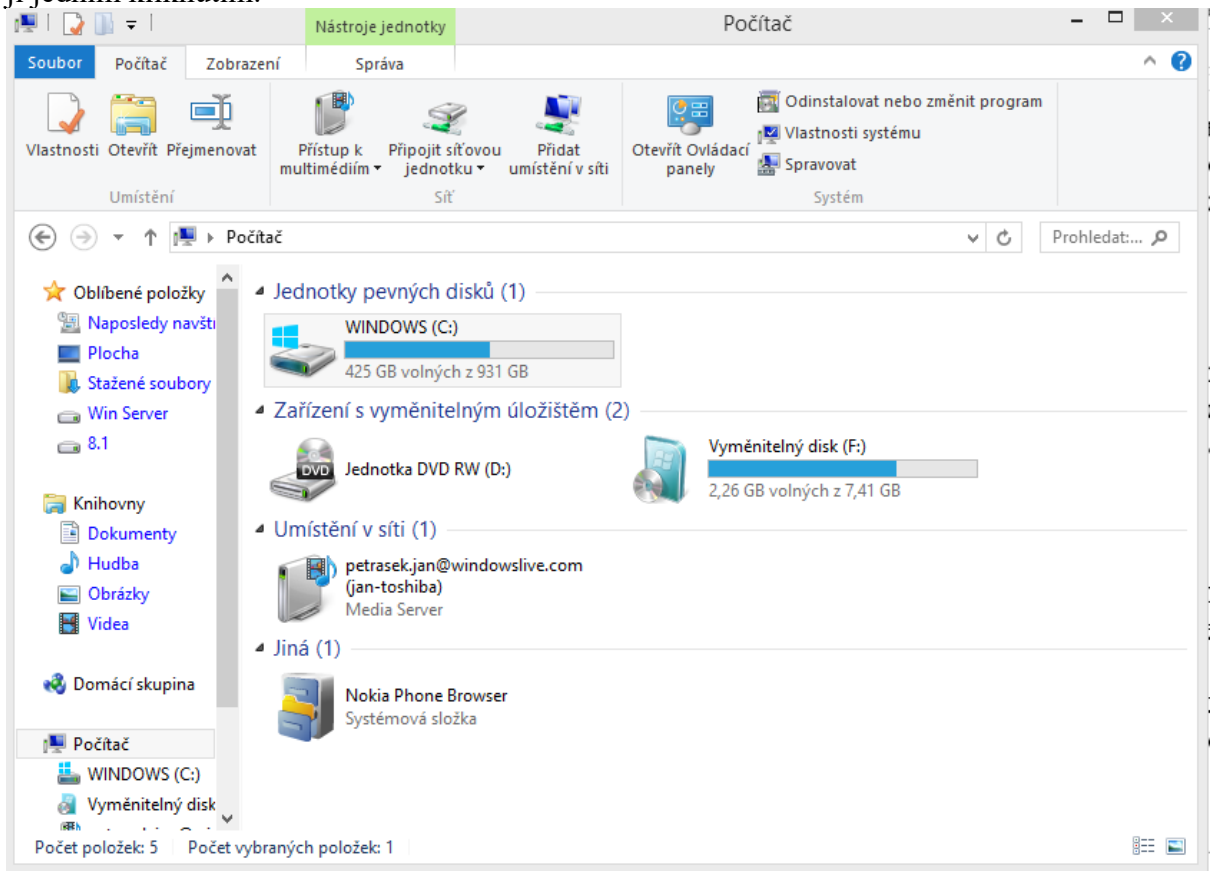
5.1. Komprese souborů

Máme několik možností, které můžeme vzájemně kombinovat. Hlavní základní variantou je komprimace jednotky. Tento proces je zdlouhavý a trvá v závislosti na velikosti disku od 10 do cca 26 hodin v závislosti na velikosti oddílu. Další možností jsou komprese pomocí převodu do komprimovaných formátů, či úprava multimediálních formátů.

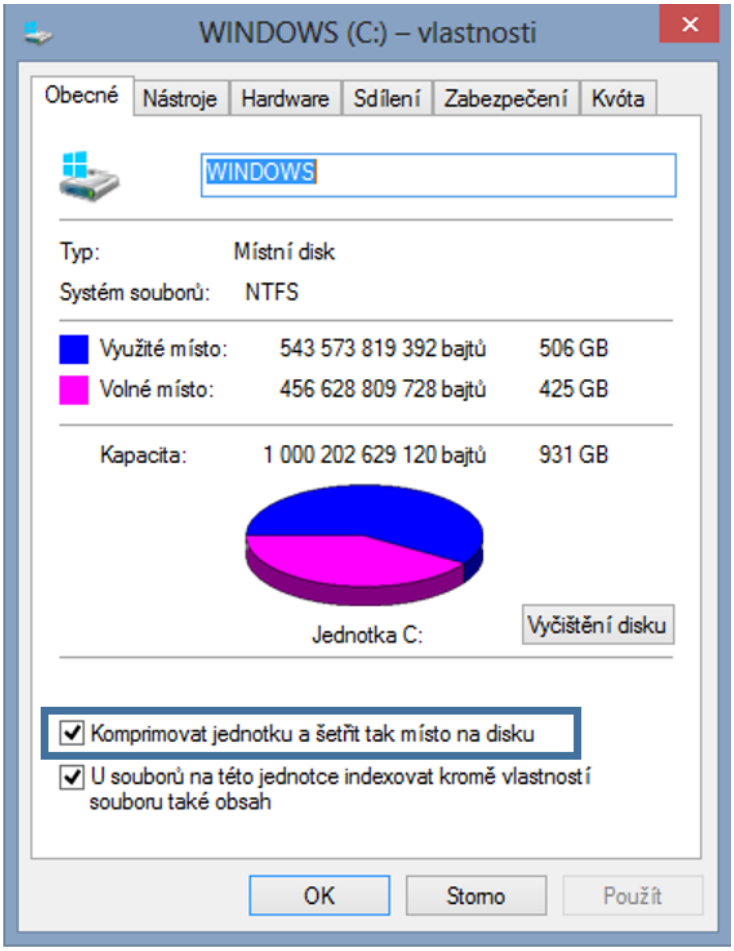
5.1.1. Komprese jednotky

Kompresí jednotky ušetříme místo za všechny typy souborů a nikterak neovlivníme rychlost ani práci se soubory. Nepotřebujeme k tomu žádný speciální program, všechny nástroje máme přímo v operačním systému.

Postup je jednoduchý a platný od Windows Me až po Windows 8.1. V položce počítač (pro XP a starší je to tento počítač) najdeme jednotku, kterou chceme komprimovat a označíme ji jedním kliknutím.



Jako další s pomocí pravého tlačítka otevřeme položku vlastnosti, kde zaškrtneme komprimovat jednotku. Pokračujeme stisknutím použít.



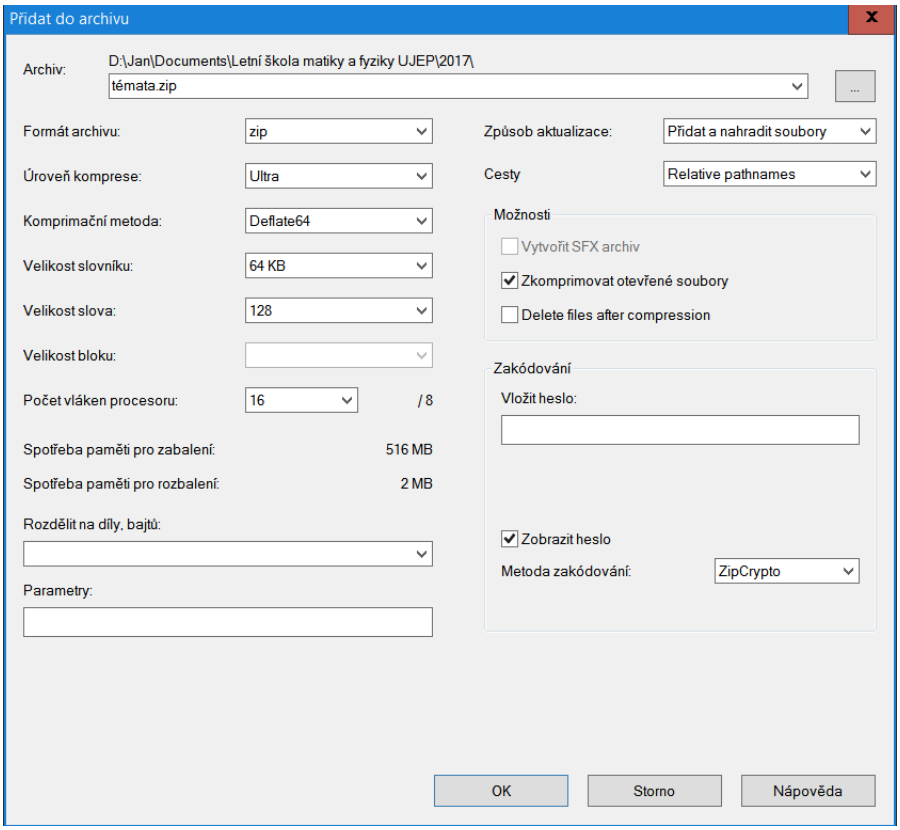
V následujícím dialogu vybereme použít pro jednotku včetně podsložek a souborů, nebo alespoň tak nějak by to tam mělo být, je to ta varianta, která je spodní a potvrdíme ok. Při této operaci je nutné mít oprávnění správce a krom toho doporučuji uzavřít všechny aplikace, jediné, co je potřeba nechat v činnosti je Explorer.exe. Dokončení je opravdu velmi prosté, stačí jen pár hodin čekat ☺.

Touto kompresí se komprimují i systémové a již komprimované soubory. Je nutné dávat si pozor, když budeme následně z takto upravené jednotky někomu kopírovat soubor, protože systém nám v tom základním náhledu zobrazí komprimovanou velikost souboru. Pokud se chceme dozvědět pravou velikost souboru, tak musíme pomocí vlastností a podívat se na velikost souboru, nikoli na velikost na disku. Další anomálií této datové komprese je nutnost rekomprese po cca roce, nebo při uložení většího množství souborů. Rekompresi provedeme tak, že zrušíme kompresi, dáme použít a po rozjetí vše stornujeme. Následně znovu provedeme kompresi.

5.1.2. Kompresní formáty

Další možností, jak ušetřit místo na uložišti je použít komprimované soubory. Velmi slabý, nicméně jeden z nejstarších a v průzkumníku stále nativních formátů je zip. Zip soubory nejsou nikterak silně komprimované a spíše nežli pro ukládání souborů či adresářů se hodí pro archivaci souborů určených k odeslání emailem, nebo pomocí jiných komunikačních prostředků. Jeho hlavní výhodou je hluboká podpora napříč operačními systémy, protože dobré kompatibilitě se těší nejen u Windows, ale i u Linuxu a mobilních systémů. Ve Windows zip vznikne pomocí nabídky pravého tlačítka myši a položky odeslat => komprimovaná složka. K rozbalení stačí dvojklik.

Pro torbu kvalitních zipů, které mají velký kompresní poměr a další pokročilou práci se zepy, například jejich dělení na party je již potřeba externí aplikace. Z externích aplikací doporučuji 7-zip, který má podporu i více formátů pro nás je ovšem zaímavý formát zip, kde si můžeme nastavit použitý algoritmus, velikost slova, velikost slovníků a počet vláken určených pro vytvoření zipu, je zde možné nastavit případně heslo či zmiňované dělení na party. Z vlastní zkušenosti mohu doporučit následující nastavení, které dosahuje ve většině případů nejlepších výsledků.

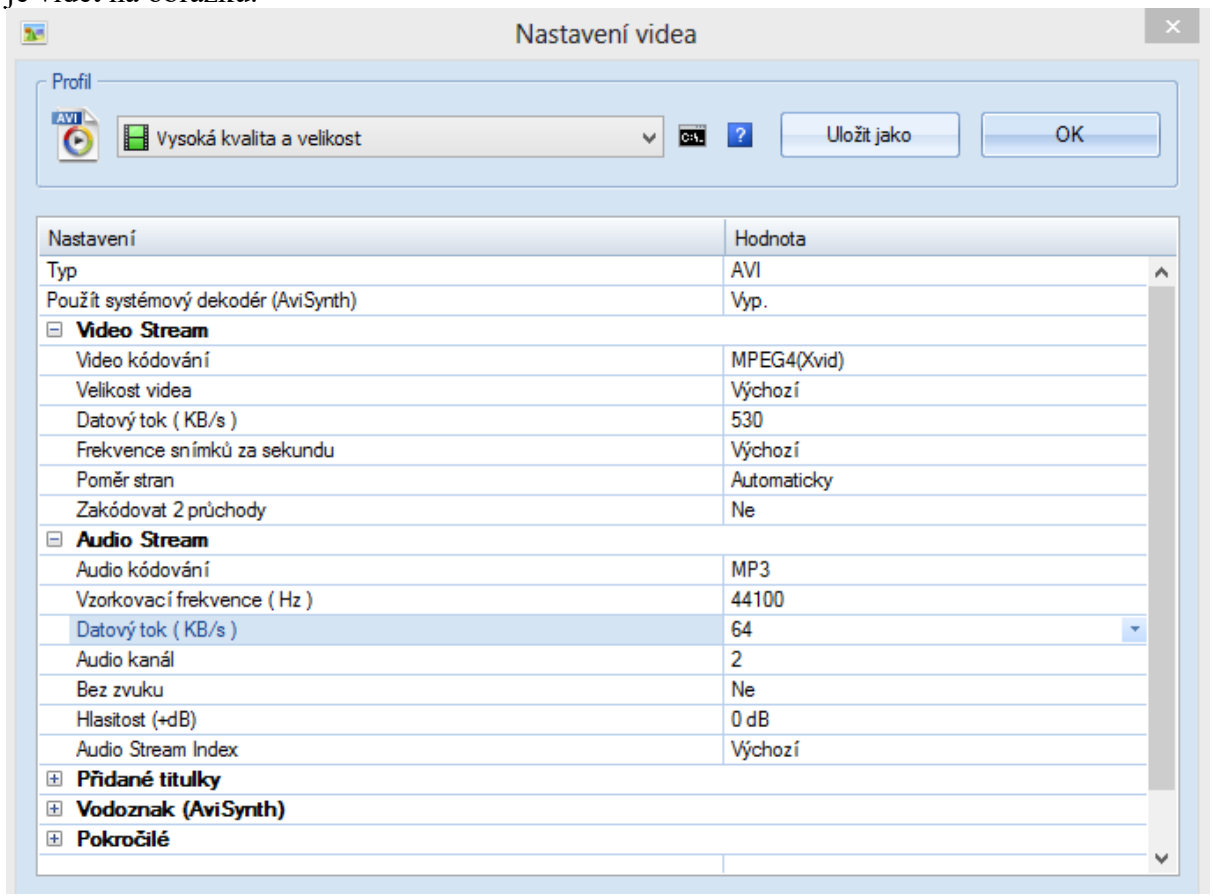


Velmi rozšířeným souborem především pro stahování je rar. Rar je již nutné vytvořit pomocí aplikace, bud Winrar, nebo je možné užít Totalcomander. Rar má slušný kompresní poměr, ale hlavně data mohou být doplněna o opravný či ověřovací kód, který zaručuje celistvost dat v archivu. Další výhodou raru je dostupnost alespoň části dat i v případě poškození archivu. Při vytváření archivu je možné nastavit nejen kompresní poměr, ale také je možné archiv rozdělit na několik menších o definované maximální velikosti, což je vhodné pro publikování větších souborů. Na rozdíl od většiny ostatních kompresních metod umožňuje přidat do archivu spustitelný algoritmus sloužící k jeho rozbalení. Problémem u samorozbalovacích archivů je náchylnost na viry, náchylnost na poškození archivu a také mírné zhoršení kompresních poměrů.

5.1.3. Komprese multimédií

Multimediální soubory lze komprimovat pomocí úpravy datového toku. Nutno upozornit, že změna datového toku je nevratná, a i po nastavení původní hodnoty nedojde k obnově původního stavu souboru. Tyto komprese využívají nedokonalosti lidských smyslů a omezení výstupních zařízení. K provedení takovéto komprese je nutné mít dostatek místa na disku, protože vznikne komprimovaná kopie originálního souboru, kterou pak můžeme v původním adresáři nahradit originální soubor. Před tím, nežli se rozhodneme nahradit originál komprimovanou verzí, tak bychom měli zkontrolovat komprimovanou verzi a ověřit, že při přehrávání nedochází k žádným potížím. Pro tento typ komprese je nutný externí program tyu Formát

Factory. Pokud máme film, který budeme přehrávat v PC tak doporučuji datový tok audia změnit na 64 kB/s a datový tok obrazu na 560 kB/s. doporučené nastavení pro účinnou kompresy je vidět na obrázku.



Pokud cheme poslouchat trošku kvalitní mp3, tak doporučuji použít datový tok 96 kB/s, pokud máme zařízení s nedostatkem paměti, tak můžeme použít i stejné nastavení jako pro zvukou stopu videa.

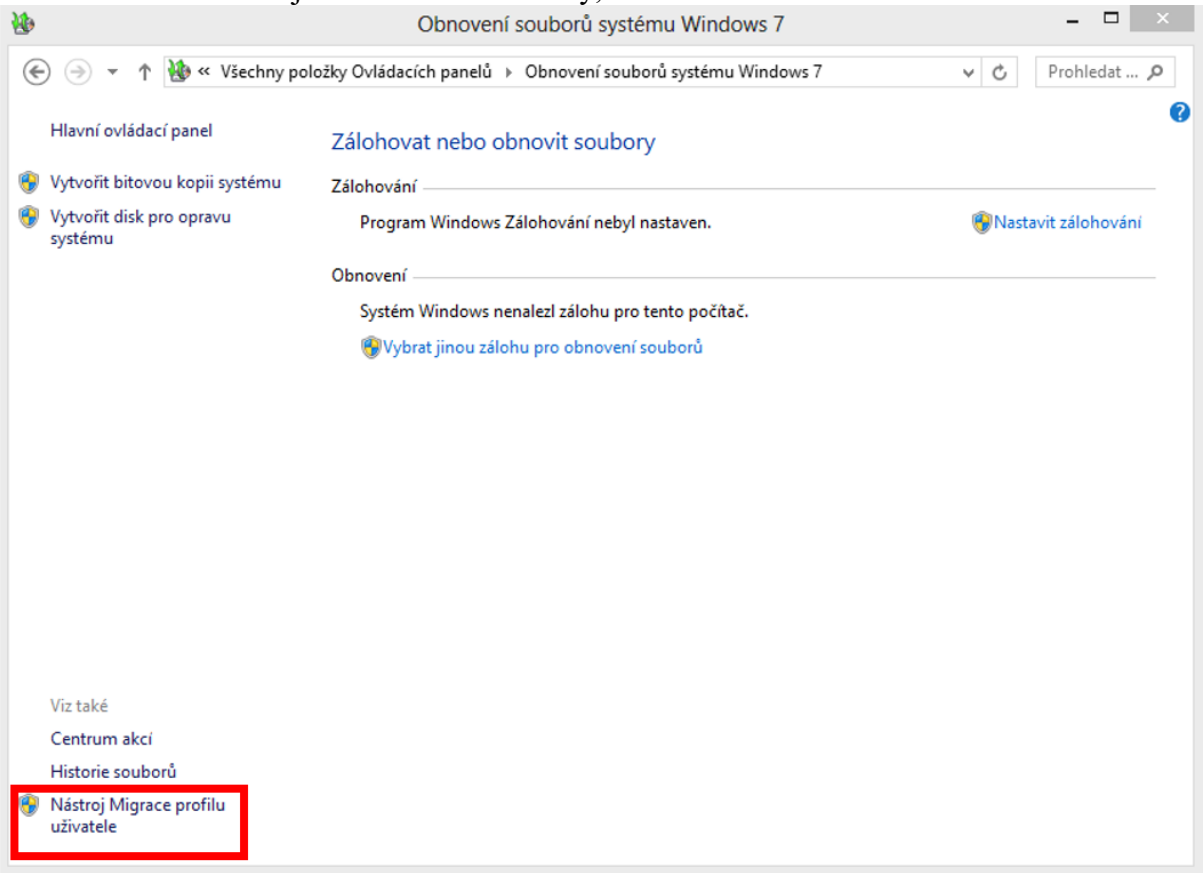
5.2. Zálohování

Systémy Windows od verze NT 6.1 mám k dispozici dva mechanismy zálohování. Windows 8.1 přinesl další, třetí mechanismus zálohování, nicméně my se budeme zaobírat první dvojicí. Jen pro pořádek bych měl uvést, že vytvoření bitové kopie systému je podporováno a verze systémů Windows 8 a 8.1 generují úspornější soubor vhdx, ale obnova systému z této bitové kopie pomocí install disku není zcela ideální a často končí chybou. Tato varianta zálohování je vhodná, pokud je k dispozici virtualizační nástroj, který umožní takto vytvořenou zálohu spustit jako virtuální stroj. V dalším textu se zaměříme na hlavní dva hlavní zálohovací mechanismy, které můžeme použít pro zálohu souborů uživatele, nastavení systému a programů. Zálohovat doporučuji cca 1 až 4 krát za měsíc, dle množství provedených změn a vytvořených souborů, samozřejmě je potřeba aktualizovat poslední, nebo vytvořit novou zálohu před větším nebo rizikovým zásahem do OS.

5.2.1. Migrace profilu uživatele

Tento soubor umožňuje zálohovat nastavení systému Windows (tapeta, spořič, nastavení napájení, obrázek účtu, písmo atd.) a soubory uživatele spolu s některými soubory aplikací. Soubor migrace uživatele umožňuje pomocí manažeru obnovit jen vybranou složku samostatně, ale není možné konkrétní adresář nebo soubor otevřít přímo v záložním souboru, na

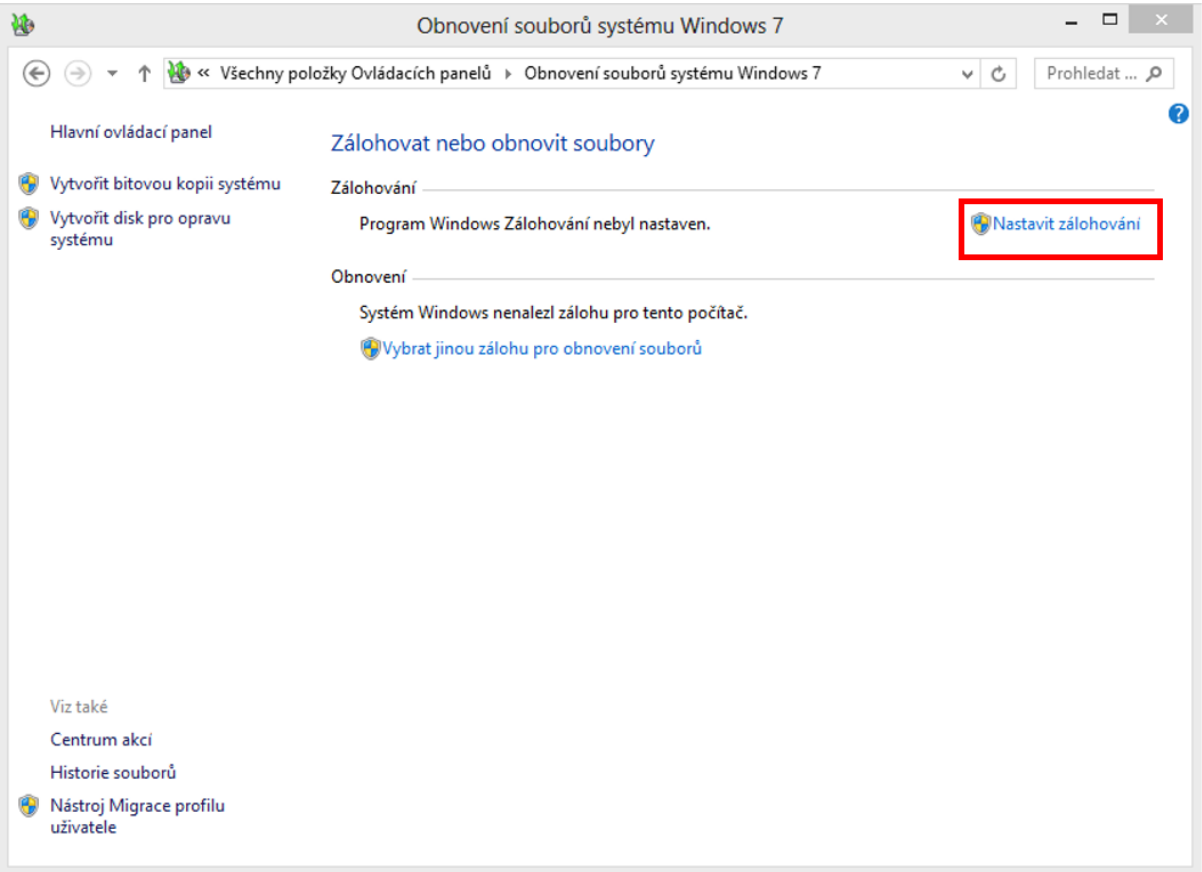
rozdíl od bitové kopie systému, která to na platformě Windows 8 dovoluje, dokonce je možné tento soubor měnit. Soubor migrace profilu uživatele je novinkou od verze Windows 7, nicméně není vhodný pro přechod mezi verzemi OS, i když z integrovaných podob záloh je asi nejvhodnější, pokud bychom chtěli libovolně přecházet mezi podporovanými systémy, tak bychom mohli zálohovat jen uživatelské soubory, nikoli nastavení OS.



Následně postupujeme dle průvodce a necháme rozhodnout systém o tom, co bude zálohováno, maximálně doporučuji zkontrolovat, zda systém zahrnul všechny uživatelem vytvořené dokumenty. Je nutné mít v cílovém umístění dostatek místa, a hlavně nedoporučuji zálohu heslovat, a když, tak totožným heslem jako počítač, protože heslo zálohy není možné změnit, a hlavně dost často jej zapomenete, protože zálohu člověk nepotřebuje každý den.

5.2.2. Záloha staré platformy

Tato záloha je jen neustále vylepšovaným mechanismem a od Windows Vista se na tomto mechanismu již dále nepracovalo, nepracovalo, soubor, který nese data dostal nový formát a novou kompresi. Tato metoda zálohování již není v systému Windows 8.1 dostupná, takže pokud potřebujeme obnovit uživatelské soubory z této zálohy, musíme užít průzkumníka Windows, nicméně ji není možné vytvořit. Tento systém vyžaduje dostatek místa, lze říct, že zabrané místo musí být o něco menší nežli volné místo na zálohovací jednotce. Zálohu je možné také vypálit na několik DVD, nicméně to nedoporučuji. Tento systém zálohy umožňuje obnovu pomocí instalačního disku systému Windows. V systémech Windows 8 a 8.1 je možné z tohoto typu zálohy vykopírovat libovolný uživatelský soubor pomocí průzkumníka, ve starších systémech to nelze. V průběhu zálohování nedoporučuji počítač používat, takže je ideální tento proces realizovat v době spánku uživatele. Při zálohování postupujeme podle průvodce, kterého ve Windows 8 nalezneme obdobně jako na následujícím obrázku.



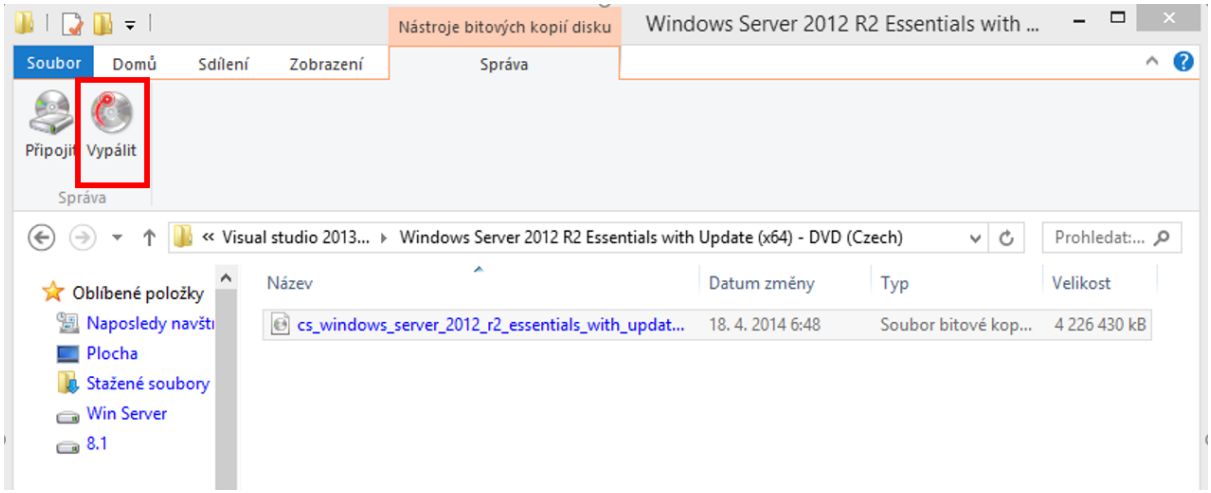
5.3. Vypalování

Jednou z variant, jak sdílet soubory a zálohovat je jejich vypálení na CD či DVD. Vypálením na DC či DVD je vhodné přenášet hudbu do automobilů. V zásadě máme dvě možnosti, jak vypálit disk, datový a multimediální (audio CD či film na DVD). Nejčastěji vypalovanými datovými disky jsou instalační média OS a aplikací.

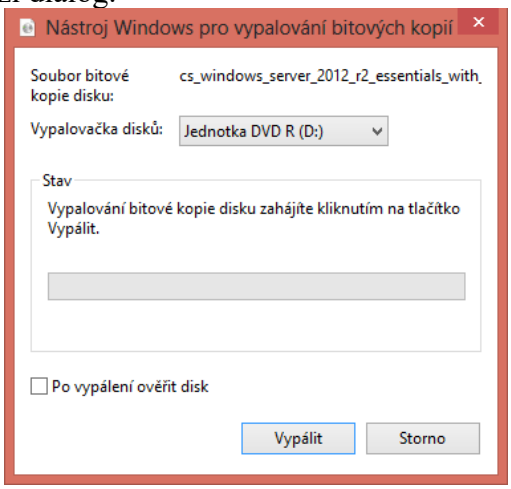
5.3.1. Datové disky

Ve Windows 8 a 8.1 máme více možností, jak vytvořit datový disk. Datový disk vytváříme hlavně pokud vypalujeme instalační disk nějakého programu. Pokud chceme vypálit například nějakou složku s fotografiemi, tak také vytváříme datový disk. Nejprve se seznámíme s vypálením formátu iso, ve kterém si nejčastěji stáhneme obraz instalačního disku ze stránek vydavatele SW. Ve Windows 7 vypálení provedeme dvojklikem na soubor formátu iso a postupem dle průvodce. Při vypalování musíme mít na paměti, jak je daný instalační soubor velký a podle toho zvolit správný optický disk.

Pokud užíváme systém Windows 8 či 8.1, tak postupujeme velmi lehce pomocí pravého tlačítka a menu v něm dostupném, nebo lépe pomocí záložky správa, která se nám zobrazí po označení iso souboru.

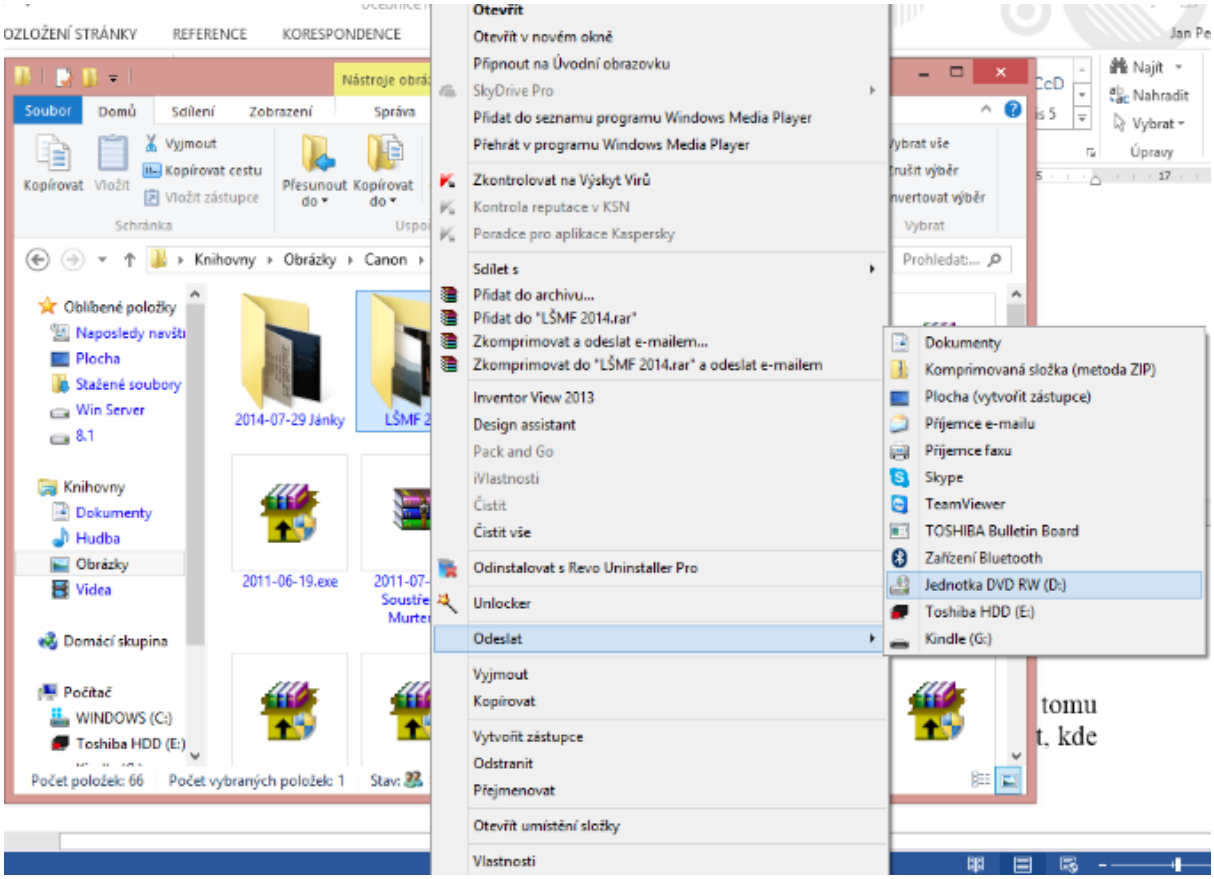


Následně se nám zobrazí dialog:

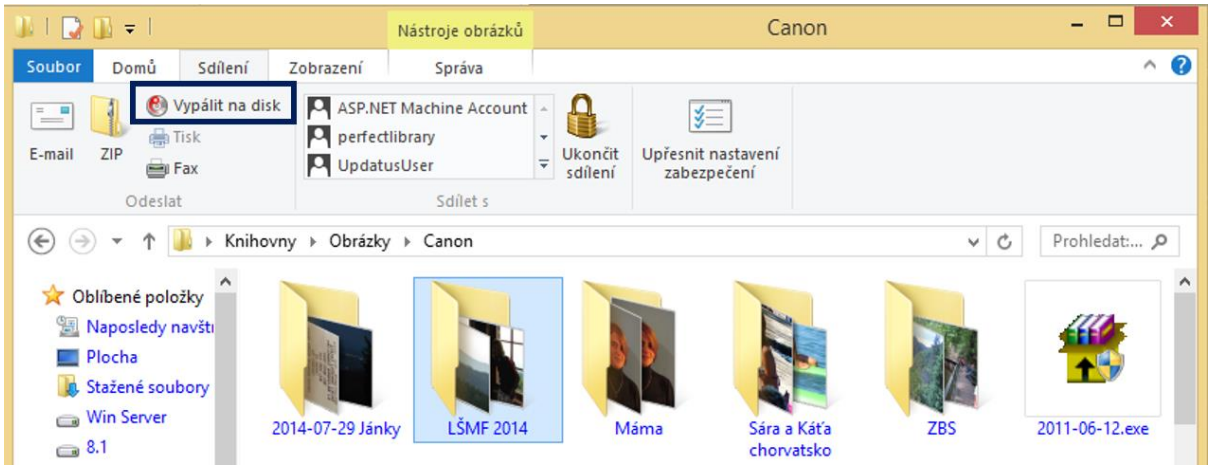


Kde po vložení disku do jednotky klikneme na tlačítko vypálit, dále jen čekáme.

Pokud chceme vypalovat vlastní data, tak máme možnost postupovat stejně jako tomu bylo v systému Windows XP, Vista a 7, tedy pomocí pravého tlačítka a nabídky odeslat, kde zvolíme mechaniku DVD.



Nebo můžeme využít možností systému Windows 8 a 8.1, tedy využít nabídku karty sdílení.



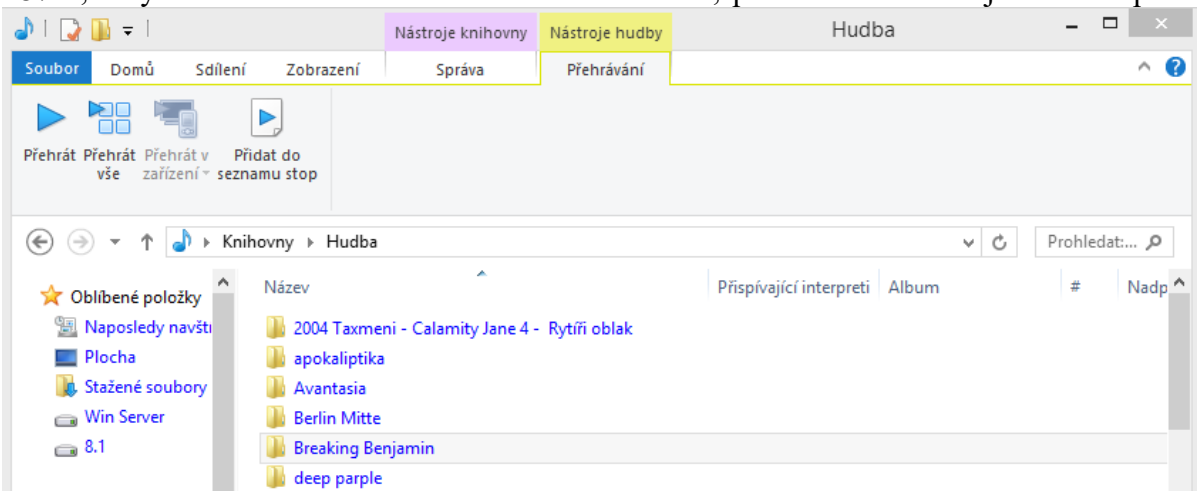
Další postup je již dle průvodce a je velmi jednoduchý, napovím, že na příslušné kartě najdeme možnost dokončit vypalování, nicméně pokud jste již se systémem něco zkoušeli, tak víte, že lze dokončit i pomocí pravého tlačítka myši na jednotku DVD.

5.3.2. Multimediální disky

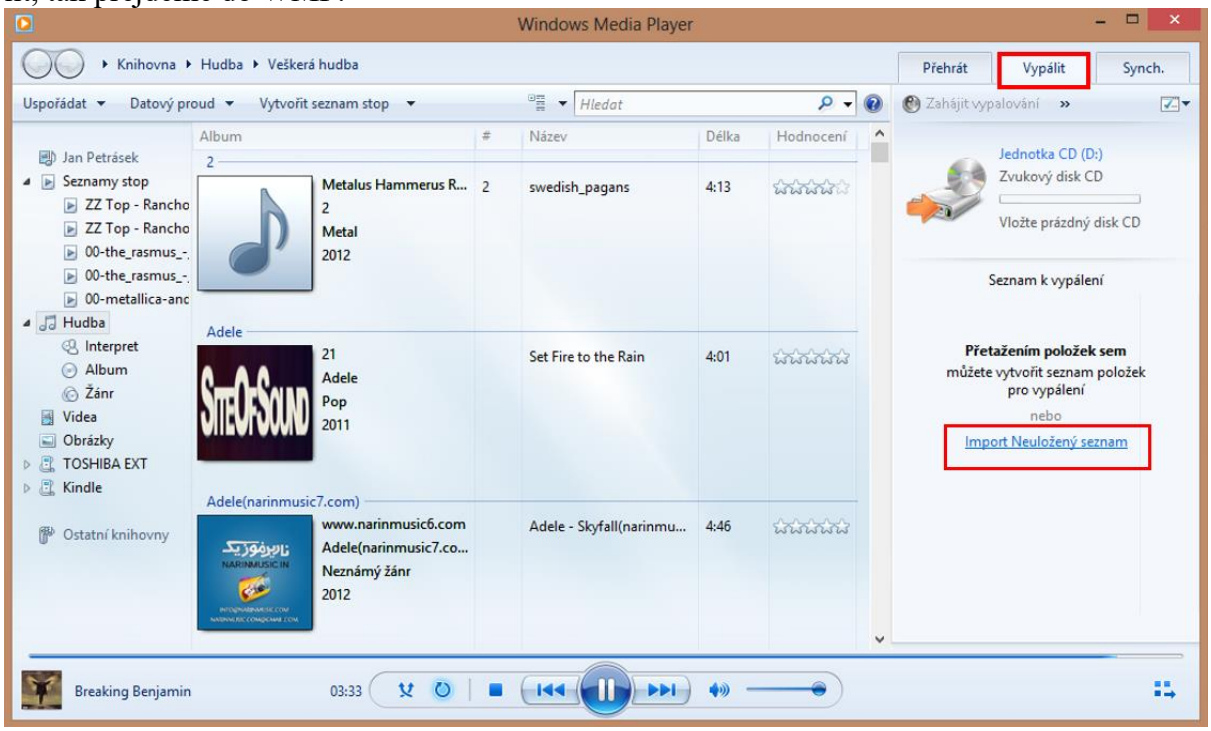
Zde je velmi důležité rozlišit, jestli vypalujeme video, nebo audio, protože to je kritérium pro výběr optického disku. Ve Windows 8 a 8.1 je možné provést jednu malou botu, kterou si ušetříme práci a umožní nám provést snadné kombinování z více zdrojových složek, ale hlavně, vyvarujeme se možné záměny typu disku. Pokud vypalujeme multimediální disk pomocí systému, tak systém musí obsahovat Windows Media Player. WMP umí mimo vypálení požadovaného seznamu provést také jeho rozdělení na více disků, pokud je to potřeba a zobrazit, jaké množství z disku bude využito, nutno podotknout, že zde na rozdíl od datových

disků neplatí přímá úměra mezi velikostí souborů a velikostí disků, zde hraje svou roli také celková délka vypalované stopy.

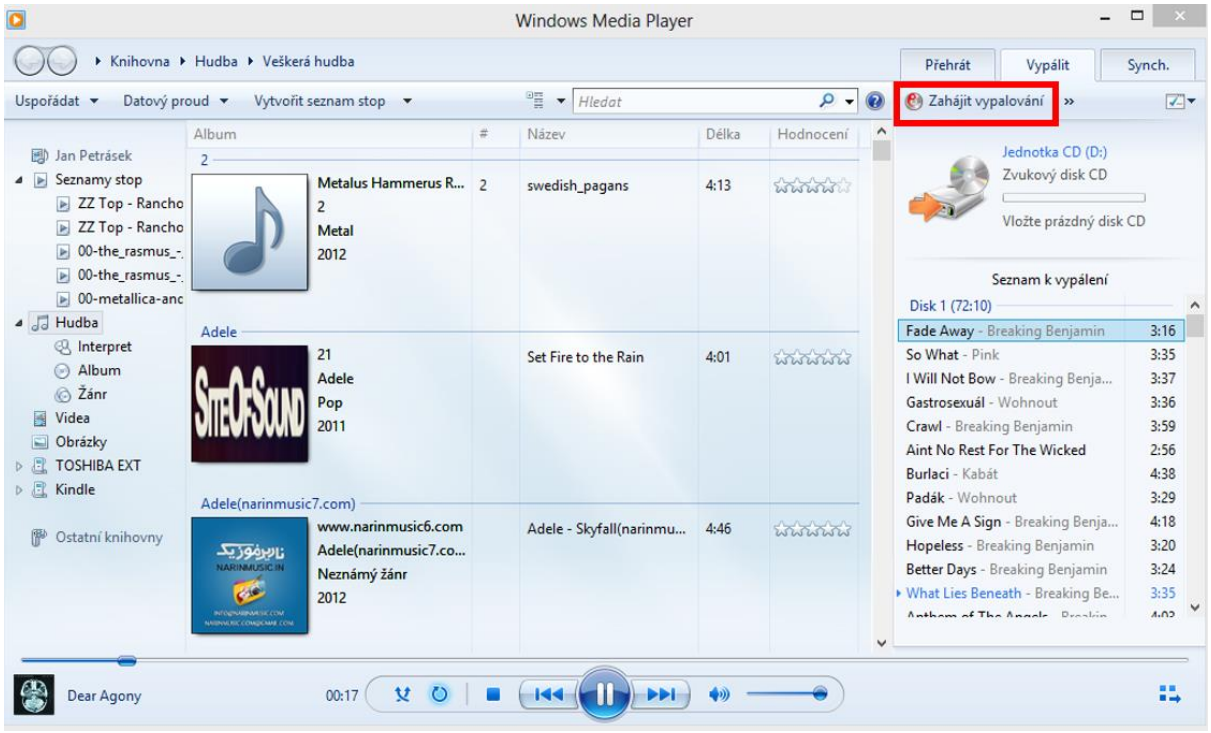
Nyní si pojďme ukázat, jak tedy na to, je důležité, aby soubory byly ve správné knihovně, tedy hudba v hudbě a videa v knihovně videa, protože to usnadňuje celkově práci.



Na kartě přehrávání zvolme přehrát, pokud máme takto zaneseno vše, co chceme vypálit, tak přejdeme do WMP.



Zde na kartě vypálit zvolíme importovat neuložený seznam.

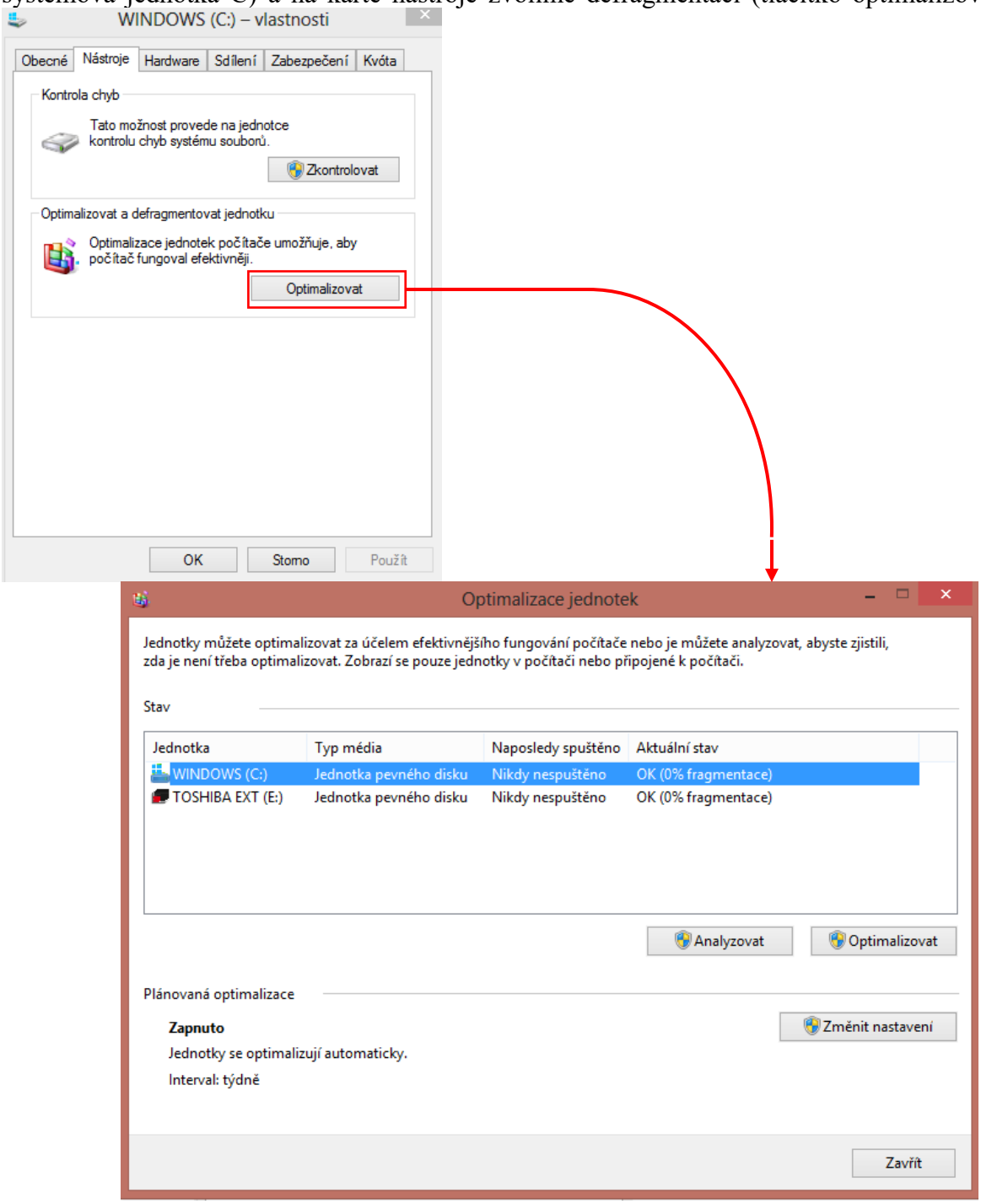


Následně stačí jen stisknout tlačítko zahájit vypalování.

5.4. Defragmentace

Jednou z vad souborových systémů je fragmentace jednotky. Soubor se na jednotku ukládá rozdělený na clustery (viz HDD) a vlivem přesouvání, úprav, mazání, zapisování atd. dochází ke fragmentaci. Fragmentace vznikne nesourodým uložením souboru, tedy soubor není uložen v po sobě jdoucích clusterech, ale na různých lokalitách disku. Souborový systém se snaží ukládat soubory do po sobě jdoucích clusterů, ale při práci s nimi dojde ke fragmentaci. Defragmentace je tedy proces, který vyhledá na jednotce všechny části souboru (fragmenty) a seskládá je na po sobě jdoucí clustery, což provede s celým obsahem disku. Výsledkem aplikace tohoto procesu je zrychlení počítače, protože systém nemusí vždy hledat jednotlivé fragmenty. Defragmentaci je vhodné provést jednou za 1 až 2 roky, pokud ji budeme používat příliš často, tak se vystavujme HDD velkému namáhání.

Postup je zcela jednoduchý, opět vyjdeme z vlastností požadované jednotky (nejčastěji systémová jednotka C) a na kartě nástroje zvolíme defragmentaci (tlačítko optimalizovat).



Provedeme výběr jednotky, protože je možné optimalizovat několik jednotek v jeden okamžik.

5.5. Ukládání souborů

Pro ukládání je důležité, abychom všichni respektovali knihovny. Ukládání souborů na plochu je ***nepřípustné!!*** Na plochu patří jen zástupci na programy i složky a soubory. Soubory běžných dokumentů (pdf, xps, doc, dosx, pptx, ppt, xlsx…) ukládáme zásadně do Dokumentů. Obrázky všech běžných rastrových formátů (jpg, jpeg, png, tif, gif…) ukládáme zásadně do

knihovny Obrázky, kde jsou nástroje pro prohlížení, práci s pozadím plochy apod. Hudení soubory (mp3, wav apod.) ukládáme výhradně do knihovny Hudba, kde jsou již připraveny nástroje pro přehrávání, tvorbu a rozšiřování play listů. Filmy a další videa (avi, mkv, mp4…) ukládáme do knohovny Videa, kde jsou rovněž nachystány nástroje pro přehrávání.

Složka stažené slouží pro dočasné uložení souborů, nebo pro soubory určené pro další sdílení a následné smazání. Soubory z této složky obvykle není potřeba zálohovat, nesmíme v ní nechávat nic důležitého, protože se pak soubory hůře hledají.

5.6. Závěrem

Tato kapitola by mohla pro spoustu z nás působit nedokončeně, ale opak je pravdou. Vytvářet složky, zástupce a nové dokumenty vás učit nebudu, protože nikdo z nás není mimino a v pásu karet je všechno na první pohled vidět. Důležité věci, které jsou částečně skryty, jsem vám ukázal.

6. Komunikace po internetu

Po internetu lze komunikovat více způsoby. Buď komunikují lidé prostřednictvím emailu, Skype a dalších sociálních sítí, sdílením souborů (což mají společné s počítači), nebo komunikují počítače, které si vyměňují informace o podobě stránky apod. My se v této kapitole zaměříme na komunikaci lidí, tedy na emaily, sociální sítě a manažery, a na konci kapitoly věnujeme pár slov také hrozbám v podobě malware.

6.1. Email

Dnes má asi každý z nás, a pokud je mezi námi někdo mladší, kdo jej ještě nemá, tak mu poradíme, jak jej založit. Email plní funkci jakési oficiální komunikace, i když úředně je oficiální komunikací využití datových schránek, což je vlastně jistá forma emailu. Pokud se ucházíme o práci, přihlašujeme se ke studiu, potřebujeme se na něco zeptat, tak často užijeme email. V dnešní době je využití emailu mezi přáteli spíše sekundární, ale jsou lidé, kteří ho využívají hojně. Běžně používáme přihlášení se k protokolům elektronické pošty pomocí webového prohlížeče, ale hlavně ve firmách jsou běžné tzv. emailové klienty (např. MS Outlook), v domácnosti používá emailového klienta většina lidí, kteří využívají nějaký program na správu emailů v chytrém telefonu, nebo uživatelé Windows 8 a 8.1 nejspíš znají aplikaci pošta, což je jedna z nejjednodušších aplikací pro správu elektronické pošty.

Když už je tu řeč o emailových klientech, tak je nutné uvést na správnou rovinu dvojici pojmů, kterou potkáte vždy při nastavování aplikace, nebo při přidávání dalšího účtu, který bude aplikace spravovat. V základu máme dvojici protokolů, IMAP, což je protokol, který vytváří odkazy a propojuje tak správu uloženou na serveru s aplikací a odesílání správ funguje tak, že protokol ji vytvoří na serveru a řídí její tvorbu. Druhou variantou je POP3, což je protokol, který stáhne do aplikace úplnou kopii správy a synchronizuje obsah, tedy co se smaže v aplikaci, to je smazáno i na serveru. Jen pro úplnost, pokud někdo využívá na outlook.com nepřetržitou práci v protokolu https, tak může mít problémy s neúplnou synchronizací POP3 a protokol IMAP je, dá se říct, nefunkční.

6.1.1. Založení

Pokud si chceme založit email, tak je dobré zakládat dvojici emailů, a každý na jiné stránce. Méně bezpečné a kapacitně nevyhovující servery jako je seznam, centrum, atlas, tiscali, post atd. je vhodné využít k založení adresy pro registrace na webu. Tento email nesmí obsahovat žádné informace přímo o vás, nebo informace, které jsou o vás nějak vypovídající (tím nemyslím barvu vlasů a očí), takže idální emailová adresa pro registrační účely je třeba bloncka117@seznam.cz (pokud jsem se někomu náhodně trefil do adresy, tak mi to prosím sdělte). I v kolonkách jméno a podobně je dobré vložit náhodnou kombinaci písmen. Pokud jde o bezpečností otázku, tak tam vybereme tuto nabídku a vyplňte tam nějaké slovo, které je vůči vám i rodině neutrální (volbu je dobré si někam uložit). Heslo by o vás také nemělo nic vypovědět, takže je ideální třeba počítej do 5 v této podobě: asdFG .

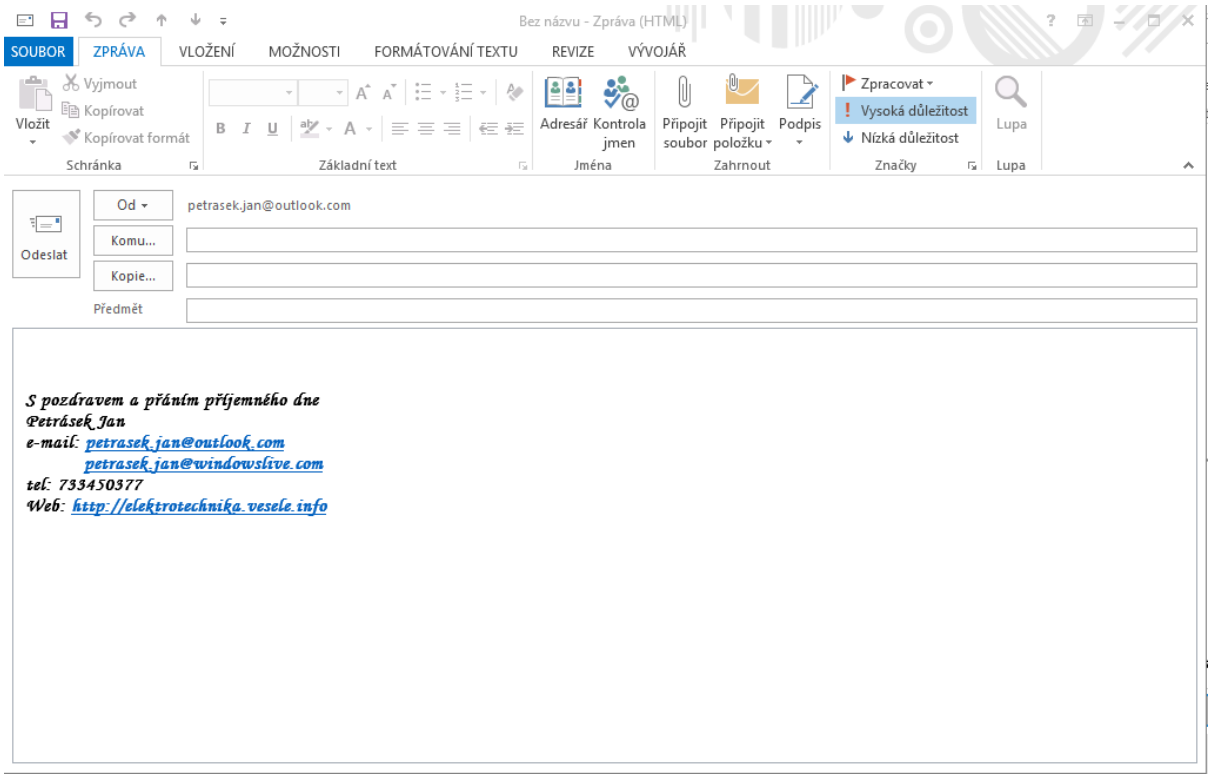
Takto vzniklá adresa je nepoužitelná pro internetové obchody a nějakou trošku seriózní komunikaci. Naopak tato nic nevypovídající adresa se hodí na sociální sítě.

Naproti tomu umístit někam svůj hlavní email je věda. Obecně je dobré zkládat na serverech, které již vaše údaje mají, tedy např.: bing.com (outlook.com, Hotmail.com), google.com (gmail.com), yahoo.com. tento email by měl již v adrese nést celé vaše jméno, nebo alespoň příjmení a iniciálu, aby bylo na první pohled vidět, kdo píše. Příkladem může být adresa: petrasek.jan@outlook.com, nebo třeba petrasekj@hotmail.cz . Chápu, že pro Nováka, Novotného a další takto frekventovaná jména je těžké založit si adresu, ale i pro ně mám dobrou správu, je možné využívat tečky, podtržítka, pomlčky, pokud je jméno po otci či matce, tak

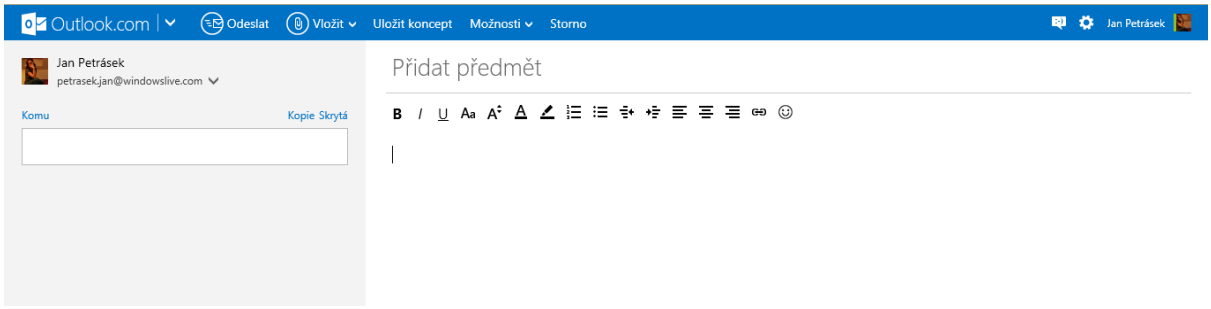
užít slůvko mladší, nebo číslici. Nicméně na rozdíl od adresy heslo nesmí nikterak souviset s vámi nebo s partnerkou/partnerem, či někým z rodiny, nebo vám hodně blízkým. Není vhodné využívat i třeba oblíbenou barvu expertnera/rky a podobně, musí to být něco neutrálního, aby to bylo těžko prolomitelné díky znalosti uživatele. Heslo také musí mít odpovídající délku, a mělo by kombinovat písmena (malá i velká), číslice a popřípadě i speciální znaky.

Zde musí být všechny povinné údaje vyplněny pravdivě, ale právě kvůli tomu se tímto emailem nikde neregistrujeme (kromě ověřených eshopů jako je alza a czc) a tuto adresu neuvádíme nikde úplně veřejně.

6.1.2. Psaní



Nebo online:



Bez ohledu na to, zda využijeme klienta, nebo použije přístup pomocí webového prohlížeče, tak zpráva musí obsahovat především adresu, na kterou má email dojít a předmět (nadpis, nebo nejvýše 8 slovný popis zprávy). Dalším důležitým bodem je oslovení a podpis, dnes jde již přednastavit podpis, takže se vloží automaticky při vytváření zprávy. Pokud na zprávu odpovídáme, tak je vhodné použít volbu odpovědět (i po delší době), čímž zůstane zachovaný předmět, akorát přibyde Re, aby bylo znát, že jde o odpověď a ještě je dobré ponechat ve zprávě předchozí komunikaci, aby adresát nemusel nikde nic hledat.

K emailu je možné přiložit soubor libovolného formátu, krom spustitelných souborů (.exe). Tento soubor by se pro kompatibilitu s firemními servery a servery typu seznam měl

vejít svojí velikostí pod 10 MB, jinak v rámci Outlook.com je to 100 MB, anebo odkazem přes One drive až 15 GB.

6.2. Sociální sítě

Sociální sítě představují nemalé bezpečnostní riziko jak pro jednotlivce, tak pro stát. Co se nedá sociálním sítím upřít, tak je jejich velký vliv na fungování současné společnosti a tím i velký hybný potenciál. Sociální sítě jsou prostředkem, který umožňuje udržet kontakt s přáteli, které nemůžete příliš vídat, ale mají také své stinné stránky. Dnes není vhodné sociálních sítí využívat k seznamování, protože zabezpečení je prakticky nulové. Vzhledem k nějaké ochraně soukromí garantované mezinárodním právem je pro administrátory velmi obtížné celou síť kvalitně monitorovat a zajistit tak dodržovaní pravidel pro danou síť. Bohužel ani v rovině šíření malware není dalece dostačující ochrana, sociální sítě se dnes v zavirovanosti rovnají většině pornografických serverů, takže dnes již neplatí, že pokud má někdo zavirovaný počítač, tak je to na 75% porno, dnes je to na 80% sociální síť. Nečekejte, že vás zde budu učit jak si vytvořit profil, nebo jak psát, abyste měli co nejvíce přátel, dám vám zde pár rad, jak být relativně v bezpečí.

6.2.1. Co do profilu nepatří?

1. Adresu bydliště
2. Mobilní telefon
3. Celé jméno
4. Školu a adresu školy + třídu / ročník
5. E-mail, který je užíván např. pro školní potřeby
6. Celé datum narození / rodné číslo /ročník
7. Čísla a typy dokladů

Pojďme se na jednotlivé body podívat podrobněji.

Co se týká adresy, tak tu nesmíme nikdy uvést celou a není vhodné uvádět ani město bez bližší specifikace, pokud se nejedná o Prahu, Brno, Ostravu, což jsou dost velká města, aby bylo reálné v nich najít podobného člověka, jinak neuvádět, nebo v případě nutnosti (údaj je povinný) uvést město, které leží ideálně na druhém konci Evropy nebo na Sibiři, k tomu je pak nutné uvést odpovídající stát.

V žádném případě na sebe neuvádíme nikde na internetu telefon, či jiný druh spojení (viz email). Toto je až příliš riskantní, ono je riskantní i přeposlání čísla v rámci konverzace, pokud toho člověka znám, tak si od něj to číslo vezmu osobně, nebo užiji email, pokud je to člověk neznámý, tak v žádném případě číslo nedávám.

Uvést školu je také velmi nebezpečné, protože pak si na vás mohou úchylové počíhat přímo u školy. Pokud uvedete třídu, tak jste úplní sebevrazi, protože tím akorát usnadníte vaše nalezení, což je vždy nežádoucí!!

Pro poslední dva body platí, nesdělujete o sobě žádné citlivé informace, datum narození je povinný údaj, proto jej vždy uveďte falešný!!, ale zase tak falešný, aby to falso bylo několik let od vás cca do 10 let, aby to nepůsobilo podezdřele. Je lepší volit starší ročník, abyste nelákali pedofily.

Registrační email uvedeme v souladu s předchozí kapitolou, musíme si dávat pozor, aby nás neprozradil.

6.2.2. Přezdívka

Přezdívka (mnohdy i přihlašovací jméno) je velmi důležitý údaj, který je za všech okolností plně veřejný a je indexovaný ve vyhledávání.

1. Nesmí naznačovat spojitost s naší osobou
2. Vhodná jsou samotná křestní jména, nebo nejlépe obecná slova (padouch, bloncka…)
3. Používat se mohou i čísla, ale nesmí to být čísla spojená s naším narozením, telefonem, adresou nebo tak

Výběr přezdívky je velké dilema vždycky, ale je nutné v první řadě myslet na svou bezpečnost, přátelům, které znáte osobně bude ukradené, jestli tam jste jako bloncka 0007 nebo jako Eva Frajerová, v adresáři si vás mohou přejmenovat a i když by nemohli, tak o nic nejde, pokud budou z osobního kontaktu vědět, že tam jste pod nějakým jménem.

6.2.3. Heslo

- Dle zásad tvorby hesel v informatice
- Mělo by obsahovat náhodná písmena a čísla (případně speciální znaky) bez spojitosti s naší osobou

6.2.4. Profilová fotografie

1. Nesmí poutat pozornost
2. NENÍ vhodné použít fotku naší osoby
3. Vhodné je užití fotografie nějakého předmětu (oblíbené hudební album, plyšák)
4. Není vhodné užívat pohyblivé gifíky, poutají moc pozornosti
5. Vhodné je publikovat fotku, která je zmenšena před publikací na malé rozměry, po zkopírování a zvětšení se rozsype (rozměr fotky na OP a menší)
6. Člověk se nesmí na fotografii poznačit
7. Majitel účtu by neměl být sám na fotce, doporučuji skupinovou fotografii
8. Nesdílejte fotografii se svým přítelem / přítelkyní
9. Nefotografujte se se ve „zranitelných polohách”
10. Dámy, nefoťte se s velkými výstřihy, či snad v plavkách, nebo spodním prádle
11. Dámy, nefoťte se tak, aby vaše přednosti poutali pozornost

Ve výše uvedených 11 pravidlech asi může být zmatek, tato pravidla platí totiž i pro fotogalerie. První 5 je nutné dodržet u profilové fotky, ostatní jsou hlavně pro fotogalerie, které nejsou veřejně přístupné. I zde platí, protože FB a další umožňují více stejných přezdívek, pokud je jiný kontaktní email, zeptejte se přátel, jak vypadá jejich profilovka, to může pomoci při přidávání přátel tak, aby nedošlo k záměně, což se může stát.



6.2.4. Statusy a sdílené stavy

- NIKDY nesdílejte svůj stav (zadaná, zasnoubená, nezadaná atd.), pokud již přeci jen no, tak vždy jen kladný (zadaná, zasnoubená) NIKDY NE SVOBODNÁ (NEZADANÁ)!!!!!!
- Do statusu NIKDY NE VĚCI TYPU: Stýská se mi po vztahu; chtěla bych někoho po svém boku
- Všeobecně by status ani stav neměl upozorňovat na to, kde nás lidé najdou a co děláme, nebo citové odkazy

- Do statusu můžeme neutrálně napsat: dnes byl dobrý den ☺; Dnes nic moc; nepsat!! nebo citáty velkých osobností: „Úspech stíhace nezávisí pouze na tom, jak dokonalý má letoun, ale především na útocném duchu zrozeném v jeho chrabrém srdci.“ Adolf Galland

Obecně se dá říct, že status může něco vypovědět o naší náladě, ale je potřeba aby ta výpověď byla realizována neutrálními prostředky, které nemají přímou spojitost s naší osbou.

6.2.5. Kontakty

1. NIKDY SI DO KONTAKTŮ NEPŘIDÁVEJTE LIDI, KTERÉ FYZICKY NEZNÁTE!!!!!
2. Pokud, chcete sociální síť užít k seznámení se s novými lidmi, je to možné, ne všichni jsou zlí, ale je nutné si toho člověka prověřit a je dobré si kontaktní údaje propůjčit až po delší době komunikace, na psanou nabídku společného odpoledne někde, nebo nějaké procházky, odmítnout, toto realizovat na základě nějakého tel. pohovoru a vzít s sebou kamarádku
3. K seznamování užijte specializované portály!!, ale ani ty nejsou bohužel příliš důvěryhodné

6.2.6. Co dělat, když se přeci jen setkáte s úchylem, nebo vás někdo začne vydírat???

1. NEBÁT SE SVĚŘIT, je možno se svěřit výchovnému poradci, či v rodině, nebo na lince bezpečí, nebo i jinému učiteli, kterému důvěřujete
2. NAHLÁSIT NEVHODNÉ CHOVÁNÍ ADMINISTRÁTOROVI
3. OZNAČIT ČLOVĚKA JAKO SPAMERA
4. POUŽÍT NÁSTOJ PRO IGNORACI A NASTAVIT NEDOSTUPNOST
5. ULOŽIT SI JEHO KONTAKTNÍ ÚDAJE A NEREÁGOVAT
6. VYHÝBAT SE MÍSTŮM, O KTERÝCH HODNĚ PÍŠE (OKOLOÍ JEHO ŠKOLY, BYDLIŠTĚ)
7. NA SMLUVENOU SCHŮZI NEPŘIJÍT
8. NIKDY NECHODIT SAMA!!!!
9. Pokud se vše stupňuje, tak si založit nový profil, který nebude dohledatelný dle starého a starý nechat zrušit
10. Po poradě s pedagogem nebo linkou bezpečí oznámit vše na policii

6.2.7. Závěrem

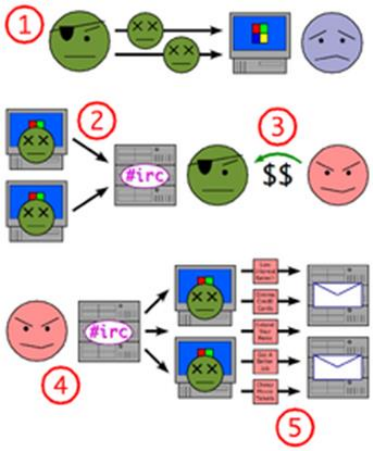
1. Řiď se heslem důvěřuj, ale prověřuj!!!
2. Nebuď důvěřivá/důvěřivý a komunikuj jen s lidmi, které znáš
3. Nezmiňuj svůj intimní život – ani na seznamce či vlastnímu příteli
4. Neprovokuj žádným způsobem
5. Buď opatrná/opatrný
6. Pokud možno nesdílej, co není nutné!!!
7. http://www.youtube.com/watch?v=c6VkPQEl74s&feature=fvwp
   Pamatujme si, že žádný profil nelze zcela odstranit, internet totiž nezapomíná.

6.3. Malware

V úvodu si v odrážkách popíšeme malware a na konci se malounko rozepíši o ochraně před malware, což je velmi podstatná záležitost pro nás uživatele. Užiji starší data, která jsem již dříve zpracoval, protože je irevalentní řešit, jaký druh malware je zrovna v kurzu.

6.3.1. Viry

- Život tzv. „zombie“: (1) Autorovi viru se podaří infikovat uživatelské PC. (2) Viry na napadených počítačích se průběžně přihlašují k IRC serveru a vytváří síť napadených počítačů, tzv. „botnet“. (3) Spammer zaplatí autorovi viru za poskytnutí vzniklé sítě. (4) Po IRC kanálu pošle spammer virům instrukce a (5) napadené počítače slouží k rozesílání spamu.



- Jako virus se v oblasti počítačové bezpečnosti označuje program, který se dokáže sám šířit bez vědomí uživatele.
- Pro množení se vkládá do jiných spustitelných souborů či dokumentů, nebo do html, které se stahuje do RAM při otevírání webu.
- V souladu s touto analogií se někdy procesu šíření viru říká nakažení či infekce a napadenému souboru hostitel.
- Viry jsou jen jedním z druhů tzv. malwaru, zákeřného softwaru. V obecném smyslu se jako viry (nesprávně) označují i např. červi a jiné druhy malwaru.
- Zatímco některé viry mohou být cíleně ničivé (např. mazat soubory na disku), mnoho jiných virů je relativně neškodných
- U některých virů se ničivý kód spouští až se zpožděním (např. v určité datum), což se někdy označuje jako *(logická) bomba*.
- Nejdůležitějším negativním důsledkem šíření virů je však samotný fakt jejich reprodukce, která zatěžuje počítačové systémy
- Viry se na rozdíl od červů samy šířit nemohou.
- Dnes (2007) jsou klasické počítačové viry na jistém ústupu oproti červům, které se šíří prostřednictvím počítačových sítí, hlavně Internetu.
- Některé antivirové programy se proto snaží chránit počítač i před jinými, nevirovými hrozbami.
- Z klasických virů je stále hodně populární Trojský kůň, jehož verze přibývají geometrickou řadou

6.3.1.1. Druhy Virů

Základními typy hostitelů jsou:

- Spustitelné soubory – COM a EXE
- Boot sektory disket a diskových oddílů.
- Master boot sektor (MBR) pevného disku.
- Dávkové soubory a skripty – BAT v DOSu, shellovské skripty na UNIXech.
- Dokumenty, které mohou obsahovat makra – např. dokumenty MS Office.
- Specializované skripty některých konkrétních aplikací.
- Obrázky a zvuková a filmová data (formáty avi a wmv jsou relativně bezpečné)

Podle různých aspektů způsobu činnosti:

- Rezidentní/nerezidentní viry
- Vir se může šířit dvěma základními způsoby: buď se ve chvíli spuštění hostitele (tzn. ve chvíli, kdy se při spouštění hostitele spustí kód viru) rozšíří do nalezených nenakažených souborů, nebo se pouze uloží do operační paměti počítače, ve které zůstane až do doby vypnutí počítače, a mezitím infikuje soubory (nebo např. diskety), se kterými uživatel pracuje.
- První typ se označuje jako nerezidentní vir, druhý jako rezidentní vir.

Stealth viry

- Stealth viry se snaží zamaskovat svou přítomnost v souboru tím, že se zachytí na přerušení, kudy prochází veškeré požadavky na čtení dat ze souboru (tedy i požadavky antiviru). Vir si pak kontroluje, zda se požadavek týká i infikovaného souboru, v tomto případě pak vrátí aplikaci data původního neinfikovaného souboru.
- Tato technika krytí se převážně týkala operačního systému MS-DOS, pro modernější operační systémy je nutno použít složitějších rootkitů (maskovacích zařízení)

6.3.1.2. Trocha historie

- V roce 1983 sestrojil jistý Dr. Frederik Cohen první samo množící program, který začal označovat jako virus.
- První skutečný vir, který mohl nějak uškodit, naprogramovali v roce 1986 bratři Basid a Amjad Farooq Alvi, pojmenovali jej Brain (mozek). Objevil se 19. ledna 1986. Sice útočil jen na na určitou část disku, ale na starších počítačích způsobil větší škody.
- Vytvářejí je programátoři velkých softwarových firem, kteří byli propuštěni ze zaměstnání. Ti se svým zaměstnavatelům pomstí vytvořením nějakého viru.
- Vytvářejí je mladí programátoři, kteří si chtějí vyzkoušet své schopnosti.
- Viry vytvářejí programátoři softwarových firem, které vytvářejí antivirové programy, za účelem zvýšení prodeje svých výrobků.
- Viry jsou jednou z cest, jak ovládnout a řídit větší množství počítačů a využívat je např. k rozesílání spamu.

6.3.1.3. Ochrana před viry

- Některé operační systémy (např. Linux, BSD aj.) jsou vůči škodlivým kódům méně náchylné, tudíž pro svou vlastní potřebu antivirové programy prakticky nepotřebují (výjimkou jsou servery, které tyto programy používají zejména na ochranu svých klientů)
- Brány firewall a antivirové programy, upgrade OS

6.3.2. Spyware

- je program, který využívá internetu k odesílání dat z počítače bez vědomí jeho uživatele. Někteří autoři spyware se hájí, že jejich program odesílá pouze data typu přehled navštívených stránek či nainstalovaných programů za účelem zjištění potřeb nebo zájmů uživatele a tyto informace využít pro cílenou reklamu.
- Existují ale i spyware odesílající hesla a čísla kreditních karet nebo spyware fungující jako zadní vrátka. Protože lze jen těžko poznat, do které skupiny program patří, a vzhledem k postoji k reklamě řada uživatelů nesouhlasí s existencí a legálností jakéhokoliv spyware.
- Spyware se často šíří jako součást shareware, a to jako adware nebo bez vědomí uživatelů (ale s vědomím autorů programu). Jakmile si takový program nainstalujete a spustíte, nainstaluje se do systému také spyware. Často se to týká například klientských programů pro peer to peer sítě umožňující stahování hudby a videa od ostatních uživatelů.

Příznaky výskytu spyware

- Nežádoucí domovská stránka (Přesměrování na jinou webovou stránku)
- Pomalý start počítače a dlouhé nabíhání internetu
- Při surfování na internetu ve zvýšené míře vyskakují reklamy
- Přesměrování telefonní linky – u vytáčeného připojení – Dialery
- Padající Windows (Častý restart, chyby, apod.)
- Nové ikony na ploše, které se záhadně objevují

6.3.2.1. Druhy

- **Adware** – obtěžují při práci na počítači reklamou
- **Browser helper object** – dll knihovna, která umožňuje programátorům změnit a sledovat Internetové prohlížeče
- **Hijacker** – mění domovskou stránku
- **Dialer** – přesměrovává telefonní linku na drahé telefonní tarify
- **Keystroke Logger** – sleduje každý pohyb na klávesnici, některé druhy odesílají uživatelova hesla
- **Miscellaneous** – je to směs spyware
- **Remote Administration** – umožní vzdálenému uživateli ovládat PC

6.3.2.2. Možnost šíření

- části registru systému (uživatelé často pracují pod účtem administrátora)
- IP adresu uživatele, někdy i MAC (fyzickou) adresu
- Historii prohlížených stránek
- Informace o software a multimediálních souborech, které jsou na počítači
- Seznam otevíraných souborů
- celé dokumenty
- uživatelova hesla
- Neprohlížet internetové stránky s podezřelým obsahem (pornografie)
- Při surfování používat bezpečnější internetový prohlížeč
- Používat antispyware
- Provádět aktualizace systému
- Používat firewall
- Neinstalovat podezřelé programy

6.4. Ochrana před malware

Jako se malwarové útoky stávají stále častější, tak se také pozornost začala přesouvat od ochrany proti virům a spyware na ochranu proti malware, a programům, které byly speciálně vytvořeny pro boj proti malware.

6.4.1. Anti-virový a antimalwarový software

Specifická komponenta anti-malwarového, či anti-virového softwaru obyčejně označovaného jako on-access nebo real-time skener se obecně připojí hluboko do jádra operačního systému nebo do funkcí jádra podobným způsobem jako to dělá malware sám, ale s uživatelským informovaným souhlasem chránit systém. Kdykoli operační systém přistupuje k souboru, on-access skener kontroluje zda jde o legitimní soubor nebo ne. Pokud je soubor skenerem považován za malware, tak je operace přístupu zastavena a soubor je skenerem řešen v před definovaném stavu (tak jak byl anti-virový program nastaven během nebo po instalaci) a zároveň o tom obeznámí uživatele. Toto obecně zpomaluje operační systém a/nebo spotřebovává velké množství operační paměti, v závislosti na tom, jak je anti-virový program nastaven. Cílem je zastavit všechny operace malwaru pokusit se usadit do OS, než k nim dojde, včetně

dalších činností, které by mohly využívat chyby operačního systému nebo spustit neočekávané chování operačního systému.

Anti-malwarové programy mohou bojovat proti malwaru dvěma způsoby:

1. Mohou poskytnout ochranu v reálném čase proti instalaci škodlivého softwaru na počítači. Tento typ ochrany funguje stejným způsobem jako anti-virová ochrana. Anti-malwarový software skenuje všechny příchozí síťová data a blokuje jakékoliv hrozby.

2.Anti-malwarové programy mohou být použity pouze pro detekci škodlivého softwaru, který již byl nainstalován do počítače. Tento typ anti-malwarového softwaru kontroluje obsah registru systému Windows, soubory operačního systému a nainstalované programy v počítači a poskytne seznam všech nalezených hrozeb. Umožňuje uživateli vybrat, které soubory chce odstranit či zachovat nebo tento seznam porovná se seznamem známých malware komponent.

Real-time ochrana před malware pracuje identicky k real-time antivirové ochraně: software skenuje soubory na disku v době stahování a blokuje ty aktivity komponent, které jsou známé jako malware. V některých případech také může zachytit pokus o nainstalování start-up položek nebo o úpravu nastavení prohlížeče. Protože mnoho malwarových komponent je nainstalováno po zneužití prohlížeče nebo chybou uživatele, pomáhá prohlížeč bezpečnostním softwarem „sandbox“ (odděluje prohlížeč od počítače a tím žádný malware nemůže provést změny) s účinným omezením jakékoliv škody, kterou by mohl malware napáchat.

Příklady antivirových programů od Microsoft Windows a anti-malwarových softwarů zahrnují Microsoft Sucurity Essentials (WinXp, 7) pro ochranu v reálném čase, Nástroj pro odstranění škodlivého softwaru systému Windows a Windows Defender. Navíc je možné si instalovat antimalwarový SW od jiných vydavatelů (Kaspersky, Norton, Eset).

Některé viry vypínají funkci Obnovení systému a další nástroje systému Windows, jako je Správce úloh a MS-DOS. Takové viry lze odstranit restartováním počítače a spuštěním Nouzového režimu s prací v síti a použitím systémových nástrojů nebo pomocí Microsoft Safety Scanner.

Nejlepší odstranění malware nabízejí boot nástroje jako je Kaspersky resue disk, akorát je občas potřeba opravit boot sektor a je potřeba mít legální OS, protože ilegální je potřeba znovu aktivovat.

V současné době není známa žádná metoda pro rozpoznání hardwarových implantátů.

Typické malwarové produkty detekují problémy na základě heuristiky a podpisů, tj. na základě informací, které mohou být vyhodnoceny jako špatné. Některé produkty mají alternativní přístup při skenování dokumentů typu Word a PDF, a to vytvořením nového čistého souboru na základě toho, o čem víme, že je dobré z definice schématu v souboru (pro tento přístup existuje patent).

6.5. Počítačová kriminalita

Termínem počítačová kriminalita (též kybernetická kriminalita či kybernalita) se označují trestné činy zaměřené proti počítačům nebo trestné činy páchané pomocí počítače. Jde o nelegální, nemorální a neoprávněné konání, které zahrnuje zneužití údajů získaných prostřednictvím výpočetní techniky nebo jejich změnu. Počítače jen poskytují novu technologii a nové způsoby pro páchání už známých trestných činů, tedy např.: sabotáž, krádež, zneužití, neoprávněné užití cizí věci, vydírání nebo špionáž.

6.5.1. Útok na počítač, program, údaje, komunikační zařízení

Jsou v první řadě fyzické útoky na zařízení výpočetní techniky, magnetická média, vedení počítačové sítě nebo elektrického rozvodu apod. Do této skupiny činů patří také vymazání nebo pozměnění dat, formátování paměťových médií nesoucích data (pokud tak nečiní majitel dat na médiu), působení počítačových infiltrací (záměrné šíření malware), nelegální tvorba a rozšiřování kopií programů (každý člověk si může vytvořit 5 kopií, které slouží pro účely zálohy, tudíž je nesmí distribuovat (ani přítelkyni či snoubence)), získání kopie hospodářských dat, databází zákazníků, ve státních orgánech únik informací o občanech apod.

6.5.2. Neoprávněný přístup k údajům, (počítačová špionáž)

Jako počítačovou špionáž označujeme proniknutí do bankovních systémů, systémů národní obrany, do počítačových sítí důležitých institucí apod. Někdy tato činnost způsobuje přímé škody velkého rozsahu, např. nelegální bankovní operace, jakož i nepřímé škody způsobené únikem informací.

6.5.3. Šíření počítačových zpráv

Vše začíná vytvořením poplašné zprávy upozorňující na fiktivní nebezpečí. Nejčastějším motivem pachatelů této trestné činnosti je pobavit se na nevědomostí ostatních, může jít i o zprávy spojené s páchaním jiné trestné činnosti. Tyto zprávy jsou v počítačovém slangu označované jako hoax. Tento druh kriminality je velmi závažný, protože může jako jeden z mála vést přímo k ohrožení lidského života a zdraví.

6.5.4. Warez

Nejvíc rozšířenou trestnou činností je používání, šíření a vytváření prostředků na odstranění ochranných prvků sloužících pro chránění autorských děl nebo používání a šíření takto upravených autorských děl, se kterými je nakládáno v rozporu s autorským právem. Takovéto prostředky se v počítačovém slangu nazývají warez. Lidi, zacházející s nimi, bývají označování jako softwaroví piráti.

- **aplikace** – všeobecně maloobchodní verze software
- **cracky** – záplaty určené na pozměnění zkušební verze programů na plné nebo obejití protipirátské ochrany
- **keys/key generators (zkráceně keygens)** - klíče k instalačním a registračním programům nebo jejich generátory, které umožňují nelegálně nainstalovat program nebo odstranit omezení zkušební verze programu.
- **počítačové hry a hry pro herní konzole**
- **filmy**
- **hudba/mp3** – pirátská alba, singly nebo jiné hudební formáty distribuované ve formě hudebního formátu MP3
- **elektronické knihy/tutoriály** – naskenované knihy a návody k programům



6.5.5. Průnik do systému

Další rozšířenou trestnou činností je neoprávněné vniknutí do systému. Člověk zabývající se touto činností se v počítačovém slangu nazývá hacker, často tento člověk pracuje na objednávku (tehdy jde o získání nějaké informace), nebo obzvláště mladí studenti IT tuto činnost provozují, aby si dokázali svoje schopnosti.

Útok hrubou silou je metoda, která spočívá ve vyzkoušení všech možných kombinací znaků, aby bylo získán přístup. Útočník sestaví program, který se pokouší postupným vyzkoušením všech možností uhádnout správné heslo. Rozluštění takovéhoto hesla lze zabránit použitím dostatečně dlouhého hesla (při současném výkonu počítačů se doporučuje minimálně 8 (respektive 16) znaků) Důležité je použít co nejširší možný okruh znaků – malá velká písmena, čísla a speciální znaky (další symboly). Hesla nesmí být logická, nesmí být stavěná algoritmem, nebo nesmí mít něco dočinění s přáteli či příbuznými!!!!! (např.: J313M1994 = J… M……. 31.3.1994 – chybné) (Např.: S l10-h/12V*18K14C – správně (18 míst)). Heslo je potřebné také často měnit!!! (alespoň po 72 dnech). Je nanejvýš nevhodné ukládat hesla na veřejně dostupných počítačích, ve sdílených složkách, discích flasch, a uvnitř počítače.

Tento útok spočívá ve zkoušení všech slov daného jazyka. Takovémuto útoku se dá předejít tak, že se použije heslo, které není slovem žádného jazyka. Bezpečné heslo se může odvodit například takto: vezmou se první písmena věty, která se dá lehko zapamatovat: A přece se točí. Galilieo Galilei. Dostane se Aprsto-Gaga. Toto heslo se může ještě vylepšit například takto Apr100-2*Ga.

Heslo se dá velmi lehko získat odposlechem nezabezpečených komunikačních linek jako jsou http:// a ftp:// . Proto se nikdy nemá zadávat svá hesla na stránky, které jsou nezabezpečené šifrovanou komunikací https:// nebo ftps:// (popřípadě jinou).

Útočník může využít, že se zapomeneme odhlásit ze systému. Využije otevřeného spojení, které zneužije ve svůj prospěch. Některé stránky se proti takovým útokům chrání automatickým ukončením spojení při nečinnosti (proto se nedá odeslat e-mail, který píšete déle než 15 minut ve webovém prohlížeči).

Útočník sestaví program nazývaný Backdoor (zadní vrátka), který mu umožní připojit se do systému bez nutnosti poznat správné uživatelské jméno a heslo. Tento program rozšíří pomocí počítačového červa nebo trojského koně.

Útočník sestaví program nazývaný Keylogger, který zaznamenává stisknuté klávesy a takto získané údaje mu odesílá prostřednictvím internetu. Tento program se rozšiřuje pomocí počítačového červa nebo trojského koně.

6.5.6. Bankovní krádeže

Bankovní krádeže uskutečněné pomocí počítače jsou zatím u nás řídké, ale ve světě se začínají stále více vyskytovat. Známé jsou následující tři typy bankovních krádeží: Phishing, Pharming a Spoofing. Následovat bude krátký odstavec o každém ze způsobů.

Zprávy, které pod určitou záminkou nabádají ke změně osobních údajů, se odborně nazývají phishing (v překladu rybaření). V takovémto e-mailu je umístěný odkaz, na kterém si heslo má uživatel změnit. Odkaz však nesměřuje na stránku banky, ale na její dokonalou napodobeninu. Takovéto zprávy jsou většinou velmi formálně napsané. Některé dokonce vypadají tak, že je jejich odesilatelem samotná banka. Proto si vždy má uživatel ověřit pravost takovéto zprávy a neotvírat stránku přes odkaz v poště.

Nejzákeřnější způsob, kterým může hacker uživatele připravit o úspory, je pharming (farmaření). Tato metoda spočívá v přesměrování názvu www stránky na jinou adresu. Každé jmenné adrese (například ib.vub.sk) přísluší IP adresa (například 215.5.214.144). Když uživatel zadá jmennou adresu do prohlížeče, místo stránky banky se zobrazí její dokonalá napodobenina. Po zadaní údajů data získá neoprávněná osoba, která falešnou stránku vytvořila. Nejjednodušší obranou je zjistit si i IP adresu stránky internetbankingu. Stačí otevřít příkazový řádek a zadat příkaz ping adresa. Potom místo jmenné adresy do prohlížeče uživatel zadá IP adresu. Další možností je ověření platnosti certifikátu a upozornění při přechodu ze zabezpečené stránky na nezabezpečenou. Některé banky se proti takovémuto způsobu elektronického podvodu brání tak, že uživateli po přihlášení do systému pošlou SMS s kódem, který musí zadat.

Nejnebezpečnější je však metoda nazývaná MITM (man-in-the-middle v překladu „muž ve středu“). Tato metoda spočívá v narušení komunikace mezi klientem a bankou, při které útočník naruší šifrovací systém veřejného a soukromého klíče, který se používá při komunikaci. Použít metodu MITM ale není jednoduché, protože pro narušení komunikace je potřebné získání klíče (certifikátu) banky, který se často mění. Je důležité nastavit webový prohlížeč, aby ověřoval, zda je certifikát platný.

7. Počítačové sítě

Počítačová síť je vzájemné propojení počítačů (příp. dalších zařízení) umožňující jejich komunikaci a sdílení hardwaru, softwaru a dat, které slouží lidem...

Výhodami počítačových sítí je hledisko ekonomické (může ušetřit náklady na hardware, často i na údržbu a správu systémů). Užití sítě také přináší výrazné zlepšení komunikace (on-line i off-line), vyšší efektivitu a produktivitu práce, úsporu lidských zdrojů, ale především je možné dělat věci zcela novými způsoby.

Jako všechno, i počítačové sítě mají své nevýhody. Mezi hlavní patří ekonomické (počáteční náklady mohou být vyšší – je třeba sledovat dlouhodobější hledisko, aby se ekonomicky síť stala výhodnou), největší nevýhodou je však snadnější zneužitelnost (nutností je dobré zabezpečení přístupu k datům a zabezpečení komunikace) a jako jednu z nevýhod na lidských zdrojích můžeme vidět vyžadování odborníků (příp. outsourcing – pomoc specializované firmy), kteří si říkají o nemalé platy. Z technického hlediska tu také v tom našem výčtu nevýhod něco máme, síť je náchylnější k poruchám nežli jednotlivá oddělení zařízení.

Historie počítačových sítí je s počítači spjata takřka od počátku jejich užívání. V 60. letech to byly sálové počítače s terminály připojenými přes telefonní linky (tak vypadaly první sítě). V 70. letech došlo k rozvoji mikropočítačů, na což reagovaly. Tehdejší sítě měly podobu minipočítače s terminály distribuované na jednotlivá pracoviště (problémy s komunikací minipočítačů). Počátkem 80. let vstupují na scénu osobní počítače (IBM PC) s distribuovaným výkonem, nastává rychlý rozvoj síťových technologií. S příchodem 90. let nastal rozvoj globálních počítačových sítí (Internet) a progresivních síťových technologií (vyšší přenosové rychlosti, vyšší výkon, bezdrátové sítě...). Dnes nám nové technologie umožňují větší centralizaci služeb a přenos dalece vyšších objemů dat.

7.1. Základní komponenty sítí

Základní komponenty sítí můžeme dělit na HW a SW.

Hardwarové:

- uzly (počítače a karty pro síťová rozhraní),
- topologie (fyzická a logická),
- pasivní a aktivní prvky sítě,
- pomocné komponenty (periferní zařízení, bezpečnostní zařízení a nástroje);

Softwarové:

- síťové operační systémy a software (OS) pracovních stanic,
- prostředky (programové vybavení serverů a ovladače HW),
- nástroje pro analýzu a monitoring provozu sítě a pro správu HW a SW
- aplikace – síťové programové vybavení.

7.2. Dělení počítačových sítí

Počítačové sítě se dají dělit podle velkého množství hledisek. My si pro jednoduchost sítě rozdělíme dle 7 základních hledisek. Tato hlediska jsou: Přenosová rychlost, rozsah, typy uzlů, vztahy mezi uzly, topologie, architektura a možnost přístupu. Pokud bychom dělali nějakou reálnou síť, tak vždy spadá do nějaké kategorie v rámci všech 7 uvedených hledisek. Zkusme se zamyslet nad tím, proč tomu tak je? Odpověď na tuto otázku nechám na každém z nás, ale zastavme se při čtení a opravdu se nad tím zamysleme. Představme si takovou síť třeba ve škole, kam jsme chodili, v nemocnici, kde sídlí náš lékař, nebo kdekoli jinde a zkusme si tuto síť popsat pomocí uvedených hledisek.

7.2.1. Přenosová rychlost

Jak šel historický vývoj sítí, tak ruku v ruce s ním šel i vznik několika generací počítačových sítí. Přenosová rychlost je ovlivněna jak užitou logikou, tak i využitým přenosovým médiem, protože každé přenosové médium má svoje fyzikální možnosti.

První sítě měly rychlost řádově kb/s. pro přenos dat používaly analogové telefonní linky popřípadě koncem období již i koaxiální kabel. Data se zde kódovala podobně jako telefonní hovor, takže z toho vyplývalo i další omezení, kterým bylo propojení jen dvojice zařízení, nejčastěji sálového počítače s terminálem.

Další generace (Ethernet, Token Ring, ARCnet) dosahovaly již rychlostí 1-20 Mb/s. pro přenos dat se začala spolu s koaxiálním kabelem, který známe z televizních rozvodů, také užívat kroucená dvojlinka, která proti klasické má nižší vyzařování (tudíž ztrátovost přeneseného výkonu), což umožnilo přenášet větší objemy dat a dosahovat vyšší rychlostí. Vznikl mechanismus packetů, kdy data jsou rozdlělena na menší části, které jsou odesílány samostatně několika cestami a na konci opět sestavena. Pokud dojde k ztrátě packetu, tak systém požádá o jeho znovu zaslání, což umožnilo vyšší propustnost sítě se zachováním přesnosti doručení požadovaných dat. Každý packet obsahuje tzv. hlavičku, která nese informace o pořadí, odesílateli a adresátovi.

Současné sítě (Fast Ethernet, FDDI, ATM, Gigabit Ethernet) dosahují rychlosti 100-1000 Mb/s, ale v posledních letech rozšiřující se technologie CESNET2 (páteřní sítě) a Gb Ethernet mají špičkové přenosové rychlosti 2,5 až 10 Gb/s. Technologie CESNET2 je vedena výhradně pomocí optických přenosových médií a dosahuje rychlostí mezi 7 a 10 Gb/s. technologie Gb Ethernet dokáže využívat i bezdrátové mikrovlnné spoje, kde dosahuje rychlostí vůči koncovému uživateli 2,5 až 6 Gb/s. Tato technologie využívá i standartní metalické rozvody.

Další směr vývoje především v Japonsku a USA směřuje k přenosovým rychlostem až 1 Tb/s, kterých by mělo výt dosaženo na jednovidových optických linkách.

7.2.2. Rozsah

Podle rozsahu máme 4 (respektive 3) hlavní skupiny sítí.

Lokální - LAN (Local Area Network):

- v omezeném prostoru (např. místnost, budova, areál),
- většinou přímé propojení počítačů bez nutnosti směrování
- většinou homogenní prostředí
- dobrým příkladem jsou domácí Wi-Fi sítě, nebo síť uvnitř počítačové učebny

Městské – MAN (Metropolit Area Network):

- rozkládají na území města
- propojení jednotlivé LAN s páteřní sítí státu
- využívají různorodé technologie pro propojení LAN, ale výstupem je optická linka
- Prostředí není homogenní
- Zpravují i monitorování a další bezpečnostní prvky ve městě (kamerový systém, kontrola ovzduší a vod atd.)

Firemní – FAN (Fabrik Area Network):

- Propojují jednotlivé divize či vzdálená pracoviště
- Dnes jsou stále výsadou velkých firem (ČD, ČEZ, RWE atd.)
- Často využívají přenosové kapacity internetu
- Nehomogenní prostředí

Rozlehlé – WAN (Wide Area Network):

- bez prostorového omezení,
- nutnost směrování dat,

- často velmi heterogenní prostředí (různý hardware, různé operační systémy, různé druhy propojení sítí).
- Nejčastějším přenosovým médiem je optika
- Patří sem páteřní sítě jednotlivých států a kontinentů

Globální – Internet:

- Propojuje všechny WAN do jednoho celosvětového celku
- Tvoří nejvyšší postavení v hierarchii sítí
- Jako taková nelze lidskými zdroji celá vyřadit z provozu

7.2.3. Typy uzlů

Sítě mohou být tvořené:

- osobními počítači (většina LAN)
- minipočítači a mainframy (např. servery v univerzitní síti nebo ve výpočetním centru);
- jinými sítěmi (tzv. páteřní sítě, Internet – síť sítí).

7.2.4. Vztahy mezi uzly

Dle vztahů mezi uzly je třízení velmi snadné, protože máme pouze dvě možnosti, jak tyto vztahy vyjádřit.

peer to peer:

- každý počítač v síti má rovnocenné postavení vůči ostatním
- každý počítač může nabízet svá zařízení ostatním a využívat zařízení ostatních
- **Příklad:** Windows (3.11, 9x, 2000Pro), Linux, DR-DOS

klient-server:

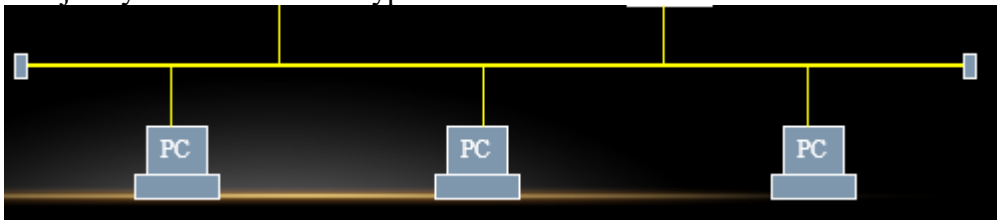
- **server** - počítač nabízející svůj hardware nebo software ostatním, řídí přístup ke sdíleným zdrojům a do sítě
- **klient** - počítač využívající hardware nebo software serverů
- server může být i **dedikovaný** (vyhrazený - nemůže fungovat jako klient) - např. Novell
- **Příklad:** Windows NT, 2000Server, Linux, Novell, IBM AS/400

7.2.5. Topologie

Toto slovo tak trošku zavání zeměpisem a mapami. Ono tmo tak trošku je i v té informatice, protože všechny počítače musí být nějak propojeny a v tom propojení musí být nějaký řád, lze jej zakreslit na tabuli či papír. Abychom popsali správně reálnou síť, tak by tyto nákresy často musely být dva, protože jedna věc je fyzické zapojení média a druhá věc je logický postoj zařízení v síti. Pokud bychom měli topologie nějak definovat, tak z pohledu informatiky jde o prostorové uspořádání a strukturu propojení počítačů a prvků sítě.

7.2.5.1. Sběrnicová

Jde o nejstarší druh topologie vůbec. Dnes se již využívá velmi vzácně, protože má dost problémů, které brání jejímu praktickému využívání v dnešní době. Realizovat lze poue pomocí metalických médií, a to buď koaxiálního kabelu (který se dnes v sítích již nepožívá), nebo pomocí dvojlinky. Jako taková síť vypadá vidíme na obrázku.



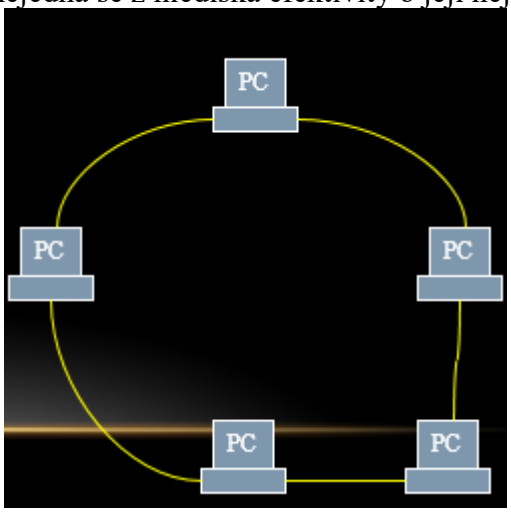
Na obou koncích je hlavní propojovací vedení přes velkou impedanci uzemněno, aby nedocházelo k rušení, nebo je také možné, že jeden konce tvoří síťové zařízení.

Výhodami využití tohoto uspořádání je nízká cena na pořízení a také velmi jednoduché přidání nového zařízení do sítě.

Nevýhodou je velká náchylnost k poruchám, protože banální závada může vyřadit celou síť z provozu nebo způsobit jiné velké komplikace. Tento typ sítí se i špatně administruje, admin musí být šikovný, a i tak na řešení jednoduchých úkonu stráví velké množství času. Další nepříjemností, která je s touto topologií spojena je kolize dat při průchodu sítí.

7.2.5.2. Kruhová

Jako fyzická topologie se používá zřídka, ale jako logická je již četnější. Její fyzické zapojení je celkem problémové, co se týká realizace a opět je omezeno na metalické nosiče. Je sice možné využít optiku, ale nejedná se z hlediska efektivity o její nejlepší využití.



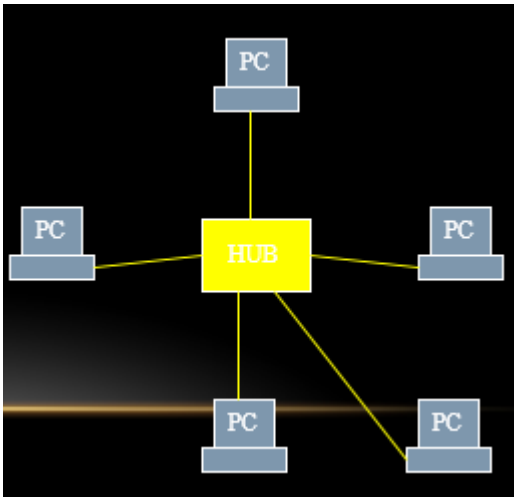
Výhodami je, že nedochází ke kolizím při přenosu dat a je méně náchylná na poruchy. S ohledem na zapojení je velmi spolehlivá, co se týká doručení dat. Tato topologie má velký význam v logické podobě.

Nevýhodou je vyšší pořizovací cena a dražší síťové prvky.

Jako příklad můžeme uvést FDDI a Token ring

7.2.5.3. Hvězdicová

Dnes je hvězdicová topologie nejpoužívanější topologií. V jejím středu je umístěn server, či router a na jednotlivých paprscích jsou jednotlivá zařízení. Pomocí paprsku je také možné spojit více těchto topologií do jedné sítě. Realizovat lze tyto sítě jak bezdrátovými tak metalickými nosiči.



Výhodou je snadné připojení nového zařízení, malá náchylnost k poruchám (porucha ovlivní pouze svůj paprsek, zbytek sítě zůstane nedotčený (s výjimkou serveru či routeru)). Další výhodou je příznivá cena síťových prvků díky jejich masivní výrobě.

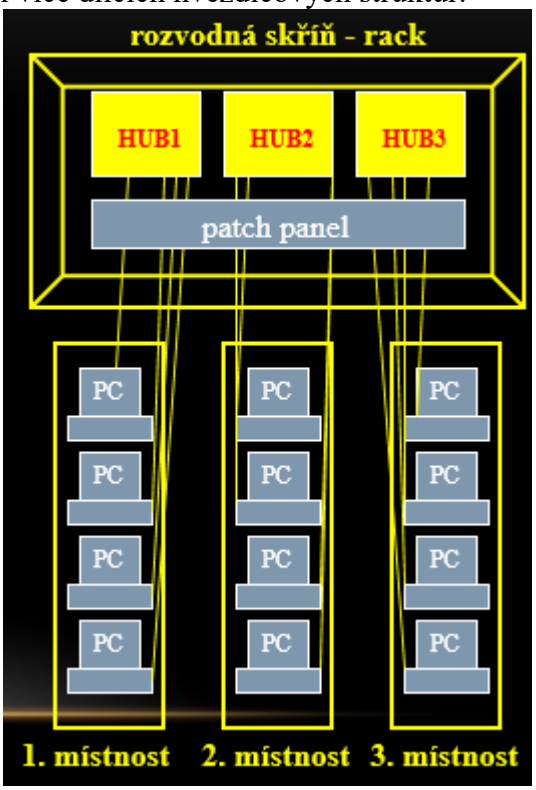
Nevýhodou jsou vyšší náklady na vytvoření sítě a také vznik kolizí při přenosu dat.

7.2.5.4. Hvězdicová kruhová topologie

Jde o příklad, který jsem nastínil. Fyzicky jsou zařízení zapojená do hvězdy, nicméně jsou řízena stejně, jako by byly zapojeny do kruhu. Tato kombinace umožňuje vysokou spolehlivost, nenáchylnost k poruchám a odstranění kolizí. Vzhledem k nákladům na vytvoření a servis tento typ sítě není příliš rozšířen.

7.2.5.5. Strukturovaná kabeláž

Tento typ sítě se užívá kromě domácností naprosto všude. V reálu se dá na tuto strukturu koukat jako na kombinaci více dílčích hvězdicových struktur.



Výhody asi většině z nás vyšli již, jak jsem řekl, že se jedná vlastně o kombinaci hvězd. Je velmi lehké zařízení přemístit, či přidat nové. Máme tu však jednu novou výhodu, snadnou centralizovanou zprávu sítě.

7.2.6. Architektura

Architektura sítě je fyzické uspořádání sítě (topologie) a/nebo protokoly (pravidla pro komunikaci a datové prvky), které jsou použity pro komunikaci. Každá architektura v sobě zahrnuje implicitní topologii (Ethernet - hvězdicová, FDDI, Token Ring - kruhová,…). Architektury se od sebe mohou lišit přenosovým médiem (FDDI - optika, ARCnet - metal. kabel,...), přičemž komunikaci různých architektur zajišťují HUBy, switche, routery, transceivery.

7.2.7. Možnosti přístupu

Sdílené médium (Ethernet, Token Ring,...) - může vysílat současně pouze jeden uzel:

s náhodným přístupem (Ethernet,…) - dochází ke kolizím, s deterministickým přístupen (Token Ring,…) - nedochází ke kolizím což vede k vyšší propustnosti sítě. Přepínané sítě

(ATM) - podle potřeb se vytváří dočasná spojení (i vícenásobná - multiplexing), což umožňuje více uzlům vysílat současně - podstatné zvýšení propustnosti sítě.

7.2.8. Pasivní prvky sítě

Za pasivní považueme fyzické propojení počítačů (kabeláž - metalická, optická, bezdrátové - infračervené, mikrovlnné, GSM, laser), počítačové zásuvky, patch panely (propojovací panely, ve kterých končí přípojky z počítačových zásuvek), racky (rozvodné skříně, v nichž jsou umístěny patch panely a některé aktivní prvky sítě). Tyto prvky sami nikterak aktivním způsobem neovlivňují práci sítě, nelze je nijak nastavovat a aktivně zpravovat.

7.2.9. Aktivní prvky sítě

Zde si uvedeme bodový seznam základních prvků, a jejich velmi stručný popis. Tyto prvky zajišťují správný chod sítě, ale není na naší úrovni vzdělání v informačních technologiích nutné, abychom je podrobně znaly, nám postačí vědět, jak se jmenují a k čemu slouží. Obrázky zde uvádět nebudeme, protože ne vždy poznáme na první pohled rozdíl mezi některými z nich a je jich vyráběno takové množství, že není reálné je poznávat na rozdíl od zbývajícího HW.

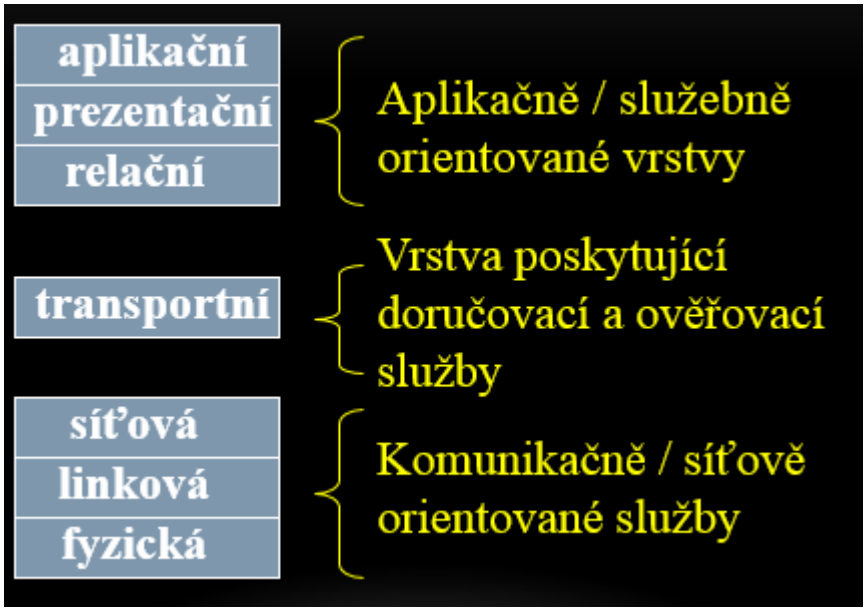
- **síťový adaptér** (NIC - Network Interface Card) - slouží k připojení zařízení do sítě - často integrováno na základní desce počítače nebo se připojuje pře standardní sloty (PCI, ISA, PC card,…),
- **HUB** (čti „hab“ - rozbočovač) - prosté propojení zařízení,
- **bridge** (čti „bridž“ - most) - odděluje provoz v lokálních sítích,
- **switch** (čti „svič“ - přepínač) - umožňuje rozdělení LAN do podsítí, vykonává také funkci bridge,
- **router** (čti „routr“ - směrovač) - umožňuje směrovýní datagramů v rozlehlých sítích,
- **repeater** (čti „ripítr“ - opakovač) - zesilovač signálu,
- **transceiver** (čti „transívr“ - převodník) - převádí signál z jednoho druhu média na jiný (FO/TP, AUI/FO,…),
- **gateway** (čti „getvej“ - brána) - propojuje sítě.

7.3. Referenční model OSI

OSI (Open Systems Interconnection) je sedmivrstvý model navržený Mezinárodní organizací pro standardizaci (International Standardization Organization – ISO). Komunikaci v rámci modelu vidíme na následujícím obrázku.



Na dalším obrázku si malinko roztřídíme vrstvy do skupin podle toho, jaké je jejich poslání.



7.3.1. Fyzická vrstva

Zabývá se fyzikálními a přenosovými charakteristikami jednotlivých přenosových médií, po nichž může probíhat komunikace. Předává datové pakety („balíčky dat“) z linkové vrstvy, převádí je na přenosové médium, po kterém jsou pakety zasílány k fyzické vrstvě příjemce, kde jsou převáděny zpět do datových paketů a předávány linkové vrstvě. Definované (fyzikální) vlastnosti přenosu: rychlost, kódování signálu… Máme i specifikace fyzické vrstvy: RS-232, IEEE 802.3.

7.3.2. Linková vrstva

Určuje způsob výměny datových paketů mezi stanicemi spojenými přímo přenosovým médiem a případně adresaci těchto stanic. Žádosti a data ze síťové vrstvy jsou „zabaleny“ do datových paketů, jež linková vrstva předává fyzické vrstvě k odeslání. Linková vrstva příjemce „vybaluje“ žádosti a data z paketů přijatých od fyzické vrstvy a předává je síťové vrstvě k dalšímu zpracování. Nejpoužívanější síťové architektury a protokoly: Ethernet, Token Ring, FDDI... Funkce fyzické a linkové vrstvy často „zadrátovány“ napevno v hardwaru síťového adaptéru.

7.3.3. Síťová vrstva

Má za úkol vytvořit podmínky pro komunikaci mezi stanicemi, které spolu nejsou přímo propojeny. Slouží k určování síťových adres nebo k převodu hardwarových adres (na linkové vrstvě) na síťové a umožňuje nalezení cesty mezi zdrojem a cílovým uzlem v síti nebo mezi dvěma mezilehlými zařízeními. Vytváří a udržuje logické spojení mezi zdrojovým a cílovým uzlem. Nejpoužívanější síťové protokoly jsou IP (Internet Protocol), ARP (Address Resolution Protocol), ICMP...

7.3.4. Transportní vrstva

Tato vrstva síťového modelu odpovídá za přenos zpráv mezi účastníky komunikace (řízení toku dat a jejich integrita). Je možné předem specifikovat rychlost přenosu a míru chybovosti (velmi spolehlivá, nespolehlivá s navázáním spojení, nespolehlivá – bez navázání spojení). Zde jsou pakety označeny pořadovými čísly a na úrovni relační vrstvy příjemce poskládány opět ve správném pořadí. Transportní vrstva je nepostradatelná!, bez ní není realizovatelný provoz sítě, když by chyběla, tak je to jako když by cikáni ukradli přenosové médium.

Nejpoužívanější transportní protokoly: TCP (s navázáním spojení), UDP (bez navázání spojení) a SPX.

7.3.4. Relační vrstva

Stará se o výměnu řídících informací mezi komunikujícími procesy (žádosti o služby, změna směru přenosu…) a odpovídá za synchronizaci, řazení paketů v síťovém spojení a za udržování spojení do doby, než je přenos ukončen. Často bývá součástí jiných konfigurací (např. na úrovni prezentační vrstvy). Nejpoužívanější relační protokoly: NetBEUI (Microsoft) a NetBIOS.

7.3.5. Prezentační vrstva

Řeší některé „kosmetické“ problémy týkající se komunikace a odpovídá za prezentování informací způsobem, který vyhovuje aplikacím nebo uživatelům (konverze dat zakódovaných v různých znakových sadách, datová komprese a dekomprese, zakončení řádků v textových souborech…)

7.3.6. Aplikační vrstva

Je součástí konkrétního programu, který realizuje předepsané úkony (přenos souborů, elektronická pošta, správa sítě...). Předává žádosti programu a data prezentační vrstvě, která je odpovědná za kódování dat do vhodné formy. Často používané aplikační protokoly: FTP (File transfer Protocol), SNMP (Simple Network Management Protocol), Telnet (terminálová emulace), X.400 (specifikuje funkce a protokoly pro předávání zpráv a elektronickou poštu), HTTP (HyperText Transfer Protocol).

7.4. Bezpečnost počítačové sítě

7.4.1. Uživatelská práva

Přístup ke sdíleným zdrojům musí být omezen (jinak hrozí nefunkčnost systému - viry, hackeři...). Uživatelské jméno a heslo zajišťuje identifikaci uživatele a přidělení odpovídajících uživatelských práv, které specifikují rozsah a úroveň přístupu k síťovým zdrojům. Rozeznáváme několik úrovní:

- host (většinou jen pro čtení veřejně dostupných dat),
- uživatel (může spouštět aplikace, má přístup k vybraným datům, aplikacím a službám),
- administrátor (neomezený přístup k systému, datům a službám, může definovat uživatelská práva a nové uživatele, může instalovat aplikace a konfigurovat systém).

7.4.2. Síťová a aplikační vrstva

I na úrovni těchto vrstev modelu ISO OSI se realizuje zabezpečení. Realizuje se nastavením aktivních prvků sítě (router) a hlavně pomocí firewall, což je specifické zařízení (počítač se speciálním softwarem), které odděluje intranet (privátní síť) od extranetu (veřejná síť - často Internet) a propouští pouze administrátorem povolené požadavky (povolené protokoly, povolené adresy);

TCP/IP filter - může být i součástí OS (Linux, Windows NT, 2000,...) nebo konkrétní aplikace (Oracle,...).

7.4.3. Zabezpečení užtelských dat

Uživatelská data se zabezpečují pomocí šifrování. Dnes využíváme 3 základná druhy:

- **tradiční šifrování** – používá tajný převodní algoritmus – nespolehlivé,
- **šifrování tajným klíčem** – používá jediný klíč známý pouze odesilateli a příjemci (např. DES algoritmus) - není možné chránit zprávu proti podvodu,

- **šifrování s veřejným klíčem** – jeden klíč – veřejný – je ve veřejně přístupné knihovně veřejných klíčů, druhý klíč – soukromý – je znám pouze jedinému účastníkovi; k zašifrování může být použit některý z těchto klíčů, k rozšifrování pak druhý klíč (např. RSA);
- **ukládání a přenos zašifrovaných dat** – podporováno některými OS nebo aplikacemi (PGP – Pretty Good Privacy).

Nejlepší je šifrování pomocí RSA šifry, tedy pomocí veřejného klíče. Nicméně ani tento způsob šifrování (ani žádné bitlockery) není odolní proti kvantovému dešifrování. Nejbezpečnější je šifrovat informace uložené v kvantovém počítači, protože pokus o prolomení data zničí, všechna data jsou čitelná pouze jednou, čehož se dá využít pro kvantové šifrování, které je zatím jen v některých fyzikálních laboratořích, nicméně se předpokládá v budoucnu jeho implementace do výpočetní techniky. Kromě šifrování jsou uživatelská data, která jsou citlivá chráněná zabezpečenými protokoly jako je např. https či ftps.

7.4.4. Elektronický podpis

Podepsat dokument není těžké, ale nastává problém: jak ověřit pravost elektronického podpisu?

Využívá se šifrování s veřejným klíčem: z digitální zprávy se vypočte jednoznačný kód – kontrolní součet, který je zašifrován soukromým klíčem autora zprávy a přiložen ke zprávě jako její elektronický podpis. Příjemce zprávy získá od důvěryhodné certifikační autority veřejný klíč autora zprávy, kdy tento klíč použije příjemce k rozšifrování přiloženého kontrolního součtu zprávy. Rozšifrovaný kontrolní součet porovná s kontrolním součtem zprávy, která mu byla doručena. Pokud výsledky souhlasí, tak je zpráva originálem, pokud se někdo pokusí zprávu narušit, tak výsledky se liší a zpráva je označena jako nedůvěryhodná.

7.4.5. Bezpečnostní politika

Problém: samotná technika nestačí – nejslabším prvkem je lidský faktor. Omezení vlivu lidského faktoru má na starosti bezpečnostní politika firmy:

- odhaluje slabá a kritická místa
- definuje postupy jak je zabezpečit;
- zahrnuje technologická, ale také organizační opatření (jež je složitější dodržet)
- týká se všech pracovišť, zaměstnanců i pracovních postupů
- podléhá pravidelnému auditu.

7.4.6. Další vývoj

Další vývoj přináší nové technologie, nové služby (rychleji, mobilněji, …). Další rozvoj kvantových počítačů, optických médií a kvantové kryptografie. Nové technologie přinášejí nové problémy (bezpečnost dat klesá se vzrůstajícím výkonem, kvantové počítače (snadné dešifrování dat chráněnými současnými algoritmy...). Pokračuje centralizace služeb (tenký klient, outsourcing…) s čímž roste potřeba vysoce kvalifikovaných lidí. Faktická moc se dostává do rukou informatiků (etický kodex informatika 21. století?), ale těžko říci, co bude za 5 let...

8. Modulace

Tato kapitola bude věnována takovému trošku fyzikálně technickému základu pro následující kapitolu o mobilních sítích.

Modulace je úprava informace - modulačního signálu - pro přenos.

Analogové modulace **– modulačním signálem** nižší frekvence se ovlivňuje **nosná vlna** vyšší frekvence – signál se „nasadí" na nosnou. Tím vznikne **modulovaná nosná vlna**.

Diskrétní modulace v základním pásmu – *baseband transmission* - analogový modulační signál se převádí na diskrétní (impulsový) signál rovněž v základním pásmu. Je možná jen na relativně krátké vzdálenosti.

Diskrétní (digitální) modulace s nosnými vlnami (též v přeloženém pásmu) - *broadband transmission* – **digitálním modulačním signálem** nižší frekvence se ovlivňuje vysokofrekvenční **analogová nosná vlna**, tedy signál se „nabalí" na nosnou jako u analogových modulací.

Důvody pro modulaci signálu:

- úprava signálu pro výhodnější přenos po dané cestě – lepší šíření vyšších frekvencí;
- přenos více signálů po jedné cestě;
- přenos digitálního signálu analogovým způsobem.

Modulační a přenosová rychlost:

Modulační rychlost - *modulation speed* - vyjadřuje počet změn nosného signálu za sekundu; jednotkou je Baud (Bd); neříká nic o tom, jaké množství informace nosný signál přenáší.

Přenosová rychlost - *transmission speed* – vyjadřuje množství informace, přenesené za sekundu; jednotkou je bit za sekundu (bit/s nebo bits per second, bps); neříká naopak nic o tom, jak rychle se mění nosný signál.

8.1. Amplitudová modulace

Modulační napětí – analogové napětí nízkofrekvenčního signálu – má rovnici $u_\omega(t) = U_\omega \cos \omega t$ (fázi položíme rovnou nule)

Amplituda modulované nosné se mění v čase podle okamžité hodnoty signálu: $u_\Omega(t) = U_\Omega + U_\omega \cos \omega t$ (též nulová fáze)

Modulovaná nosná má tedy rovnici $u(t) = (U_\Omega + U_\omega \cos \omega t) \cos \Omega t$ (i fázi nosné uvažujeme nulovou).

Výraz upravíme podle součtových vzorců

$\cos(\alpha + \beta) + \cos(\alpha - \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta + \cos\alpha\cos\beta + \sin\alpha\sin\beta = 2\cos\alpha\cos\beta$.

$u = U_\Omega \cos\Omega t + \frac{1}{2}U_\omega \cos[(\Omega+\omega)t] + \frac{1}{2}U_\omega \cos[(\Omega-\omega)t]$. Zavedeme tzv. ***hloubku modulace*** $m = \frac{U_\omega}{U_\Omega}$ (může být i v %)

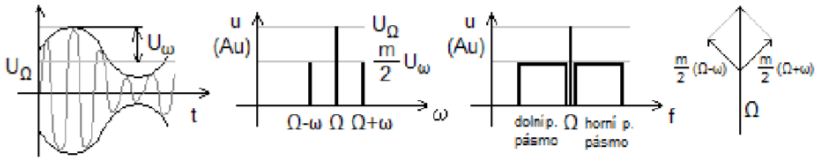
Rovnice modulované nosné je $u = U_\Omega\left\{\cos\Omega t + \frac{1}{2}m\cos[(\Omega+\omega)t] + \frac{1}{2}m\cos[(\Omega-\omega)t]\right\}$.

Frekvenční spektrum AM tedy obsahuje nosnou Ω, dolní postranní kmitočet Ω−ω a horní postranní kmitočet Ω+ω. Fáze modulovaného signálu se oproti nosné nemění (výsledný fázor je ve fázi s nosnou). Přenášíme-li místo jednoho kmitočtu ω celé nf pásmo, budou ve frekvenčním spektru postranní pásma.

Postranní kmitočty mají však amplitudu jen polovinu z hloubky modulace vzhledem k nosné, ta navíc nepřenáší žádnou informaci.

Energeticky úspornější varianty AM: přenos s potlačenou či redukovanou nosnou, přenos jen s jedním postranním pásmem.

na obrázku zleva: časový průběh AM; frekvenční spektrum s jedním ω a s nf pásmem; fázorový diagram AM



8.2. Frekvenční modulace

V rovnici $u(t) = U_\Omega \cos(\Omega t + \varphi)$ položíme opět počáteční fázi φ= 0. Potom $u(t) = U_\Omega \cos \Omega t = U_\Omega \cos \varphi(t)$; φ(t) je okamžitá hodnota fáze.

Protože jde o frekvenční modulaci, mění se úhlová frekvence. Její okamžitá hodnota $\Omega(t) = \Omega + \Delta\Omega \cos \omega t$.

Ω je úhlová frekvence nosné, ΔΩ = 2π.ΔF, ΔF je tzv. ***frekvenční zdvih***, frekvence, o kterou se změní frekvence nosné při amplitudě signálu.

Je-li okamžitá hodnota signálu nulová, je frekvence modulované nosné stejná jako nemodulované; při amplitudě signálu je frekvence modulované nosné nejvíce vzdálená (o frekvenční zdvih) od frekvence nosné.

Protože úhlová frekvence je časovou derivací fáze $\Omega = \frac{d\varphi}{dt}$, je fáze integrálem úhlové frekvence

$$\varphi(t) = \int \Omega(t)dt = \int (\Omega + \Delta\Omega \cos \omega t)dt = \Omega t + \frac{\Delta\Omega}{\omega} \sin \omega t$$

Rovnice modulované nosné je $u(t) = U_\Omega \cos(\Omega t + \frac{\Delta\Omega}{\omega} \sin \omega t)$, kde $M = \frac{\Delta\Omega}{\omega} = \frac{\Delta F}{f}$ je tzv. ***index frekvenční modulace*** a je to maximální odchylka fáze nosné vlny.

Jde o rovnici, kde v argumentu goniometrické funkce je další goniometrická funkce. Řešením jsou tzv. Besselovy funkce:
$u(t) = J_0 \cos \Omega t + J_1 \cos(\Omega + \omega)t + J_2 \cos(\Omega + 2\omega)t + ... + J_1 \cos(\Omega - \omega)t + J_2 \cos(\Omega - 2\omega)t + ...$ což je rovnice nekonečné řady.

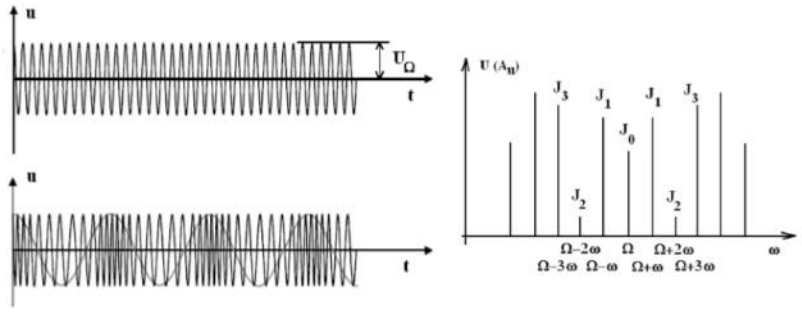
Frekvenční spektrum je proto teoreticky nekonečně široké a obsahuje na obě strany od nosné frekvence rovné celistvým násobkům modulační frekvence.

V praxi lze spektrum omezit přibližně na šířku 2(ΔF + f). Např. u FM monofonního rozhlasu bývá ΔF = 75 kHz a modulační frekvence f = 15 kHz, (index frekvenční modulace M = 5), šířka pásma je 180 kHz. To je nicméně mnohem více než u AM.

FM stereofonní rozhlas: f = 53 kHz, zdvih 75 kHz, index 1,33

Zvuk analogové televize: f = 15 kHz, zdvih 50 kHz, index 3,33.

na obrázku zleva: časový průběh FM (nahoře nosná, dole signál a modulovaná nosná); frekvenční spektrum



Preemfáze a deemfáze frekvenční modulace

U vyšších kmitočtů je modulovaný signál náchylnější k rušení šumem. Proto se ještě přes vysíláním zesílí amplituda modulačního signálu (preemfáze) a v přijímači se stejným způsobem zeslabí (deemfáze), čímž se zeslabí i šum.

Vlastnosti FM – porovnání s AM

(+) menší náchylnost k rušení
(+) energeticky výhodnější
(+) menší vzájemné rušení dvou vysílačů
(+) lepší odstup signálu od šumu
(+) dobrá dynamika (poměr silný/slabý signál)
(-) větší šířka pásma, proto nelze použít u DV a SV, pouze KV a u kratších vln
(-) menší vysílací dosah
(-) složitější demodulátor

8.3. Fázová modulace

lně dosti podobná FM.

jde o fázovou modulaci, mění se v rovnici $u(t) = U_\Omega \cos(\Omega t + \varphi)$ fáze φ. Zavedeme ***fázový zdvih*** Δφ, t

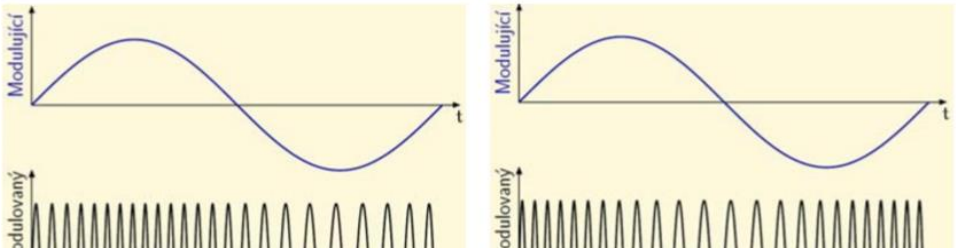
e nejvíce změní fáze nosné. Fáze se mění podle signálu: $\varphi = \Delta\varphi \sin \omega t$.

modulované nosné je $u(t) = U_\Omega \cos(\Omega t + \Delta\varphi \sin \omega t)$. Porovnáme tuto rovnici s rovnicí FM a zjistíme, ž

formálně roven indexu frekvenční modulace: $\Delta\varphi = \frac{\Delta\Omega}{\omega} = M$

úhlová frekvence je derivací fáze, je největší odchylka frekvence modulované nosné od nemodulované při ignálu, tedy při průchodu nulou. Je-li signál v amplitudě, je změna nulová a frekvence nosné je základní. průběh modulované nosné FM a PM je tedy stejný, ale časově posunutý: u PM přebíhá o 90° před FM.

zku vlevo je FM (frekvence nosné v nule signálu, šipka uprostřed), vpravo PM (frekvence nosné v amplitudě rostřed)



Vlastnosti PM v porovnání s FM

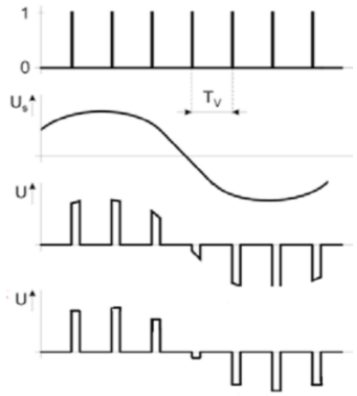
lepší odstup signálu od šumu při vyšších frekvencích
není nutná preemfáze a deemfáze
Přesto se PM pro komerční vysílání nepoužívá, protože by musely být všechny přijímače vybaveny kmitočtovým analyzátorem (PLL nebo DDS) a přijímače by tak byly složitější a dražší. Užívá se v radioreléových a družicových spojích.

8.4. Nekódované diskrétní modulace

Diskrétní (impulzní) modulace jsou založeny na principu odebírání vzorků z původního analogového signálu. Diskrétní impulsový signál se nachází rovněž v základním pásmu. Mohou být buď nekódované nebo kódované (digitální).
Podle Shannon-Kotělnikovova teorému musí být vzorkovací frekvence alespoň dvakrát větší než vzorkovaný (modulační) signál.

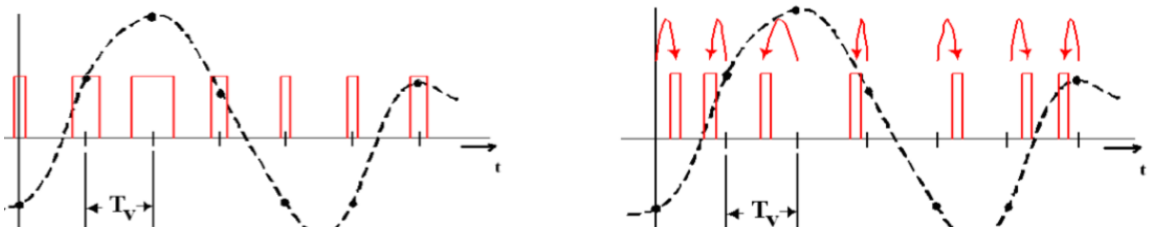
ulsně amplitudová modulace (Pulse Amplitude Modulation)

ní signál se přivádí na vzorkovač, který propouští vždy jen jeho část. Jde o lebírání vzorků z původního signálu s určitou vzorkovací frekvencí. Mění se lsu – amplituda. Nekonečně úzké pulsy (obr. nahoře) nejsou možné. Podle tvaru e rozlišuje věrné a pravoúhlé vzorkování.
zku vpravo je perioda vzorkování T_v, původní analogový signál, věrné a é vzorkování.
ma závisí na šířce pulsu a frekvenci signálu.
aný signál může být velmi ovlivněn rušením podobně jako při AM.
používá také jako první krok při PCM. V tomto případě je kvantovaná (QPAM), orky se kvantují – zaokrouhlují – na nejbližší diskrétní úroveň.



Pulsně šířková modulace (Pulse Width Modulation)

vzorkovací frekvenci (perioda T_v) se vytvářejí pulsy, jejichž šířka je úměrná hodnotě modulačního signálu. Amplituda vzorků je konstantní. Existuje více WM.
využívá i mimo sdělovací techniku, např. v elektrických pohonech.



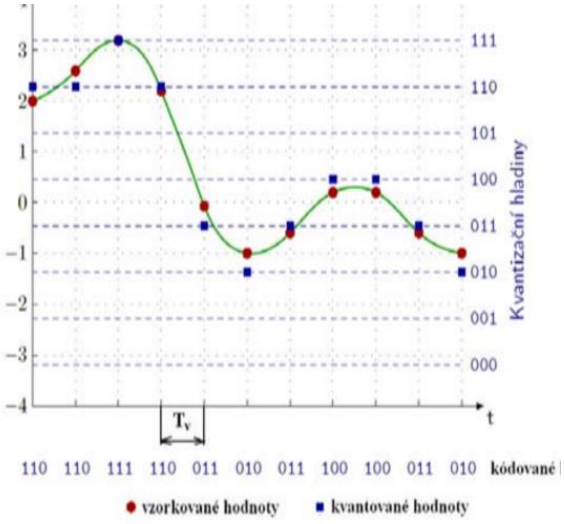
PPM – Pulsně polohová modulace (Pulse Position Modulation)

Pulsy mají vzhledem ke vzorkovací frekvenci s periodou T_v fázový posun, který závisí na okamžité hodnotě modulačního signálu. Šířka i amplituda pulsů je stejná.
Vliv rušení je minimální.
PPM se používá také např. při dálkovém řízení modelů.

8.5. Kódované diskrétní modulace

M – Pulsně kódová modulace (Pulse Code Modulation)

této modulaci se provádějí tři kroky:
rkování - vzorky se odebírají při vzorkovací frekvenci eriodou T_v
ntování – zaokrouhlování na určitý počet hladin; počet din je 2^n, kde n je počet bitů; při kvantování vzniká chyba ačovaná jako kvantizační zkreslení či šum a závisí na počtu din; na obrázku jsou 3 bity – $2^3 = 8$ hladin
lování – převedení na binární kód, tedy jedničky a nuly; že se přidat ještě kontrolní bit.



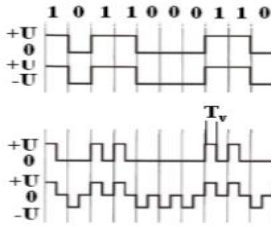
klady:
, telefonní kvalita - 8 bitů, vzorkovací frekvence 8 kHz, ový tok je 64 kbit/s.
dba, CD kvalita - 16 bitů, 44,1 kHz, 2 kanály, tok dat Mbit/s.

stnosti: odolnost proti rušivým vlivům, velký datový tok – často se zmenšuje pomocí různých komprimačních metod.

8.6. Dvojstavové digitální modulace

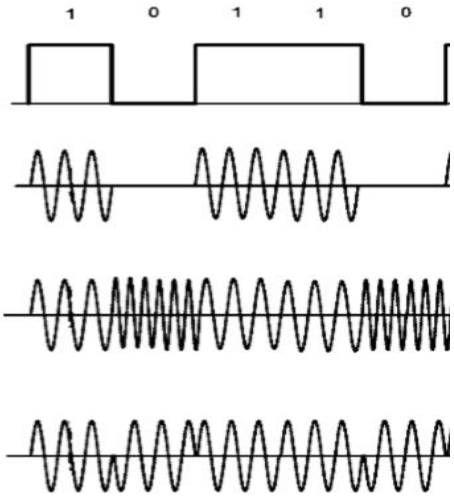
ch signálů

uhů, uvedeme zde:
turn to Zero) – signál se nevrací k nule a je buď unipolární (napětí je ebo bipolární (+U nebo –U);
› Zero) – signál se po době T_v vrací k nule; může být také unipolární či

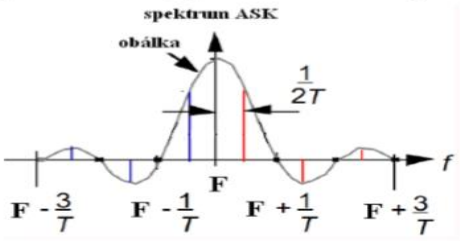


tudové klíčování (Amplitude Shift Keying)

sné se mění podle hodnoty binárního signálu. Např. při e pro binární 0 nulová (jde o prosté vypínání a zapínání N – OFF Keying), při 10% ASK je pro 0 o 10% nižší než



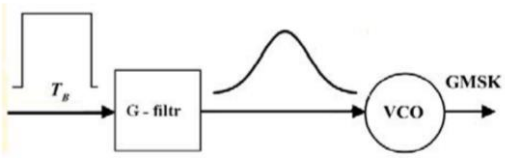
ekvence spektrálních čar s závisejí na šířce impulsu a pásma je teoreticky nekonečná, přenáší se však jen jící většinu energie formy modulace, užují (MSK, GSK). je spektrum signálu ASK (T –



užití: bezdrátové EPROM, IrDA pomocí o záření), čipové karty

FSK - Frekvenční klíčování (Frequency Shift Keying), MSK, GMSK

Podle hodnoty binárního signálu se mění frekvence nosné. Vzhledem k nespojitostem při přechodu frekvencí je spektrum široké. Zajistíme-li spojitý přechod při průchodu nulou (spojitou fází), vznikne **MSK** (Minimum Shift Keying – na obrázku) a spektrum pak obsahuje pouze dvě frekvence souměrné k neexistující nosné. Další zúžení pásma (a tím minimální rušení na jiných frekvencích) vznikne, tvaruje-li se G-filtrem pravoúhlý impuls podle Gaussovy křivky a přechod fáze nosné je pak hladký – jde o **GMSK** (Gaussian MSK).



Příklad použití: mobilní telefony GSM (GMSK)

PSK či BPSK - Fázové klíčování (Binary Position Shift Keying)

Při změně hodnoty binárního signálu se mění fáze nosné o 180° podle obrázku (může být i 90° nebo 270°). Je zajímavé, že spektrum PSK neobsahuje ve spektru nosnou frekvenci. PSK je velmi odolná proti rušení, ale má širší pásmo vhledem k ASK a FSK.

Příklad použití – radioamatérské vysílání na KV; mnohem více se využívá vícestavových PSK (viz dále).

8.7. Více stavové digitální modulace

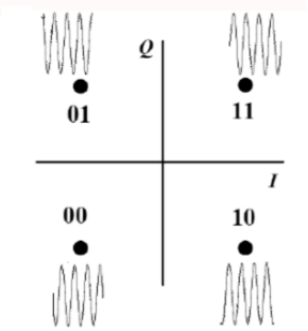
Nutnost zvýšení přenosové rychlosti při přijatelně velké šířce pásma (tedy zvýšení spektrální účinnosti) vedla v praxi k zavádění nových způsobů digitální modulace. Přitom se ovšem může zvýšit chybovost přenosu.

Vícestavové ASK a FSK nejsou pro praxi příliš vhodné. Zato jsou používány vícestavové PSK nebo kombinace ASK a PSK, nazývané obecně QAM nebo APSK.

Fázor nosné nabývá více stavů, které je vhodné znázorňovat v komplexní rovině pomocí tzv. konstelačního diagramu. Místo fázorů pro lepší přehlednost se často kreslí jen jejich koncové body. Reálná osa se často označuje I (In-Phase), imaginární Q (Quadrature Phase). Q je symbolem fázového posunu o 90°.

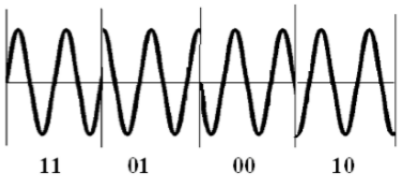
Vícestavové fázové modulace 4-PSK čili QPSK (Quadrature PSK), 8-PSK

Čtyřstavová modulace QPSK (též 4-QAM) přenáší signál o dvou bitech (dibit). Čtyři stavy se liší fázovým posuvem nosné vždy o 90°. Na obrázku dole je vidět, jak se mění fáze nosné při modulaci dibitem. Konstelační diagram obsahuje čtyři body odpovídající polohám fázoru modulované nosné (obr. vpravo).



Podobně lze vytvořit např. 8-PSK pro tři bity (úhel 45°).

Příklady použití QPSK: satelitní vysílání televize DVB-S, modulace subnosných v OFDM



Kvadraturně amplitudové modulace 16-QAM, 64-QAM

Kromě fáze nosné pro jednotlivé stavy digitálního signálu se liší též její amplituda.

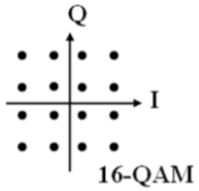
Signál o čtyřech bitech přenáší šestnáctistavová kvadraturně amplitudová modulace. V každém kvadrantu jsou čtyři stavy.

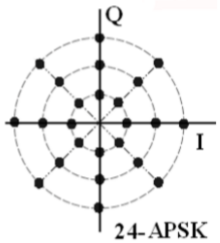
Podobně vznikne čtyřiašedesátistavová QAM pro přenos 6 bitů.

Čím více stavů QAM má, tím menší šířku pásma pro daný datový tok potřebuje, nebo při stejné šířce pásma přenese větší tok dat.

Více stavů však přináší také vyšší chybovost při přenosu – signál je náchylnější k rušivým vlivům. Při volbě druhu modulace je třeba vzít v úvahu přenosové podmínky – odrazy signálu, rušení aj.

Příklad použití: kabelové vysílání televize DVB-C (64-QAM), přenos dat v mobilních sítích.



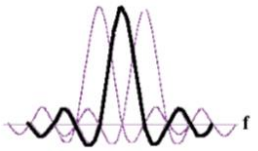


Amplitudově fázové modulace M-APSK

Jsou podobné QAM, ale mají jiný konstelační diagram – kruhově symetrický (obr. vpravo).

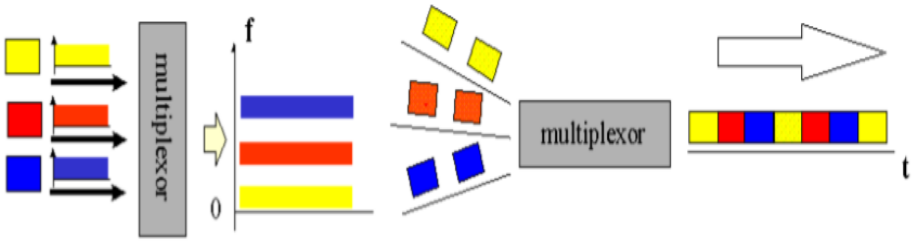
Ortogonální frekvenčně dělený multiplex OFDM (Orthogonal Frequency Division Multiplex)

Tento modulační princip je založen na tom, že přenosový kanál obsahuje velký počet rovnoměrně rozmístěných dílčích subnosných. Dle normy jsou dva módy: 2k (1705 subnosných) a 8k (6817 subnosných). Každá dílčí subnosná vlna je modulována digitální modulací QPSK, 16-QAM nebo 64-QAM. Sériový přenos dat se tak mění na paralelní.
Ortogonální znamená, že ve spektru má každá subnosná maximum tam, kde mají ostatní nulu.
Proto se vzájemně neovlivňují. Na obrázku je část spektra pro pouze tři subnosné.
Výhody: má velmi dobrou spektrální účinnost; je imunní vůči odrazům; lze užít v jednofrekvenčních sítích SFN (více vysílačů na jedné frekvenci, přijímá se jen jeden).
Příklady použití: pozemní vysílání televize DVB-T, digitální rozhlas DRM, DAB, přenos dat v mobilních sítích 4G.



8.9. Multiplex

kvelní multiplex (FDM – Frequency Division Multiplexing nebo FDMA – FD Multiple Access, vlevo) – různé kaná
ášejí na různých frekvencích a v podstatě se neovlivňují. Lze použít i v analogové technice, např. v analogové telefonii.
ový multiplex (TDM - Time Division Multiplexing nebo TDMA, vpravo) – přenos všech kanálů se děje na j
venci, ale v časové posloupnosti, kdy každý kanál má vyhrazený určitý časový interval (timeslot). Je vhodný spíše
tální přenos, např. v mobilní telefonii.



9. Mobilní sítě

9.1. Obecné principy

9.1.1. Metody vícenásobného přístupu

Máme-li pro daný radiokomunikační systém k dispozici určité frekvenční pásmo, které sdílí více uživatelů, můžeme použít jednu ze základních přístupových metod. Metoda tedy určuje, jak přenosové médium rozdělit na jednotlivé kanály, které účastníci dočasně používají ke komunikaci. Základní metody jsou následující:

Vícenásobný přístup s kmitočtovým dělením, FDMA (Frequency Division Multiple Access). Tato metoda spočívá v tom, že dané pásmo rozdělíme na subpásma, kterým jsou dále přiřazeny jednotlivé kanály. Jednotliví účastníci tak komunikují na různých kmitočtech. Tento princip je využíván například u analogových celulárních systémů pro mobilní komunikaci.

Vícenásobný přístup s časovým dělením, TDMA (Time Division Multiple Access). Jedná se o přístupovou metodu použitelnou pouze v digitálních systémech. V daném frekvenčním pásmu jsou vytvořeny časové úseky (intervaly, sloty) a ty jsou poté přidělovány jednotlivým uživatelům. Skupina takovýchto intervalů příslušejících různým účastníkům se nazývá rámec. Nevýhodou této metody je omezený dosah. Ten je způsoben různou dobou šíření rádiového signálu od různě vzdálených uživatelů, a například u systému DECT je tímto problémem omezen dosah na 7,5 km. Omezení tohoto mechanizmu lze řešit měřením a kompenzací doby šíření signálu mezi základnovou stanicí a mobilní stanicí, tzn. vysíláním mobilní stanice v předstihu, aby na přijímací straně nedocházelo k časové kolizi se signály jiných stanic. Tento mechanismus je implementován například v systému GSM, kde je za tímto účelem použit parametr TA (Timing Advance) a dosah u tohoto systému je 35 km.

Vícenásobný princip s kódovým dělením, CDMA (Code division Multiple Access).Jde o novou techniku, která spočívá v tom, že datová posloupnost každého komunikačního kanálu je na vysílací straně podrobena procesu dalšího kódování podle speciálního, pro každého uživatele jedinečného, kódovacího předpisu. Tímto dojde k tzv. rozprostření úzkopásmového signálu do širokého kanálu, společného všem uživatelům. Signály všech účastníků se tedy přenášejí ve stejném frekvenčním pásmu a bez nutnosti časového oddělení. Na přijímací straně probíhá naopak dekódování a jednotlivé kanály jsou odděleny na základě dekódovacího předpisu.

Všechny tyto metody jsou v praktickém nasazení většinou kombinovány, a tak se setkáváme například s kombinací TDMA/FDMA (GSM). Pásmo je tak nejdříve rozděleno metodou FDMA na rádiové kanály, v nichž jsou potom pomocí TDMA vytvořeny kanály uživatelské.

9.1.2. Způsoby přenosu

Pro účastnickou rádiovou komunikaci mezi dvěma body je vždy použita jedna z následujících přenosových metod.

Simplexní přenos – Jedná se o případ, kdy komunikace mezi dvěma body může probíhat pouze jednosměrně a stačí využití jednoho přiděleného kanálu.

Poloduplexní přenos – V tomto případě opět stačí jeden přidělený kanál, ale komunikace může probíhat obousměrně, ne však současně v obou směrech. Jednotlivé směry přenosu je třeba během komunikace přepínat.

Duplexní přenos – Komunikace probíhá současně oběma směry – je tedy nutné použít dva komunikační kanály, pro každý směr jeden.

Pro bezdrátovou účastnickou komunikaci se ve většině případů vyžaduje duplexní přenos. Nejčastěji se používá pro každý směr přenosu rozdílného frekvenčního pásma a příslušná

metoda je označována FDD (Frequency Division Duplex). Tato technika je používána pro analogové systémy nebo digitální systémy s větším dosahem (NMT, GSM).

V některých případech se oba kanály příslušející jednomu okruhu rozlišují na základě časového duplexu, to znamená, že komunikace probíhá na jednom kmitočtu a dopřednému kanálu je přiřazen jeden časový interval, zpětnému pak jiný časový interval. Tento princip je nazýván TDD (Time Division Duplex) a je používán především pro systémy s malým dosahem, například pro bezšňůrové telefony (DECT)

9.1.3. Problémy šíření rádiového signálu

V pozemní mobilní komunikaci se velice často mezi mobilní a základnovou stanicí nacházejí nejrůznější překážky, jako terénní nerovnosti, domy a další různé objekty, takže není možno docílit přímé viditelnosti LOS (Line Of Sight) mezi těmito dvěma stanicemi. Tento případ je velice častý v městských aglomeracích, některé z uvedených překážek se však mohou vyskytovat i ve venkovských oblastech. Nastane-li tedy taková situace, nepřijímá přijímač pouze vlny vysílané základnovou stanicí, ale i vlny, které přicházejí vlivem ohybu, odrazu a rozptylu na zmíněných překážkách. Dochází zde tedy k mnohacestnému šíření rádiových vln (multipath propagation).

Jestliže se mezi základnovou a mobilní stanicí nacházejí relativně velké stínící překážky (vůči vlnové délce λ), závisí intenzita přijímaného signálu na ohybu a na rozptylu rádiových vln způsobených těmito překážkami. Příslušný průběh přijímaného signálu potom vykazuje mírné zvlnění, přijímaný signál je tedy postižen únikem. Tento únik je však nevýrazný, neboť kolísání signálu od střední úrovně zpravidla nepřesáhne 3 až 5 dB. Vzdálenost mezi maximy a minimy v popisované závislosti je většinou několik stovek až tisíc délek vlny λ a proto se tento efekt označuje jako kolísání signálu ve velkém měřítku nebo pomalý únik.

V prostředí s mnohacestným šířením rádiových vln přijímá anténa mobilní stanice mnoho složek vysílaného signálu, vzniklých odrazy od různých okolních objektů, které procházejí různě dlouhými drahami šíření a proto na tuto anténu přicházejí s různým zpožděním; dochází tedy k rozptylu dob zpoždění (delay spread). Všechny složky se v anténě vektorově sčítají, což má v případě pohybujícího se přijímače za následek výrazné kolísání intenzity přijímaného signálu. Takto vzniklý únik signálu bývá velmi hluboký, běžně 20-30 dB. Mezi maximy a minimy je zde vzdálenost odpovídající polovině vlnové délky λ, proto se tento efekt označuje jako kolísání signálu v malém měřítku nebo rychlý únik. Rozložení okamžitých amplitud signálu zde sleduje Rayleighovu statistiku, a proto je tento typ úniku nazýván také jako Rayleighův únik.

Další jev komplikující situaci při rychlém pohybu mobilní stanice je Dopplerův jev (posun). Tento jev způsobuje při vzájemném pohybu vysílače a přijímače kmitočtový posun přijímaného kmitočtu oproti původně vysílanému. Dopplerův jev způsobuje parazitní kmitočtovou modulaci, která způsobuje problémy při demodulaci signálu.

Popisované efekty se objevují nejen při šíření rádiových vln pozemskými rádiovými kanály, ale i v případě komunikace pozemských stanic a stanicích umístěných na kosmických družicích. Ze jsou tyto jevy výrazné především u nestacionárních družic blízko nad obzorem, kdy pozemská stanice přijímá jednak vlnu přímou a jednak vlny odražené od různých překážek.

9.2. Obecný popis bezšňůrových telefonních systémů

Bezšňůrové telefony (**Cordless Telephone**) jsou velice důležitým prostředkem pro osobní radiovou komunikaci. Na rozdíl od běžných celulárních telefonních sítí, které představují kompletní telekomunikační systémy, jsou bezšňůrové telefony specifikovány jako rádiové přístupové sítě (rádiová rozhraní) k externím servisním systémům. Těmi pak jsou fixní i mobilní veřejné telekomunikační sítě, pobočkové ústředny apod.

Analogové bezšňůrové telefony první generace byly určeny hlavně pro privátní účely, kde rádiové spojení plnilo funkci bezdrátového "prodloužení" telefonního vedení s dosahem maximálně několik desítek metrů. S pevnou telefonní sítí bylo možné komunikovat pomocí přenosné mobilní stanice (handy, handset), rádiově spojené se základnovou (pevnou) částí. Později se tyto systémy začaly používat i ve profesionální sféře telekomunikací. Vývoj se ubíral (podobně jako v oblasti mobilních radiotelefonů) od lokálních standardů přes systémy národní až po systémy kontinentální. Během vývoje byl kladen důraz převážně na:

- Zvyšování kvality služeb
- Zlepšení ochrany proti zneužití
- Zvětšování provozní kapacity
- Rozšiřování kompatibility s jinými fixními i mobilními telekomunikačními sítěmi
- Zajištění spolehlivé funkce uvnitř budov, ale také v centrech měst, na předměstích apod.

Tyto nároky je možno splňovat pouze za přispění digitální techniky, a proto se vývoj ubíral jednoznačně směrem k digitálním bezšňůrovým telefonním systémům.

	1. generace	**2. generace**	**3. generace**
Celulární systémy	analogové systémy NMT, C-NET, TACS, ...	digitální systémy GSM, D-AMPS, ...	UMTS
Bezšňůrové systémy	analogové systémy CT0, CT1, ...	digitální systémy CT2, DECT, ...	

Současné digitální bezšňůrové telefony mají v porovnání s mobilními telefony určité přednosti.

- Znatelně lepší kvalita přenosu hovorových signálů, která je plně srovnatelná s kvalitou přenosu v moderních veřejných telefonních sítích. To je způsobeno především přenosovou rychlostí přenášeného signálu, jež je např. u systému DECT (32 kbit/s) znatelně vyšší než u sytému GSM (13 kbit/s).
- Podstatně vyšší provozní kapacita, jíž se dosahuje použitím velmi malých buněk. Typické jsou zde mikrobuňky rozměrů řádově 100 metrů nebo ještě menší pikobuňky.
- Nižší vysílací výkon na obou stranách rádiového spoje, plynoucí z malých rozměrů buněk.

První bezšňůrové telefony byly používány téměř výhradně pro domácí použití, s postupem času se však oblast jejich použití rozrostla o nové konfigurace a způsoby využití. Zde jsou základní možnosti využití současných bezšňůrových telefonů:

- Domácí bezšňůrový telefon
- Pobočková ústředna (PABX)
- Výstavba bezdrátových účastnických smyček **WLL** – Náhrada kabelové účastnické smyčky rádiovým spojem.

Důležitou funkcí většiny bezšňůrových telefonů je tzv. **intercom**, což je možnost komunikace mezi základnovou stanicí a mobilní stanicí. Dnešní moderní systémy poskytují mnoho služeb jako klasické telefony (zkrácená volba, pulsní a tónová volba, podpora služeb ISDN) a někdy nabízejí i možnost pagingu.

9.2.1. Analogové bezšňůrové telefonní přístroje

	1. generace	2. generace	3. generace
Celulární systémy	analogové systémy NMT, C-NET, TACS, ...	digitální systémy GSM, D-AMPS, ...	UMTS
Bezšňůrové systémy	analogové systémy CT0, CT1, ...	digitální systémy CT2, DECT, ...	

Analogové systémy **CT** (Cordless Telephone) se vyznačují podobnými charakteristickými vlastnostmi jako analogové celulární systémy – použita je FM modulace, kmitočtový duplex apod. První bezšňůrové telefonní přístroje se objevily v Evropě kolem roku 1980. Šlo o systémy, které používaly pásmo do 100 MHz, **FM** modulaci, mnohonásobný přístup **FDMA** a kmitočtový duplex **FDD**. Počet kanálů byl různý (8,12,15, ...), dosah těchto zařízení byl 50-200 m a přístroje byly velmi málo odolné proti rušení. Přístroje neměly zabezpečení proti napojení cizích účastníků a ve většině případů se jednalo o ilegálně používaná a neschválená zařízení, sloužící pro domácí použití. Tyto první systémy bezšňůrové telefonie jsou dnes označovány jako systémy **CT-0**.

Evropské státy reagovaly a pod patronací **ETSI** (European Telecommunications Standards Institute) v roce 1983 vytvořily pokročilejší standard **CT-1**. Tento analogový standard pracuje v pásmech 914-915 MHz (směr mobilní stanice-základová stanice) a 959-960 MHz, která jsou rozdělena na 40 rádiových kanálů o šířce pásma 25 kHz. Opět je tedy použito principu **FDMA** a kmitočtového duplexu. **FM** modulace používá zdvih 5 kHz. Přístroje obsahují zabezpečení proti napojení cizích účastníků a kodér, který zamezuje odposlechu. Možná je zde také komunikace několika mobilních stanic s jednou základnovou stanicí.

Standard **CT-1+** vznikl z dvou hlavních příčin. Jednak je komunikace přeložena do pásma 885-887 MHz / 930-932 MHz, což částečně řeší problém kmitočtové kolize se systémem GSM (i zde však nastává kolize s rozšířeným kmitočtovým pásmem E-GSM). Další zlepšení tento systém přináší v rozšíření dostupných rádiových kanálů z 40 na 80, čímž dochází k výraznému zvýšení kapacity systému. Ve verzi CT1+ byl již také aplikován princip dynamického přidělování kanálů **DCA** (Dynamic Channel Allocation) přidělující aktivnímu účastníkovi právě volný duplexní pár kanálů. Systémy typu CT1 a CT1+ byly však poměrně drahou záležitostí, a proto byly ve Velké Británii a Francii vyvinuty vlastní a levnější varianty těchto systémů označené UK CT1 (také MPT-1322), resp. F CT1 (NF C 98-220), které snižují počet kanálů a používají fixní přidělování kanálů **FCA** (Fixed Channel Allocation).

V Japonsku je pro analogový systém **JCT** (Japanese Cordless Telephone) vymezeno v pásmu 254/380 MHz 89 duplexních kanálů o odstupu 12,5 kHz. Opět je použita **FM** modulace, přístupová metoda **FDMA** a duplex **FDD**.

Především z finančních důvodů je stále možno se s těmito analogovými zařízeními setkat a nedochází k jejich úplnému vymizení, které však v nejbližších letech lze očekávat.

9.2.2. Digitální bezšňůrové telefonní přístroje standardu CT

	1. generace	2. generace	3. generace
Celulární systémy	analogové systémy NMT, C-NET, TACS, ...	digitální systémy GSM, D-AMPS, ...	UMTS
Bezšňůrové systémy	analogové systémy CT0, CT1, ...	digitální systémy CT2, DECT, ...	

Druhá generace bezšňůrových telefonních přístrojů je charakterizována digitálním standardem **CT-2** (Cordless Telephone 2) zaváděným ve Velké Británii počátkem roku 1989 a adaptovaným ETSI v roce 1992 jako dočasný evropský standard pro bezšňůrovou telekomunikaci. Tento standard vznikl jako reakce na obchodní neúspěch systémů CT-1 a měl za úkol vyřešit použití bezšňůrových telefonů ve funkci pobočkové ústředny PABX.

V doporučení tohoto standardu je definováno společné rádiové rozhraní **CAI** (Common Air Interface), které vymezuje rozhraní na rádiové cestě a zaručuje kompatibilitu zařízení od různých výrobců. V případě tohoto systému je pak použito konkrétní označení **CT-2/CAI**. Je tak umožněna komunikace se základnovými stanicemi různých operátorů, tady mezisystémový roaming **ISR** (Inter System Roaming). Systém využívá pásmo 864-868 MHz, které je dále rozděleno na 40 duplexních kanálů o šířce pásma 100 kHz. Systém tedy využívá vícenásobný přístup s kmitočtovým dělením **FDMA** a duplex **TDD**. Existují však také základnové stanice pracující v daném okamžiku pouze na jednom kanálu. Tyto stanice jsou použity převážně v domácnostech, kde jsou vícekanálové základnové stanice poměrně nákladnou záležitostí. Zdrojové kódování je zvoleno **ADPCM** (Adaptive Differential Pulse Code Modulation), po jehož aplikaci má signál bitovou rychlost 32 kbit/s. Výhodou tohoto kódování je snadná konverze na standardní formát PCM 64 kbit/s, která může probíhat v základnové stanici. Pro samotná hovorová data systém nepoužívá žádné kanálové kódování, které je nutno použít u digitálních celulárních systémů. To je dáno především faktem, že se uživatelé tohoto systému nepohybují tak rychle jako je tomu u buňkových systémů a nedochází proto k tolik závažné ztrátě signálu. Místo toho je možno použít diverzitního příjmu, při kterém jsou na základnové stanici umístěny 2 antény. Modulací **GFSK** (Gaussian Prefiltered Fraquency Shift Keying) s filtrací modulačního signálu gaussovskou dolní propustí s relativní šířkou pásma B x T=0,5 je do rádiového kanálu umístěn bitový tok o rychlosti 72 kbit/s. Systém umožňuje také přenos dat – v doporučení se však udává přenosová rychlost pouze 2,4 kbit/s při použití klasického datového měniče (modemu).

Standard CT-2 existuje i v národní variantě **CT-2+,** která je používána v Kanadě. Pět ze 40 rádiových kanálů je zde vyhrazeno pro signalizaci a na každém z těchto kanálů je pomocí metody TDMA vytvořeno 12 signalizačních kanálů (celkem tedy 60), které tomuto systému poskytují další služby, jako např. registrace polohy. Další změnou oproti klasickému standardu CT2 je použité kmitočtové pásmo 944-948 MHz.

Posledním standardem CT je systém **CT-3**. Protože se však evropské země dohodly na použití standardu DECT, který má podobné vlastnosti, nedošlo k jeho výraznému rozšíření a systém byl nasazen pouze ve Švédsku. Jako metoda mnohonásobného přístupu je zde použito kombinace **TDMA/FDMA** v kombinaci s **TDD**. Přidělené frekvenční pásmo je 862-866 MHzje rozděleno na 4 rádiové kanály o šířce 1 MHz. Na každém z nich je pak vytvořeno pomocí TDMA 8 uživatelských kanálů. Použité zdrojové kódováni je **ADPCM**, modulace GFSK.

Existence mnoha standardů bezšňůrové telefonie a mnoha národních variant vedlo Konferenci evropských poštovních a telekomunikačních správ CEPTspolu s Evropským ústavem

pro telekomunikační normy ETSI k rozhodnutí vypracovat jednotný standard pro moderní, celoevropský digitální systém DECT. Ostatní standardy bezšňůrových telefonních přístrojů zůstávají sice v provozu, ale z hlediska budoucnosti je perspektivní pouze standard DECT.

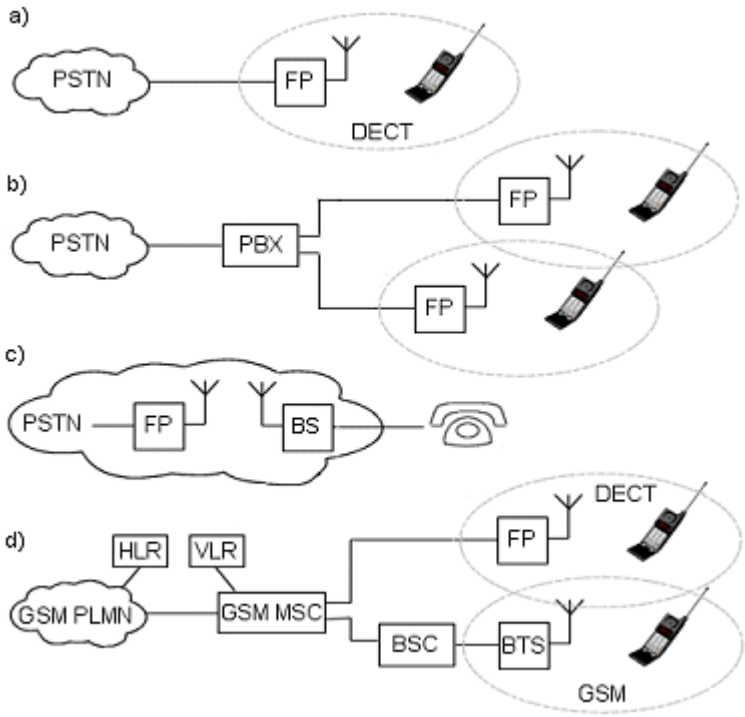
9.2.2.1. Systém DECT

Název **DECT**, vzniklý původně jako zkratka pro Digital European Cordless Telephone (Digitální evropský bezšňůrový telefon), je dnes chápán ve smyslu Digital Enhanced Cordless Telephone (Digitální pokročilý bezšňůrový telefon). Jedná se o evropský standard v rámci ETSI vzniklý v roce 1992, který obecně umožňuje hlasovou i datovou komunikaci a má tyto základní vlastnosti:

- Zaručená kvalita hlasových služeb minimálně na úrovni pevné telefonní sítě
- Vysoká provozní kapacita
- Možnost spolupráce s dalšími druhy sítí, jako je analogová veřejná telefonní síť, ISDN, GSM, přenos dat podle X.25, LAN bezdrátová komunikace aj.
- Transparentnost pro většinu služeb, které jsou užívány v pevných sítích
- Vysoký stupeň zabezpečení proti odposlechu a rušení
- Pružnost při vytváření systémů od malých jednobuňkových až po rozsáhlé mnohabuňkové sítě
- Využitelnost pro přenosy dat rychlostmi do řádu stovek kbit/s
- Dosah v rámci jedné buňky činí desítky metrů v zástavbě a cca 300 m na volném prostranství
- Pružná a efektivní koncepce při využívání kmitočtového pásma
- Možnost aplikací pro řešení účastnických přípojek veřejné sítě umožňující odchozí i příchozí volání
- Možnost vytváření sítí s piko-buňkami a mikro-buňkami

Evropský telekomunikační standardizační institut ETSI koncipoval standard DECT jako digitální bezšňůrový systém s vysokou kvalitou poskytovaných služeb. Systém nabízí možnosti přístupu do řady jiných koexistujících telekomunikačních sítí, přičemž disponuje extrémně vysokou spektrální účinností (provozní kapacitou). Bezšňůrové mobilní stanice MS (Cordless Mobile Station, Phone Handset) komunikují prostřednictvím rádiových kanálů s fixními základnovými stanicemi FP (Fixed Part, Radio Fixed Part). Každá stanice FP obsluhuje jednu buňku mikrocelulárního systému. Pro potlačení vlivu úniku signálu a interferencí mohou tyto stanice využívat prostorovou diverzitu se dvěma anténami, z nichž v činnosti je ta, která poskytuje lepší signál. Mezi stanicemi MS a FP je společné rádiové rozhraní **CAI** (Common Air Interface). Během komunikace může dojít k vnitrobuňkovému handoveru, při kterém se komunikace např. z důvodu rušení přepne na jiný kanál. Handover je řízen přenosnou částí, jde tedy o tzv. **MCHO** (Mobile Controlled HandOver).

Zařízení DECT lze použít v mnoha oblastech a konfiguracích, základní z nich jsou uvedeny na následujícím obrázku.

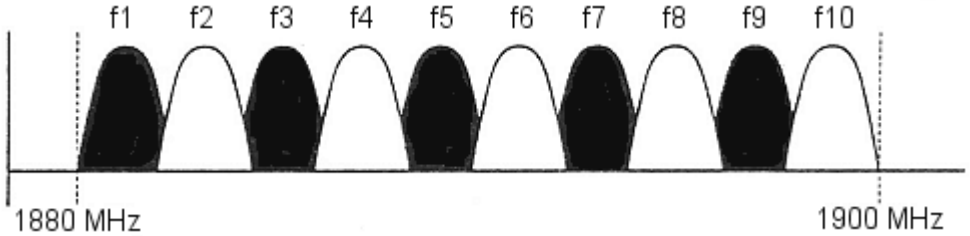


Základní konfigurace systému DECT jsou uvedeny v následujícím výčtu, použití tohoto systému je však velice univerzální a lze jej úspěšně využít v mnoha dalších aplikacích v návaznosti na veřejné i neveřejné sítě.

- Domácí bezšňůrový telefon – na obrázku a)
- Kombinace s pobočkovou ústřednou PBX – na obrázku b)
- Bezdrátová účastnická smyčka WLL (Wireless Local Loop) - na obrázku c)
- Izolovaná veřejná mikrocelulární síť, umožňuje napojení na radiotelefonní ústřednu MSC systému GSM – na obrázku d)
- Použití pro přenos dat (Local Area Network)

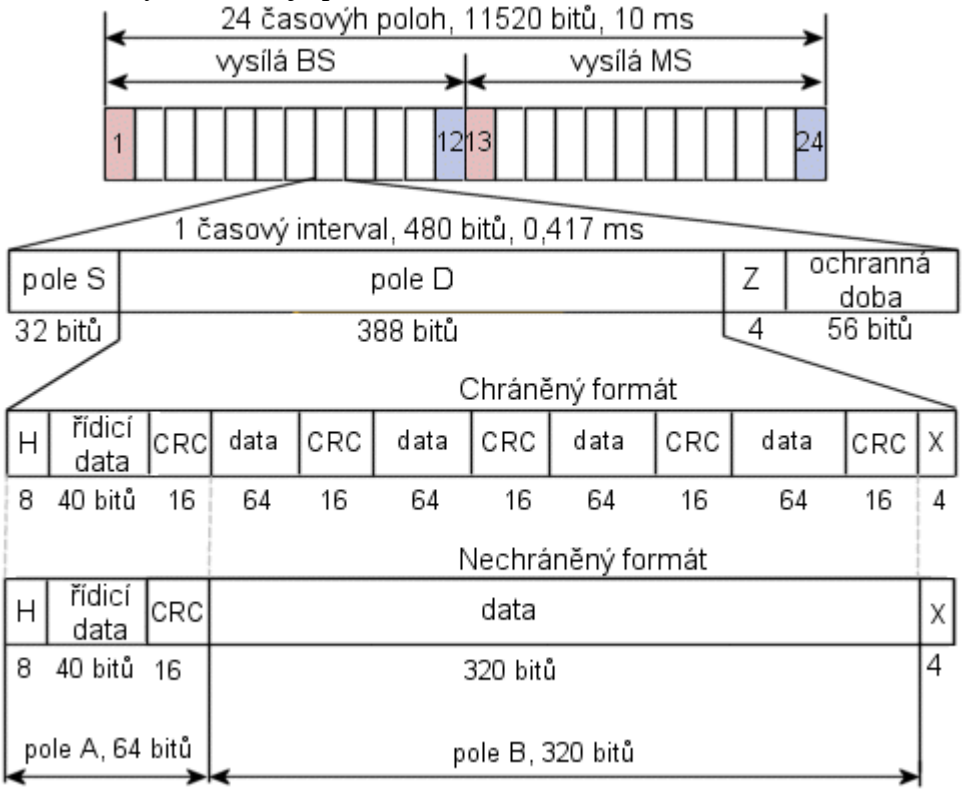
Rádiový sektor

Standard DECT pracuje v rozsahu 1880 až 1900 MHz (v budoucnu 1880 až 1920 MHz). V něm je rozloženo celkem 10 nosných vln, se vzájemným odstupem 1728 kHz.



K oddělení účastnických kanálů se tedy využívají principy multiplexu FDMA, avšak uvedená koncepce se zde označuje jako systém MC (Multi Carrier tj. mnoho nosných vln), neboť účastnické kanály neustále dynamicky mění obsazení jednotlivých nosných podle okamžité situace. Dále zde jsou uplatněny principy přístupu TDMA a to tak, že je v pásmu o šířce necelé 2 MHz přenášen vždy po dobu 10 ms jeden časový rámec TDMA. Ten obsahuje celkem 24 časových intervalů TDMA, jež mají dobu trvání cca 0,417 ms a přenášejí 480 bitů. Rámec tak obsahuje celkem 480 x 24 = 11 520 bitů. Za 1 sekundu se vystřídá na této nosné 100 rámců, tedy celková přenosová rychlost signálu zde je 11 520 bitů x 100 = 1,152 Mbit/s. Původně byla pro tento systém zvolena modulace GFSK, později se však výhodnější ukázalo použít modulaci GMSK (B x T = 0,5), odzkoušenou v systému GSM. Zdrojové kódování je zde užito ADPCM s přenosovou rychlostí výstupního signálu 32 kbit/s.

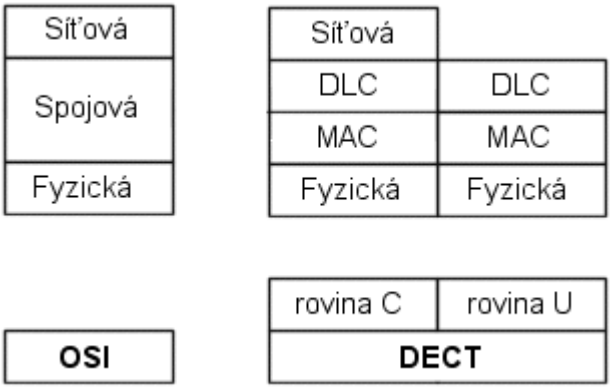
Standard DECT aplikuje časové sdružování duplexních párů, tj. časový duplex TDD. Ve slotech 1 až 12 rámce vysílají základnové stanice a ve slotech 13 až 24 vysílají mobilní stanice. Jedná se tedy o časový duplex s 12 duplexními kanály na jeden rádiový kanál a celkové množství uživatelských kanálů je potom 120.



V jednom časovém intervalu se přenáší celkem 480 bitů. Prvních 32 bitů vytváří synchronizační skupinu (pole S). Dále následuje skupina 48 bitů přenášejících signalizaci (pole H - záhlaví nesoucí 8 bitů pro permanentní řízení a dalších 40 řídicích bitů) a skupina l6 bitů cyklického redundandního kódu CRC. Kód CRC se využívá pro kontrolu chyb v řídicím slově s případným následujícím opakovaným vysíláním řídicí informace. Těchto 48 + 16 = 64 bitů vytváří pole A, jehož přenosová rychlost je 6,4 kbit/s. Za polem A následuje pole B, obsahující 320 bitů určených k přenosu užitečného signálu (digitalizovaného hovoru, nechráněných dat) rychlostí až 32 kbit/s. Jsou definovány dva formáty pole B – chráněný a nechráněný. Chráněný formát se vyznačuje kvalitním zbezpečením, kdy je informační tok rozdělen do skupin po 64 bitech, které jsou pak chráněny stejným CRC kódem jako pole A. V případě nechráněného formátu není toto zabezpečení použito a ochranu tak obstarává zabezpečovací pole X o 4 bitech, které je použito vždy. Pole A+B+X se souborně označují jako datové pole D. Následující pole Z o čtyřech bitech přispívá rovněž ke kontrole chyb. Časový slot zakončuje 56 bitů ochranného intervalu, kompenzujícího chyby v časování apod. Za 1 sekundu se opakuje celkem 100 časových rámců TDMA a tedy i 100 slotů přidělených jednomu ze dvou kanálů určitého duplexního páru. Tomu odpovídá celková přenosová rychlost pro přenos hovoru nebo nechráněných dat 100 x 320 = 32 kbit/s na jeden kanál. Časové intervaly TDMA je možné vhodně kombinovat a tím vytvářet jediný kompozitní kanál s vyšší bitovou rychlostí, jejíž horní hranice je 12 x 32 = 384 kbit/s.

9.2.2.1.1. Vrstvový model DECT

Na následujícím obrázku je znázorněn vztah vrstvového modelu OSI a vrstvového modelu DECT, který je definván pro 3 nejnižší vrstvy.



- **Fyzická vrstva**

Fyzické vrstva dělí rádiové spektrum na jednotlivé uživatelské kanály. Toto dělení probíhá jednak ve frekvenční a jednak v časové oblasti. Toto spektrum je také pomocí DCA (Dynamic Channel Allocation) rozdělováno do jednotlivých buněk, ve kterých může být použito stejných kanálů.

- **Vrstva řízení přístupu k přenosovému prostředku (MAC – Medium Access Control)**

Tato vrstva obstarává dvě hlavní funkce. Vybírá fyzické kanály, sestavuje a ukončuje spojení na těchto kanálech a dále doplňuje informační bity z vyšších vrstev o řídicí a zabezpečovací informace na velikost jednoho časového intervalu.

- **Vrstva řízení datového spoje (DLC – Data Link Control)**

Vrstva DLC poskytuje síťové vrstvě velice spolehlivý datový spoj. Mnoho nedostatků rádiového přenosu je odstraněno již na úrovni MAC. Vrstva DLC s MAC blízce spolupracuje a síťové vrstvě poskytuje vyšší úroveň celistvosti dat, než by mohla poskytnout vrstva MAC sama.

Vrstvový model DECT je dále dělen dle činnosti na vrstvě DLC na uživatelskou a řídicí rovinu (User and Control Plane).

- **Síťová vrstva**

Tato vrstva je nejdůležitější v souvislosti se signalizačními protokoly. Je velice podobná třetí vrstvě v ISDN a nabízí podobné funkce.

9.2.2.2. Ostatní digitální bezšňůrové telefonní systémy

9.2.2.2.1. PHS (Personal Handyphone System)

Jedná se digitální bezšňůrový telefonní systém, dříve také nazývaný PHP (Personal Handy Phone). Vyvinut a používán je v Japonsku od roku 1995. Systém je velice podobný evropskému standardu DECT, využíván je však převážně v konfiguraci bezdrátové účastnické smyčky WLL (Wireless Local Loop), která umožňuje zavedení telefonu i do míst, kde je to klasickým způsobem obtížné nebo z jiného důvodu nevýhodné. Systém v Japonsku dále pracuje jako alternativa k celulárním mobilním sítím a uživatelé mohou mobilní stanici užívat v některých hustě osídlených oblastech, omezena je však rychlost pohybu.

PHS popisuje pouze rádiový sektor a nikoliv kompletní infrastrukturu. Standard je založený na přístupu TDMA/FDMA a duplexu TDD. Systém má k dispozici 40 rádiových kanálů v pásmu 1906,1-1918,1 MHz pro veřejné aplikace a 37 rádiových kanálů v pásmu 1895-1906,1 pro nasazení v domácnostech a kancelářích. Celkem je zde tedy 77 rádiových kanálů. Šířka rádiového kanálu je tedy 300 kHz, každý je dělen na 4 uživatelské kanály a modulace je $\pi/4$

DQPSK. V systému je použito zdrojové kódováni ADPCM (32 kbit/s) a proti chybám je přenos zabezpečen kódem CRC.

Zajímavostí je fakt, že je možná přímá komunikace mezi dvěma mobilními stanicemi bez účasti pevné složky. Handover nastat může, není však bezešvý (jako u systému DECT) a dochází na 2 sekundy k výpadku spojení, což je dáno méně robustní signalizací.

9.2.2.2.2. PACS (Personal Access Communication Services)

Americký standars PACS je kombinací technologií užitých u staršího systému WASC (Wireless Access Communication System) a japonského PHS. V porovnání se systémem PHS, popsaného výše, je odlišných několik technických parametrů. Například je použito kmitočtového duplexu na pásmech 1850-1910 MHz / 1930-1990 MHz. Metodou FDMA je zde vytvořeno 200 rádiových kanálů a na každém pak 8 kanálů uživatelských. Opět je použito zdrojové kódování ADPCM, modulace je π/4 QPSK.

V případě nasazení této technologie jako alternativy k mobilní buňkové síti je až 160 tzv. rádiových portů (RP – Radio Port) připojeno na jednu řídicí jednotku rádiových portů (RPC - Radio Port Controller). Tyto RPC mohou komunikovat s centrální databází a dohledovým centrem. Rádiové porty velmi malých rozměrů jsou umístěny na ulicích a zajišťují pokrytí signálem pro hustě osídlené oblasti (města, nádraží, obchody).

9.2.2.3. Digitální bezšňůrové telefonní přístroje – přehled

	CT 2	DECT	PHS	PACS
Frekvenční pásmo [MHz]	864-868	1880-1900	1896-1918	1850-1910 / 1930-1990
Duplex	TDD	TDD	TDD	FDD
Šířka pásma	100 kHz	1,728 MHz	300 kHz	600 kHz
Metoda přístupu	FDMA 40 kanálů	FDMA 10 kanálů TDMA 12 slotů	FDMA 77 kanálů TDMA 4 sloty	FDMA 200 kanálů TDMA 8 slotů
Zdrojové kódování	ADPCM, 32 kbit/s	ADPCM, 32 kbit/s	ADPCM, 32 kbit/s	ADPCM, 32 kbit/s
Modulace	dvoustavová GMSK	dvoustavová GMSK	π/4 DQPSK	π/4 QPSK
Dosah		100 - 500 m	50 - 150 m	300-500 m

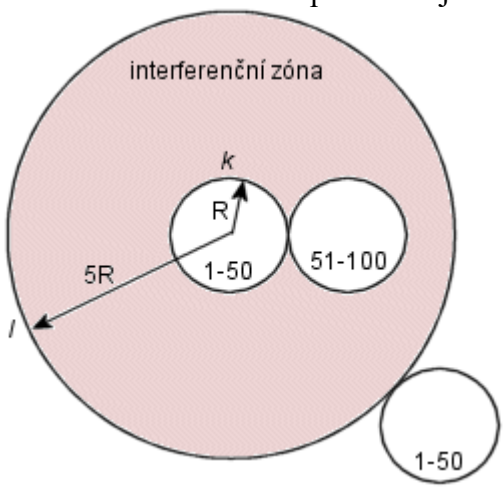
9.3. Buňkové telefonní sítě

9.3.1. Princip buňkového systému

První pozemní komunikační systémy měly takzvanou ostrůvkovitou koncepci. Jediná základnová stanice zajišťovala spojení s mobilními stanicemi v celém, obvykle velice rozlehlém území. Taková koncepce však měla řadu nevýhod:

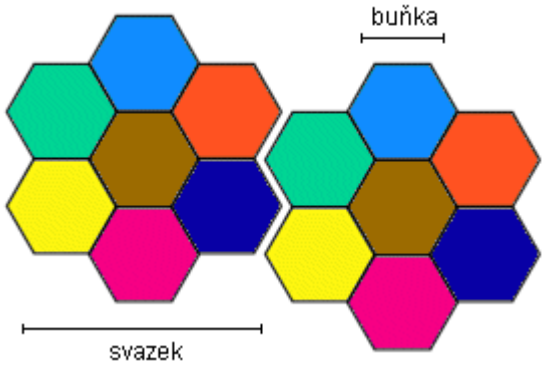
- Mobilní účastník takovéto sítě si při přesunu do oblasti sousední základnové stanice musel sám přeladit mobilní stanici na jiná frekvenční pásma.
- Koncepce vyžadovala velké vysílací výkony u vysílačů základnových stanic i mobilních stanic, které jsou potom rozměrné a energeticky náročné.

- Zásadním nedostatkem je špatné využití přiděleného frekvenčního pásma. Aby systém mohl pracovat bez "hluchých zón", musel by mít k dispozici velmi velký počet přidělených kanálů a tím pádem velmi široká frekvenční pásma, což je však při nedostatku frekvenčního spektra problém. Podstata tohoto problému je znázorněna na obrázku.



V oblasti znázorněné kruhem k o poloměru R je umístěna jediná základnová stanice, která obsluhuje celé toto území a disponuje rádiovými kanály číslo1 až 50. Je-li spolehlivě pokryta oblast kruhu k, signál základnové stanice je jistě poměrně silný i v tzv. interferenční zóně v podobě kruhu l a v celé této oblasti se již kanály 1-50 nesmí vyskytovat a musí být využívány kanály 51-100, 101-150 atd. Celkově tedy bude pro pokrytí oblasti l zapotřebí přibližně 1250 různých kanálů. Tyto kanály se mohou opakovat až vně interferenční zóny, tedy po překročení bezpečnostní vzdálenosti (reuse distance) 5R.

Revoluční zlom ve vývoji přišel v roce 1946, kdy byl v laboratořích firmy Bell formulován princip tzv. celulárních (buňkových) systémů pro mobilní komunikaci. Základním rysem současných mobilních buňkových radiotelefonních systémů je velmi efektivní hospodaření s frekvenčním spektrem, které je výsledkem mnohonásobného použití stejné přidělené frekvence v obsluhované oblasti. Princip buňkové struktury ukazuje obrázek.



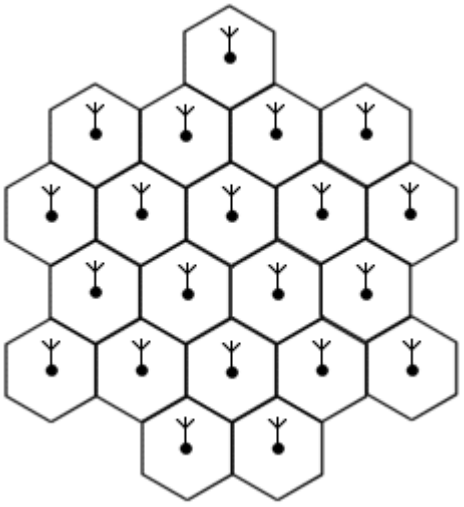
Celá obsluhovaná oblast je zde rozdělena do 14 šestiúhelníkových buněk, které tvoří dva svazky (clusers) po sedmi buňkách. Dalším přidáváním svazků je potom možno pokrýt neomezeně velké území. Uvnitř každé buňky je jedna základnová stanice s určitou přidělenou skupinou kanálů a komunikující s mobilními účastníky, kteří se nacházejí pouze v této buňce. Zbývajících 6 buněk příslušejících jednomu svazku má přiděleny své skupiny kanálů. Za podmínky, že se oblast všech sedmi buněk přibližně rovná interferenční oblasti, je v každém z obou svazků možno použít stejné kanály.

Pro porovnání se starší koncepcí (viz. výše) předpokládejme, že je plocha svazku zhruba stejná s plochou kruhu o poloměru R. V první buňce jsou použity např. kanály 1-7, ve druhé 8-15 a v sedmé 42-49. Svazek potom obsahuje 49 kanálů, což je prakticky stejný počet jako obsahuje kruh k. Důležité je to, že lze tyto kanály opakovat ihned v sousedním svazku. Vzdálenost buněk používajících stejnou frekvenci je totiž rovna přibližně pětinásobku jejich poloměru a je tudíž splněna podmínka 5R. Systém tedy potřebuje pro pokrytí libovolně velkého území 50 kanálů, kdežto starší systém by jich potřeboval asi 1250. Toto je jedna z hlavních předností celulárních systémů, mezi které patří:

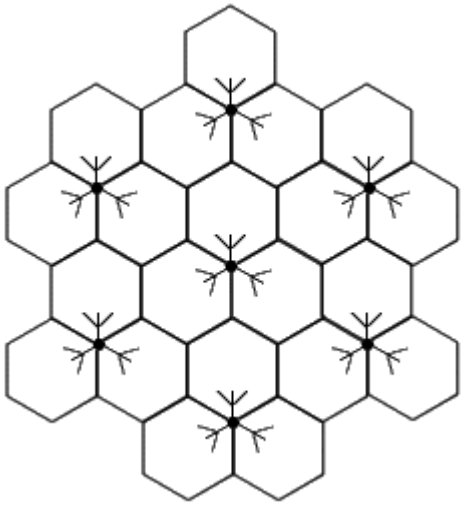
- Již zmíněné efektivní hospodaření s přiděleným rádiovým spektrem.
- Možnost použít handover.
- Podstatně menší vysílací výkon na straně základnové stanice i mobilní stanice.

Tyto výhody však nejsou úplně zdarma. Jestliže ve starším systému obsloužila oblast o poloměru R jediná základnová stanice, je jich v buňkovém systému pro přibližně stejně velký svazek potřeba sedm.

Pro vytvoření sítě s lepšími provozními vlastnostmi (např. nižší vysílací výkony a zvětšení počtu současně obsluhovaných mobilních stanic) je možno použít princip tzv. sektorizace. Jeden svazek nyní rozdělíme na 21 menších buněk.



Nezmění se tím počet potřebných kanálů, ale stoupne počet základnových stanic ze 7 na 21. Jejich počet však lze sektorizací redukovat na sedm umístěním tří samostatných směrových antén do společných bodů tří sousedních buněk.



Velké buňky mají poloměr zpravidla větší než 3 km a antény bývají umístěny nad nejvyšším bodem okolní zástavby. Malé buňky mají poloměr menší než 3 km a antény základnových stanic jsou umístěny pod nejvyšším bodem okolní zástavby, avšak výše než je střední výška zástavby. Mikrobuňky o poloměru menším než 300 m mají antény základnových stanic pod úrovní střech a okolních budov; signál se zde šíří v uličních "kaňonech" díky rozptylu a ohybu kolem těchto budov. Uvnitř budov a v místech s vysokou koncentrací osob mohou být použity pikobuňky, které zpravidla mívají dosah desítek metrů.

9.3.2. Handover

V sousedních buňkách se vždy používají kanály s odlišnými frekvencemi a při přechodu mobilního účastníka přes hranice buněk je tedy nutné přeladit jeho stanici. Tento proces se v moderních systémech uskutečňuje zcela automaticky, bez zásahu obsluhy stanice, a nazývá se handover. Provoz systému vyžaduje, aby byla neustále registrována okamžitá poloho mobilní stanice alespoň na úrovni buněk. Určování polohy stanice může být realizováno například tak, že mobilní stanice neustále udržuje spojení s nejbližšími základnovými stanicemi, které jsou v dosahu. Ty vyhodnocují kvalitu spojení a informují o tom ústřednu. Z těchto informací poté ústředna určí tu základnovou stanici, jejíž spojení s mobilní stanicí je nejkvalitnější a která je tedy pravděpodobně nejblíže. Dojde-li k přemístění mobilní stanice z jedné buňky do druhé v průběhu hovoru, ústředna jí přidělí nové hovorové kanály. Celý tento proces trvá velice krátce a během hovoru je nepostřehnutelný.

Mezibuňkový handover: Automatické přelaďování mobilních stanic při přechodech přes hranice buněk se označuje pojmem inter-cell handover. V momentě 1 mobilní stanice komunikuje se základnovou stanicí A. V okamžiku 2, kdy se stává spojení se stanicí B kvalitnější, dochází k přeladění a komunikace přechází na stanici B a nadále již probíhá komunikace s touto základnovou stanicí.

Vnitrobuňkový handover: V moderních sítích se používá i metoda přelaďování mobilní stanice v průběhu pohybu v rámci jedné buňky, neboť se mohou objevit kanály, zajišťující kvalitnější spojení než kanály původně přidělené. Tento proces se nazývá intra-cell handover.

Celý tento mechanizmus probíhá v jediné buňce a mobilní stanice stále komunikuje s jedinou základnovou stanicí. Do momentu 2 probíhá komunikace na určitých přidělených kanálech a v tomto okamžiku se stanice přeladí na kanály nové, které poté používá. Popisovaný proces může být řízen stacionárními složkami celulárního systému, bez spoluúčasti mobilní stanice. Mobilní stanice se však může řízení také zúčastnit, případně jej na sebe zcela převzít. Rozlišujeme tedy tři základní typy handoveru:

Handover řízený systémem (Network Controlled HandOver - NCHO): Mobilní stanice pouze vysílá kontrolní signál, který je přijímán základnovými stanicemi v rádiovém dosahu. Příslušný systém, umístěný např. v ústředně, poté na základě porovnání kvality signálu rozhodne, který signál je nejlepší a se kterou základnovou stanicí je pro mobilní stanici nejvýhodnější komunikovat. Výhodou této metody je fakt, že mobilní stanice neprovádí žádné měření, jehož výsledky by bylo nutné zpětně přenášet. Na druhou stranu slabinou jsou vysoké nároky na přenosovou kapacitu tras mezi základnovými stanicemi a radiotelefonní ústřednou, zatížení rozhodovacích subsystémů ústředny a nemožnost mobilní stanice aktivně se zúčastnit rozhodovacího procesu. Tento handover používají zejména analogové celulární systémy, například systém NMT.

Handover řízený za spoluúčasti mobilní stanice (Mobile Assisted HandOver - MAHO): Mobilní stanice nepřetržitě měří intenzitu, případně poměr signál/šum signálů okolních základnových stanic a výsledky předává základnové stanici, s níž právě komunikuje. Měření signálu vysílaného mobilní stanicí může provádět i tato, případně okolní základnové stanice. Rozhodnutí je opět provedeno příslušným systémem v radiotelefonní ústředně, ale jako základní rozhodující informace jsou brány údaje přicházející z mobilní stanice. Výhodou této metody je

akceptování lokálních podmínek, v nichž se stanice nachází, dále posuzování kvality sestupné i vzestupné rádiové trasy a menší nárok na přenosovou kapacitu fixních složek sítě. Nevýhodou je pak větší intenzita přenosu rádiovým sektorem systému. Tento typ handoveru používá například síť GSM.

Handover řízený mobilní stanicí (Mobile Controlled HandOver – MCHO): Zde se rozhodovací proces o přechodu na nové realizuje pouze na mobilní stanici. Vychází se jak z údajů měřených přímo mobilní stanicí, tak z měření prováděných základnovými stanicemi. Výsledný požadavek na změnu poté mobilní stanice předává přímo nové základnové stanici, do jejíž buňky právě vstoupila. Tento model mění koncepci mobilních stanic, ale výrazně šetří rádiový sektor a je z uvedených principů nejrychlejší. Tento handover je aplikován například u bezšňůrových telefonů DECT.

Z hlediska přepínacího procesu dále rozlišujeme následující druhy handovery:

Hard handover "Tvrdý": V tomto případě je nejdříve mobilní stanice odpojena od původního kanálu a teprve poté připojena na kanál nový. Čas přepojení je co možná nejmenší, přesto dojde ke krátkému přerušení spoje, které je však v hovoru těžko postřehnutelné. Tento typ handoveru však může činit problémy při přenosu dat. Handover je většinou řízen sítí (NCHO). Výhodný je v tom, že terminál nemusí být schopen pracovat na více kanálech současně a je tedy jednodušší. Tato technologie je používána například v sítích NMT nebo GSM.

Seamless handover "Bezešvý": Jedná se o handover, při němž se nejdříve vytvoří spojení na novém kanálu a teprve poté je zrušeno spojení na kanálu opouštěném. Mobilní stanice tedy musí určitý čas komunikovat současně na dvou kanálech, což vyžaduje synchronizaci. Řízení většinou přebírá mobilní stanice (MCHO) a tento handover používá například systém DECT.

Soft handover "Měkký": Tento způsob handoveru využívá faktu, že je stanice neustále připojena alespoň ke dvěma dostupným základnovým stanicím. Při pohybu terminálu jsou jednotlivá spojení rušena a další nová vznikají. Tento typ způsobuje větší nároky na kapacitu sítě a je používán například v americkém systému IS-95. Uplatnění najde také v systému třetí generace UMTS.

9.3.3. Analogové systémy ve světě

Analogové celulární mobilní radiotelefony obecně spadají pod systémy mobilní komunikace první generace.

	1. generace	2. generace	3. generace
Celulární systémy	analogové systémy NMT, C-NET, TACS, ...	digitální systémy GSM, D-AMPS, ...	UMTS
Bezšňůrové systémy	analogové systémy CT0, CT1, ...	digitální systémy CT2, DECT, ...	

Ze současného pohledu se sice jedná o překonané varianty telefonních systémů. Díky počtu jejich účastníků (řádově desítky miliónů) dosaženému v průběhu jejich rozvoje s nimi bude třeba ještě několik let počítat. Analogové mobilní systémy používají převážně frekvenční modulaci FM, přístupovou metodu FDMA a duplex FDD. Tyto systémy trpí několika hlavními nedostatky:

- Nízká kvalita přenosu
- Malé zabezpečení proti odposlechu a zneužití
- Neefektivní využití přiděleného frekvenčního pásma
- Z důvodu neslučitelnosti systémů není možný mezinárodní roaming (volný mezinárodní pohyb účastníka)

Prvním systémem uvedeným do veřejného provozu byl v roce 1979 americký systém **AMPS** (Advanced Mobile Phone System), následovaný v roce 1981 systémem **NMT** (Nordic Mobile Telephone). Spuštěny byly národní varianty v mnoha zemích a nástup těchto systémů uzavíraly v roce 1985 systémy **TACS** (Total Access Communications System) ve Velké Británii a **C-Net** (Cellular Network) ve Spolkové republice Německo. Systém NMT 450 byl v roce 1991 zaveden i v ČSFR.

Přehled základních analogových systémů mobilní komunikace včetně základních parametrů je uveden v tabulce:

Označení systému	Zahájení provozu	Země	Vysílací pásmo základnové stanice [MHz]	Vysílací pásmo mobilní stanice [MHz]	Šířka kanálu [kHz]	Počet kanálů
NMT 450	1981	Švédsko, Norsko, Dánsko, Finsko, ČR, ...	463-467,5	453-457,5	25	180
AMPS	1979 zkušebně 1983 komerčně	USA, ...	869-894	824-849	30	832
TACS	1985	Velká Británie	935-950	890-905	25	1000
C-Net 450	1985	Německo, Rakousko, Portugalsko, JAR	460-465,74	450-455,74	10	573

Další systémy první generace používané ve světě:

Označení systému	Zahájení provozu	Země
AURORA-400	1983	Alberta, Kanada
NTT	1979	Japonsko
JTACS	1991	Japonsko
RadioCom	1985	Francie
RTMS	1985	Itálie

9.3.4. Digitální systémy ve světě

Do systémů druhé generace patří digitální celulární mobilní telefony, které mají v porovnání se systémy první generace mnohé výhody, plynoucí hlavně z použití časového multiplexu TDMA a používáných metod digitálního zpracování signálů. Mezi hlavní výhody bezesporu patří:

- Efektivnější využití přidělených frekvenčních pásem a z toho plynoucí vyšší provozní kapacita sítě
- Vyšší kvalita pojení
- Vysoká úroveň zabezpečení
- Menší rozměry, hmotnost a energetická náročnost mobilních stanic
- Více nabízených služeb, které kromě přenosu hovorových kanálů umožňují přenos textových zpráv (SMS), dat apod.

- Snadná možnost zavedení mezinárodního roamingu v rámci států používajících jednotný systém
- Kompatibilita s fixními sítěmi ISDN i většinou ostatních perspektivních komunikačních systémů

	1. generace	2. generace	3. generace
Celulární systémy	analogové systémy NMT, C-NET, TACS, ...	digitální systémy GSM, D-AMPS, ...	UMTS
Bezšňůrové systémy	analogové systémy CT0, CT1, ...	digitální systémy CT2, DECT, ...	

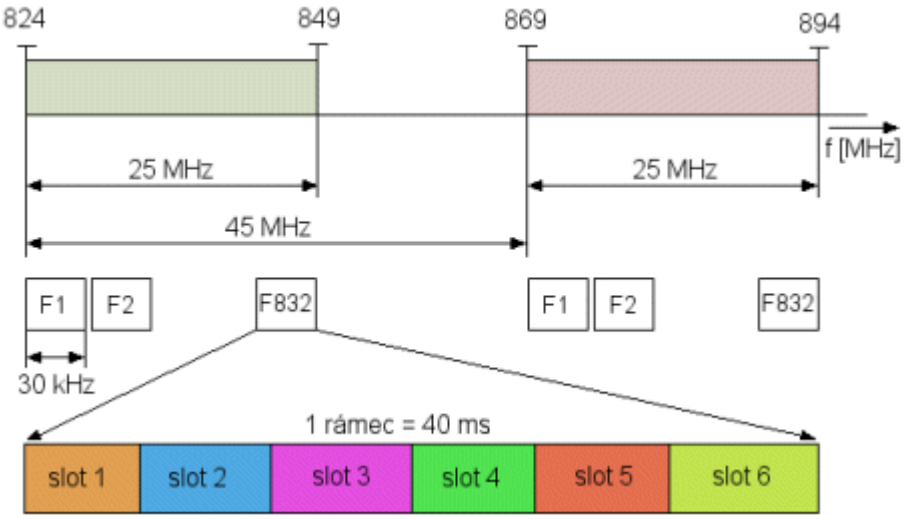
9.3.4.1. Přehled základních digitálních systémů celulární mobilní komunikace

9.3.4.1.1. GSM

V Evropě (především) se od počátku devadesátých let uplatňuje globální systém pro mobilní komunikaci **GSM**, pracující v okolí frekvence 900 MHz. Kapacita frekvenčního pásma se však ukázala nedostatečnou, a proto bylo třeba hledat další kmitočtová pásma pro uspokojení kapacitních požadavků. Byla tak vytvořena varianta systému GSM, pracující v okolí frekvence 1800 MHz, nazývaná **GSM 1800** nebo také **DCS 1800.** Výhodou tohoto systému je kompatibilita se systémem GSM 900 a tedy jednoduchá výroba mobilních stanic pracujících v obou pásmech. Tyto duální stanice jsou dnes běžným standardem. Systém GSM také překročil hranice Evropy, kde byl však použit především v jiném kmitočtovém pásmu. Jedná se o systém **GSM 1900** (DCS 1900 nebo PCS 1900), kterému bylo v USA přiděleno kmitočtové pásmo 1850-1910/1930-1990 MHz. Tento systém se od GSM 900 liší právě použitými frekvencemi a dále výkonovými úrovněmi. Další variantou systému GSM je systém **GSM-R** (Railway), který schválili evropští provozovatelé železnic jako jednotný systém používaný v železniční dopravě. Uvažuje se i o výrobě zařízeni pracujícího podle standardu GSM, avšak v pásmu 450 MHz. Výhodou tohoto systému by bylo lepší šíření rádiových vln a velice výhodné by bylo použití např. v odlehlých, řídce osídlených oblastech.

9.3.4.1.2. ADC, IS-54, D-AMPS

S malým zpožděním za GSM se na americkém kontinentu začal budovat systém **ADC** (Amrican Digital Cellular), nazývaný také **D-AMPS** (Digital-Advanced Mobile Phone System), nebo **IS-54** (Iterim Standard-54). Jedná se o standard vypracovaný asociacemi EIA (Electronic Industries Association) a TIA (Telecommunications Industries Association). Z důvodu kompatibility se starším analogovým systémem AMPS je šířka kanálu 30 kHz a je použito stejné kmitočtové pásmo 824-849/869-894 MHz. K dispozici je 832 rádiových duplexních kanálů. Systém je založen na TDMA/FDMA přístupu a duplexu FDD. Datový tok modulovaný pomocí modulace $\pi/4$ DQPSK má rychlost 48,6 kbit/s. Rámec o periodě 40 ms je rozdělen do 6 časových intervalů (každý je tedy schopen přenést 8,1 kbit/s) po 324 bitech.



Systém používá ekvalizaci pro potlačení mezisymbolové interference, ale v mnoha případech není použití ekvalizátoru nutné. Hlasový signál je komprimován metodou **VSELP** (Vector Sum Excited Linear Predictor), který má výstupní bitovou rychlost po zabezpečení 13 kbit/s. Dalších 3,2 kbit/s je využito pro řízení, synchronizaci apod. Pro jeden hovorový kanál je tedy potřeba využít 2 časové intervaly, tj. 16,2 kbit/s. Na jeden rádiový kanál pak lze umístit 3 účastnické kanály (6 při použití kodeku s poloviční rychlostí), do celého přiděleného pásma pak 4992 (6 x 832) účastnických kanálů. Mobilní stanice jsou dvoumódové, tzn. mohou spolupracovat s analogovým systémem D-AMPS i se systémem AMPS.

9.3.4.1.3. ADC, IS-136, D-AMPS

Počáteční verze systému IS-54 používala signalizační systém AMPS (10 kbit/s). Později byl vytvořen nový standard signalizace **IS-136** (DCC-Digital Control Channel), který má rychlost 48 kbit/s a umožňuje přenos krátkých zpráv, uzavřené skupiny a taktovaný pagingový kanál pro podporu sleep modu u mobilní stanice z důvodu šetření energie. Z důvodu těchto odlišnotí se systém nenazývá IS-54 ale IS-136.

9.3.4.1.4. JDC, PDC

V roce 1991 byl nasazen v Japonsku systém **JCD** (Japanese Digital Cellular), nazývaný též **PDC** (Personal Digital Cellular). Jedná se o systém pracující v pásmech 810-829/940-956 MHz a 1429-1453/1477-1501 MHz. Systém používá TDMA/FDMA přístup se třemi časovými intervaly na jeden rádiový kanál o šířce 25 kHz a obousměrná komunikace je zabezpečena kmitočtovým duplexem FDD. Celkový počet rádiových kanálů je tedy 1600, účastnických je 4800. Systém používá modulaci π/4 DQPSK; pro kódováni hlasového signálu je použito algoritmu VSELP a handover je typu MAHO.

9.3.4.1.5. ADC, IS-95, cdmaOne, CDMA

K digitálním systémům 2. generace patří také vývojově nejmladší americký standard **IS-95**, nesoucí také názvy **cdmaOne** nebo **CDMA**. Systém používá přístupovou metodu CDMA/FDMA a kmitočtový duplex. Systémy CDMA mají v mobilní komunikaci řadu výhod. Největší z nich je možnost pracovat bez vzájemného rušení v těch frekvenčních pásmech, kde již existují jiné rádiové služby. Přínosem je také imunita těchto systémů vůči interferencím a dalším rušivým efektům, typickým pro pozemské kanály. Z důvodu zpětné kompatibility je opět použito shodné frekvenční schéma 869-894/824-849 MHz. Mobilní terminály pracují v kanálech o šířce 1250 kHz, kterých je 20. Jeden rádiový kanál pojme až 798 účastnických kanálů a celkem je tedy k dispozici 15960 duplexních uživatelských kanálů. Systém pracuje s

QPSK a měkkým handoverem. Stejně jako IS-54 je i systém IS-95 kompatibilní se signalizačním systémem AMPS a terminály jsou schopny pracovat v obou módech (AMPS i IS-95)

9.3.5. Historie systému GSM

V průběhu 80tých let svět zaznamenal rychlý růst analogových celulárních systémů v Evropě, především pak ve Skandinávii, Velké Británii, Francii a Německu. Každá země měla vyvinutý svůj systém, který však byl neslučitelný s jakýmkoliv jiným systémem celulární komunikace. Tato situace byla neudržitelná nejenom z důvodu nepoužitelnosti zařízení za hranicemi země, které ve sjednocující se Evropě ztrácely na významu, ale také z důvodu velmi omezeného trhu pro jednotlivé typy zařízení, což s sebou přinášelo ekonomické problémy. Proto Konference evropských správ a pošt CEPT vytvořila v roce 1982 novou standardizační skupinu **GSM** (Groupe Spécial Mobile), která měla za úkol vytvořit standardy pro nový digitální systém, který by byl kompatibilní v zemích celé Evropy (světa). Navrhnutý systém musel splňovat určitá kritéria:

- Perfektní subjektivní kvalita přenášené řeči
- Nízká cena vybavení a služeb
- Podpora mezinárodního roamingu
- Frekvenční hospodárnost
- ISDN slučitelnost
- Efektivitu v budoucnosti

V roce 1989 byla odpovědnost za standardizaci tohoto systému přesunuta na Evropský telekomunikační normalizační institut (ETSI) a v roce 1990 byla specifikace fáze 1 sítě GSM prohlášena standardem.



Komerční provoz první GSM sítě byl zahájen v polovině roku 1991 a již v roce 1993 existovalo 36 GSM sítí v 22 zemích. Ačkoliv byl standardizován v Evropě, GSM není jen evropský standard, ale například i Jižní Afrika, Austrálie a mnoho dalších zemí středního a dálného východu zvolily z hlediska kompatibility tento systém. S jistým zpožděním použili tuto technologii i v USA, kde pod názvem PCS 1900 pracuje na odlišných frekvencích. Systém GSM tak existuje na všech kontinentech a zkratka GSM je interpretována jako "**Global System for Mobile Communication**", tedy "Globální systém pro mobilní komunikaci". Analogové buňkové systémy jako například AMPS v USA nebo TACS ve Velké Británii začaly pomalu upadat a v současné době je systém GSM velice rozšířeným mobilním komunikačním prostředkem. V květnu 2001 dosáhl počet uživatelů GSM 900/1800/1900 na celém světě 500 miliónů. V České republice byl systém GSM spuštěn v roce 1996 společností Eurotel a dále následován společnostmi Radiomobil a Český mobil.

9.3.6. Základní struktura sítě GSM

9.3.6.1. Mobilní uživatelské stanice – MS (Mobile Station)

Mobilní telefon je v podstatě transciever (vysílač/přijímač) komunikující se základnovou stanicí BTS doplněný řídícími obvody (mikroprocesor) a vstupně/výstupními zařízeními (klávesnice, display, sluchátko, mikrofon, porty). Mobilní stanice je jednoznačně identifikována pomocí čísla IMEI (International Mobile Equipment Identity), uloženého v její paměti. Sám účastník je identifikován pomocí SIM karty (Subscriber Identification Module).

9.3.6.2. Subsystém základnových stanic – BSS (Base Station Subsystem)

Jedná se o subsystém, se kterým prostřednictvím rádiového rozhraní komunikují jednotlivé mobilní stanice MS. Samozřejmě není možná komunikace jednotlivým mobilních stanic přímo mezi sebou. Mobilní stanice nepatří do BSS ani do žádného z dalších subsystému, ale jsou samostatnou součástí systému GSM.

9.3.6.3. Síťový spojovací subsystém – NSS (Network Switching subsystem)

Jde vlastně o systém mobilních, resp. radiotelefonních ústředen. Na rozdíl od klasických telefonních ústředen však celý tento systém vykonává kromě obvyklých přepojovacích funkcí ještě mnoho dalších činností, vyplývajících z mobility účastníků (určování polohy, handover, přidělování kanálů apod.).

9.3.6.4. Operační a podpůrný subsystém – OSS (Operational and Support Subsystem)

Cílem tohoto subsystému je zajišťovat řádnou činnost a údržbu celé sítě GSM. Je zde prováděn monitoring, diagnostika a opravy poruch systému a mnoho dalších úkonů. Další funkcí tohoto bloku je také administrativní podpora.

9.3.6.5. Systém GSM dále spolupracuje s externími složkami.

Uživatelé systému, účastníci (subscribers): Ti přicházejí do styku se systémem pouze prostřednictvím svých mobilních stanic MS.

Operátoři: Budují a provozují systém GSM. Pod dohledem regulačních orgánů se starají o stránku technickou, administrativní i finanční.

Externí telekomunikační sítě: K těmto sítím náleží především veřejná komutovaná telefonní síť PSTN (Public Switching Telecommunication Network) a digitální sítě integrovaných služeb ISDN (Integrated Services digital Network), dále například sítě PSPDN (Packet Switched Public Data Network), PLMN (Public Land Mobile Network) nebo družicové telekomunikační systémy.

9.3.6.5.1. MS

Na následujícím obrázku je ve formě tzv. síťového (blokového) schématu znázorněna celková struktura systému GSM. Zobrazený systém je rozdělen na tři subsystémy a jsou zde uvedeny i významy jednotlivých zkratek.

Dle specifikací GSM se mobilní stanicí rozumí jednak vlastní mobilní zařízení (vysílač/přijímač), navíc také předplatitelský identifikační modul SIM (Subscriber Identification Module) - ten umožňuje unikátní identifikaci uživatele v rámci celé sítě GSM.

- **Mobilní zařízení (Mobile Equipment)** obsahuje rádiový přijímač/vysílač, pomocí kterého komunikuje se základnovými stanicemi. Mezi další bloky náleží předmodulační a podetekční obvody, mající na starost zdrojové a kanálové kódování, dále mikroprocesorové obvody a další pomocné bloky. Uživatel přichází do styku s klávesnicí, displejem, sluchátkem a mikrofonem. Pro některé účely, především komunikaci MS s dalším zařízením je nutné použít další rozhraní. Mobilní zařízení jej jednoznačně identifikováno pomocí čísla IMEI (International Mobile Equipment Identity), uloženého v její paměti.
- **Karta SIM (Subscriber Identification Module)** je nutnou součástí mobilní stanice a ta je bez této karty nepoužitelná (s výjimkou tísňového volání 112). V obvodech této karty jsou uloženy specifické údaje o právoplatném majiteli karty, dále jeho čtyřmístné identifikační číslo PIN (Personal Identification Number) a neměnné identifikační číslo PUK (Personal Unblocking Key). Hlavním smyslem SIM karty je ověření a identifikace uživatele a služeb jemu přístupných. Karta je přenosná a lze ji použít s libovolným mobilním telefonem. SIM dále uchovává následující údaje :

- IMSI (International Mobile Subscriber Identity)
- Autentikační klíč (Ki)
- Šifrovací klíč (Kc)
- TMSI (Temporary Mobile Subscriber Identity)
- LAI (Location Area Identity)

Podle maximálního vysílacího výkonu se mobilní stanice dále dělí do pěti výkonnostních skupin.

GSM Class	1	2	3	4	5
max. výkon	20W	8W	5W	2W	0,8W

Uvedené výkony jsou maximální pro danou třídu, v praxi je výkon adaptivně nastavován. Většina dnešních příručních mobilních stanic spadá do kategorie GSM Class 4.

9.3.6.5.2. BSS

Základními stavebními prvky subsystému BBS jsou následující zařízení:

- **Základnové stanice BTS (Base Transciever Station)**
- **Základnová řídicí jednotkaBSC (Base Station Controller)**
- **Transkodér TC (TransCoder)**

V systému GSM obvykle obsahuje jeden svazek 9 buněk, používají se však i svazky s menším počtem buněk. Uvnitř každé buňky je umístěna základnová stanice (není-li využito principu sektorizace), která zajišťuje komunikaci s mobilní stanicí. Tato komunikace probíhá přes rádiové rozhraní (Air interface), označované jako rozhraní Um nebo I/F. Několik základnových stanic je společně řízeno jedinou základnovou řídicí jednotkou BSC. Jednotlivé BTS mohou spolu s BSC vytvářet hvězdicovou, kaskádní, stromovou i jinou topologii. Základnové stanice se dělí do osmi výkonových tříd s výkony 2,5; 5; 10; 20; 40; 80;1 60; 320 Wattů a komunikace s BSC je často realizována radioreléovými spoji. Stejně jako v případě komunikace BSC - MSC však může být použito optického nebo metalického vedení. Základnové řídicí jednotky BSC výrazně odlehčují ústředně MSC tím, že vykonávají funkce handoveru (v systému GSM typu MAHO), přidělování kanálů (dynamické) a částečně také plní funkce přepojovací. Mezi BTS a BSC se nachází rozhraní Abis. Na rádiovém rozhraní (Um) má telefonní kanál přenosovou rychlost 13 kbit/s, na rozhraní Abis již 16kbit/s. Rozhraní mezi základnovou řídicí jednotkou a ústřednou MSC se nazývá rozhraní A. Mobilní ústředna však z důvodu kompatibility s externími sítěmi používá telefonní kanály s rychlostmi 64 kbit/s. Proto je nutností zařdit mezi BSC a MSC transkódovací jednotku TC (TransCoder), která má na starost přizpůsobení rychlostí. Tato jednotka může být umístěna jednak na straně BSC nebo na straně MSC, což je z ekonomických důvodů výhodnější.

9.3.6.5.3. NSS

Základními stavebními prvky subsystému NSS jsou následující zařízení:

- **Mobilní radiotelefonní ústředna MSC (Mobile Switching Centre)**
- **Domovský lokační registr HLR (Home Location Register)**
- **Návštěvnický lokační registr VLR (Visitor Location Register)**
- **Centrum autentičnosti AuC (Authentication Centre)**
- **Identifikační registr mobilních stanic EIR (Equipment Identity Register)**
- **Jednotkou spolupráce s externími sítěmi IWF (Inter-Working Functionality)**
- **SMS Centrum**

Síťový spojovací subsystém plní v systému GSM především spojovací funkce, obdobně jako je uskutečňuje klasická telefonní ústředna. Tuto funkci plní v subsystému NSS mobilní radiotelefonní ústředna MSC. Jde zde o běžný typ telefonní ústředny, která je však doplněna o další funkce plynoucí z mobility přepojovaných účastnických stanic. Tato ústředna je nadřazena nad systémem řadičů BSC a jedna nebo více z nich plní funkci tzv. Gateway MSC a

umožňuje propojení mobilní sítě GSM s externími telekomunikačními sítěmi, fixními nebo mobilními. Za tímto účelem je potřeba ústřednu vybavit jednotkou spolupráce s externími sítěmi IWF (Inter-Working Functionality)

Subsystém NSS dále realizuje celou řadu specifických úloh, spojených s mobilitou účastníků. Součástí každé sítě GSM je domovský lokační registr HLR, což je v podstatě hlavní databáze, ve které jsou uložena veškerá důležitá data o uživatelích sítě. Obsahuje důležitá čísla IMSI (International Mobile Subscriber Identity), MSISDN (Mobile Subscriber ISDN Number), zpřístupněné služby a dále například údaje týkající se polohy uživatele. V síti jednoho operátora je vždy minimálně jeden HLR, může jich být i více. Součástí registru HLR je i centrum autentičnosti AuC (Authentication Centre), což je chráněná databáze, obsahující klíče pro ověřování totožnosti účastníků. Toto centrum má dále na starost šifrovací klíč, podle kterého se šifruje každý účastnický signál přenášený rádiovým rozhraním. Tento klíč je unikátní pro každého účastníka a je proměnný v čase. Na registr je dále napojen identifikační registr mobilních stanic EIR. V této databázi jsou uložena čísla IMEI (International Mobile Equipment Identity) mobilních stanic, které jsou autorizovány k použití v dané síti, dále čísla ukradených MS a je zde i seznam stanic, které jsou označeny jako porouchané, případně nesplňující určitá požadovaná specifika. Velmi důležitou roli hraje také návštěvnický lokační registr VLR, který přechodně uchovává a obnovuje data o uživatelích, v dané chvíli se nacházejících v oblasti příslušné MSC. Obsahuje podobné informace jako HLR , ale pouze dočasně, tzn. že jakmile účastník opustí oblast, data jsou vymazána.

K přenosu signalizace mezi jednotlivými zařízeními je použita signalizační síť SS7 a jsou definovány následující rozhraní:

- **B** mezi MSC a VLR
- **C** mezi MCS a HLR
- **D** mezi registry VLR A HLR
- **E** mezi dvěma ústřednami MSC
- **F** mezi MSC a EIR
- **G** mezi HLR a AuC

9.3.6.4. Operační a podpůrný subsystém – OSS

Základními stavebními prvky subsystému OSS jsou následující zařízení:

- **Provozní a servisní centrum OMC (Operational and Maintenance Centre)**
- **Centrum pro řízení sítě NMC (Network Management Centre)**
- **Administrativní centrum ADC (Administrative Centre)**

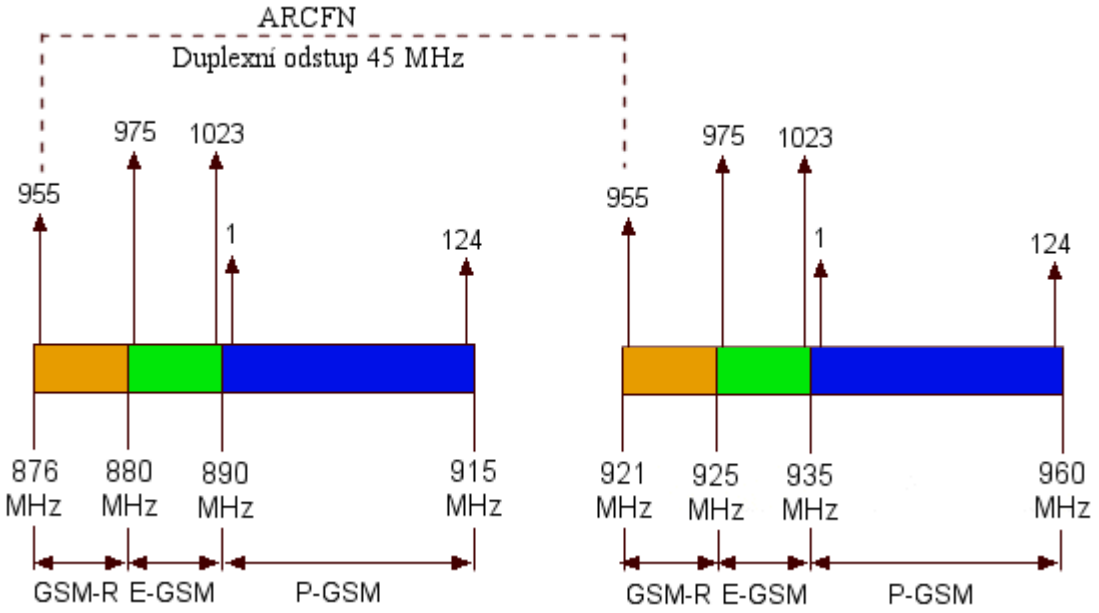
Tento subsystém provádí zejména následující základní funkce:

- Podílí se na managementu mobilních stanic, tyto stanice monitoruje a eviduje porouchané
- Dohled a konfigurace sítě
- Provádí kontrolu a údržbu systému GSM
- Podílí se na administrativním managementu účastníků GSM, konkrétně jejich registraci, tarifikaci

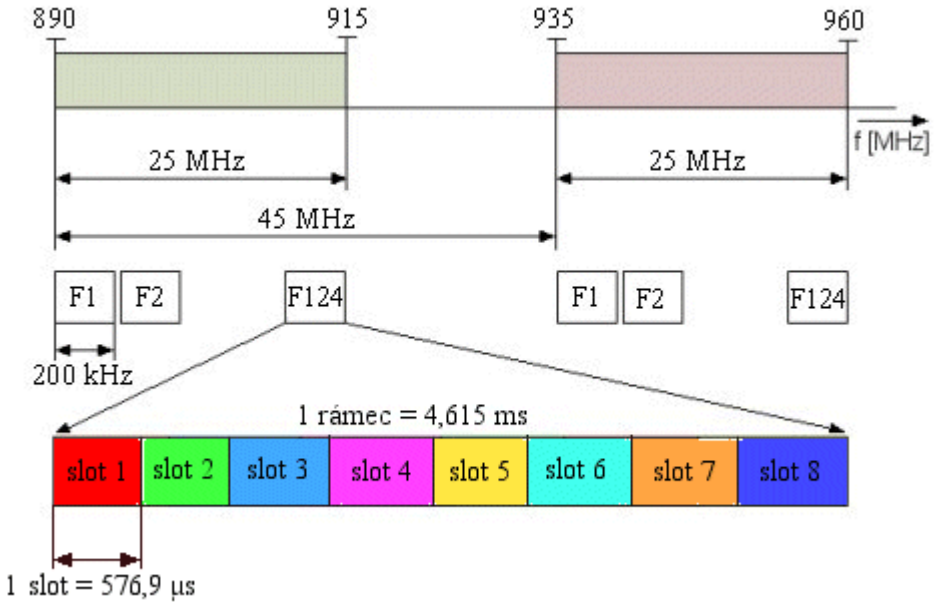
9.3.6.6. Rádiové rozhraní systému GSM

Systém GSM má vyhrazena dvě rádiová pásma o šířce 2x25 MHz. Pro vzestupnou trasu (vysílají mobilní stanice) je to 890-915 MHz a pro trasu sestupnou (vysílaji základnové stanice) je to 935-960 MHz. Systém poskytuje plně duplexní provoz ve formě frekvenčního duplexu FDD s duplexním odstupem 45 MHz. Nosné vlny mají vzájemný odstup 200 kHz, takže v pásmu 25 MHz je jich celkem 125. Kanál č. 0 je oddělovací a nepoužívá se pro přenos hovorů, využitelných je tedy 124 duplexních kanálů. Každá dvojice uplink/downlink je potom označena absolutním číslem rádiového frekvenčního kanálu ARFCN (Absolute Radio Frequency Channel Number). U systému GSM nabývá ARFCN hodnot 1-124. Kapacita frekvenčního

pásma v oblasti 900 MHz se ukázala nedostačující, a proto bylo rozhodnuto o použití dalších kmitočtových pásem. Vznikl tak systém nazývaný **GSM 1800 (DCS 1800)**, který je založen na standardech GSM a je koncipován pro nasazení v buňkách malých rozměrů. Liší se především v použitém kmitočtovém pásmu (1710-1785/1805-1880), které poskytuje kapacitu 375 rádiových kanálů (ARCFN 512-885). Šířka přiděleného pásma je tedy 75 MHz a duplexní odstup 95 MHz. V USA je systému **PCS 1900 (DCS 1900, GSM 1900)** přiděleno pásmo 1850-1910/1930-1990, což znamená že šířka pásma je 60 MHz a odstup 80 MHz. Použít lze 300 rádiových (ARFCN 512–810) a 2400 uživatelských kanálů. Dále vznikla novější varianta GSM, označená **E-GSM (Extended-GSM)**, kde je rozšířeno frekvenční pásmo standardní P-GSM (Primary-GSM) na 880-915/925-960 MHz (ARFCN jako u P-GSM a navíc 975-1023) a zmínit je třeba i variantu **GSM-R (Railway GSM)**, kterou odsouhlasili provozovatelé železnic jako jednotný komunikační systém pro použití v železniční dopravě. Pro tuto službu ETSI vymezila pásmo 876-880/921-925 MHz (odpovídá ARCFN 955-974).



Na každém rádiovém kanálu je metodou TDMA vytvořeno 8 časových slotů, přičemž každý interval představuje 1 uživatelský kanál. Celkem je tedy u standardního systému GSM 900 k dispozici 8 x 124 = 992 duplexních kanálů (3000 u varianty GSM 1800). To ovšem platí při zdrojovém kódování "plnou rychlostí", tzv. full rate. Zavedením dokonalejšího a účinnějšího zdrojového kódování half rate je potom možno docílit přenosu 16 hovorových kanálů na jedné nosné vlně. Uvedený způsob přenosu s kombinovaným frekvenčním a časovým multiplexem se pak označuje zkratkou **FDMA/TDMA**.



Každý digitalizovaný hovorový kanál má po zakódování v kodéru zdroje přenosovou rychlost 13 kbit/s. Ta se dále po přidání ochranných bitů, prováděném při kanálovém kódování, zvýší na 22,8 kbit/s. Sdružením takových osmi kanálů a přidáním dalších pomocných a signalizačních bitů dojdeme k celkové přenosové rychlosti signálu připadajícího na jednu frekvenci o hodnotě 270,883 kbit/s. Odpovídající perioda jednoho bitu je potom 3,692 µs a efektivní přenosová rychlost na jeden kanál 33,854 kbit/s. Jako optimální modulační metoda byla vybrána pro systém GSM gaussovská modulace MSK, tj. **GMSK** s normovanou šířkou pásma předmodulační gaussovské dolní propusti B x T = 0,3.

Odolnost proti selektivnímu úniku je zajištěna tzv. ekvalizací. Při ekvalizaci je do přenosového řetězce na straně přijímače zařazen filtr, který kompenzuje vlastnosti přenosového kanálu na přijímací straně. Proto je v každém slotu vysílána tréninková posloupnost (známá posloupnost bitů) a v přijímači se deformovaný signál porovná se správným signálem a podle výsledku se nastaví parametry ekvalizačního filtru. Ekvalizace je málo účinná v prostředích, kde vznikají dlouhé výpadky signálu, a proto základnová stanice v každém časovém slotu mění kmitočet komunikace (mobilní stanice se jí přizpůsobují). Použitá ekvalizace spolu se strukturou rámců umožňuje používat mobilní stanice až do rychlosti 250 km/h, což je i maximální rychlost pro úspěšný handover. Mobilní stanice neustále sleduje kvalitu rádiových kanálů nejen z hlediska intenzity signálu ale také z hlediska bitové chybovosti BER. Měří se až 6 sousedních základnových stanic a na základě těchto údajů BSC nebo MSC rozhoduje o handoveru.

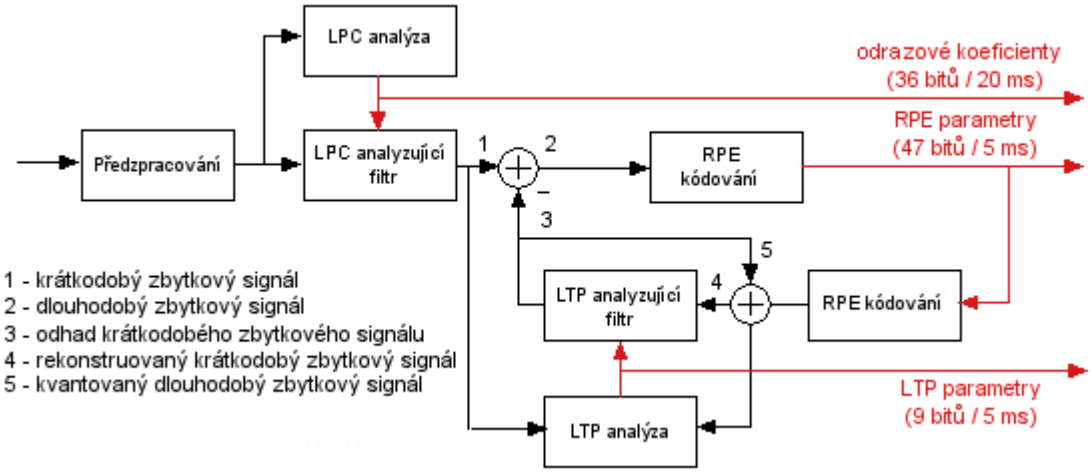
9.3.6.7. Zpracování signálu v systému GSM

Na vstup vysílací části mobilní stanice přichází z mikrofonu analogový elektroakustický (hovorový) signál. Ten je nejprve v kodéru zdroje signálu digitalizován a zbavován redundance, čímž se výrazně snižuje jeho bitová rychlost. V následujícím kodéru je zvětšována jeho imunita vůči rušivým vlivům, působícím v pozemském rádiovém kanálu. Zakódovaný signál je poté podroben prokládání (interleaving) a dále se uskutečňuje formátování burstů. Následuje proces šifrování a takto upraveným signálem se poté v modulátoru moduluje nosná vlna. Modulovaný signál je po frekvenční transpozici a výkonovém zesílení vysílán. Na přijímací straně probíhá tentýž proces v obráceném pořadí, navíc je do řetězce zapojen proces ekvalizace.

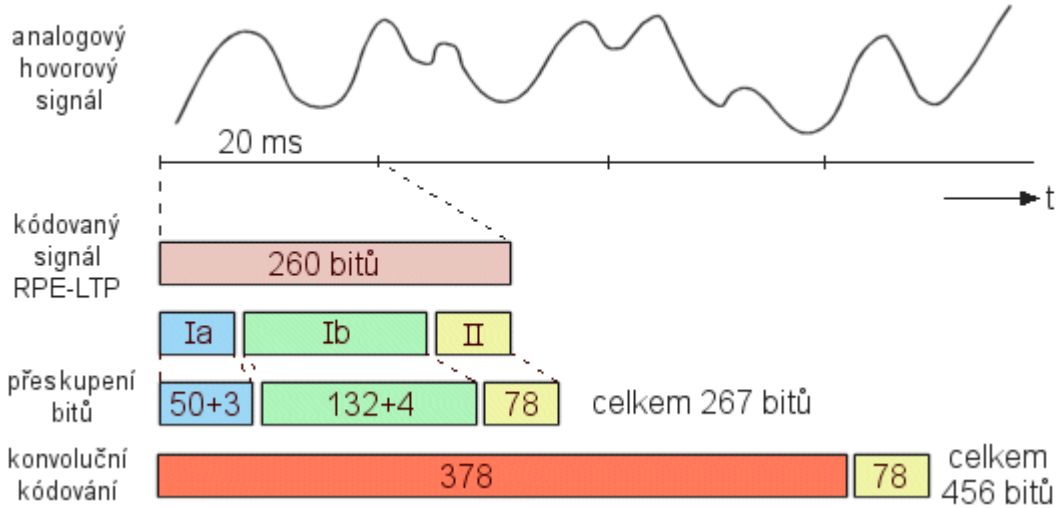
Na následujícím obrázku jsou schematicky znázorněny funkce uskutečňující se ve vysílači a v přijímači mobilní stanice. Princip zde použitý je v podstatě shodný s principem použitým v jakýmkoliv jiném digitálním rádiovém komunikačním systému. Animace: http://tomas.richtr.cz/mobil/gsm-sig.htm

9.3.6.7.1. Zdrojové a kanálové kódování

V systému GM se používá moderní parametrická metoda zdrojového kódování, označovaná zkratkou **RPE-LTP** (Regular Pulse Excitation – Long Time Prediction), umožňující redukovat rychlost nezakódovaného hovorového signálu 64 kbit/s na 13 kbit/s. Pro buzení filtru na přijímací straně je přenášen zakódovaný zbytkový rozdílový signál mezi originálním hovorovým signálem a signálem syntetickým. Dlouhodobá predikce spočívá ve využití korelací mezi sousedními periodami základního tónu hovorového signálu k predikci hodnoty signálu v následující periodě. Na následujícím obrázku je schéma kodeku RPE-LTP.



Analogový elektroakustický signál je v kodéru vysílače členěn do časových segmentů o délce 20 ms. Procesem kódování jsou potom jednotlivé segmenty převedeny na digitální hovorové rámce po 260 bitech, čemuž odpovídá přenosová rychlost 260/0,02 = 13 kbit/s. Každý rámec obsahuje 188 bitů tzv. excitačního signálu, z něhož se v dekodéru přijímač rekonstruuje požadovaný výstupní signál. Dále je zde 2 x 36 = 72 bitů, reprezentujících koeficienty digitálních filtrů. Tyto bity jsou dále přeskupeny a uspořádány do skupin vhodných pro následující ochranné kanálové kódování. Při tomto procesu se přihlíží k významu jednotlivých bitů pro zajištění dostatečné kvality přenosu hovorových signálů.



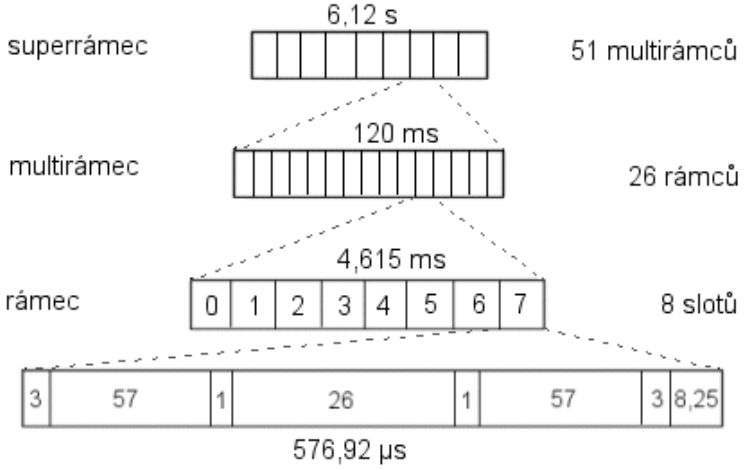
Z původních 260 bitů se použije nejprve prvních (co se důležitosti týče) 50 bitů typu I_a a ty jsou zabezpečeny kódem CRC (jsou přidány 3 kontrolní bity). Následujících 132 bitů typu I_b je doplněno o 4 nulové bity a vytvoří se tak blok bitů typu I o 189 bitech. Tento blok je poté podroben konvolučnímu kódování (rychlost 1/2, omezovací délka K=5), čímž získáme 378 bitů, které jsou dále sloučeny se zbývajícími 78 nejméně důležitými bity typu II, které se protichybově nekódují. Výsledný blok má pak délku 456 bitů a odpovídá časovému segmentu 20 ms. Ochranným kódováním se tak zvyšuje přenosová rychlost zakódovaného signálu z původních 13 kbit/s na 456/0,02=22,8 kbit/s.

- V doporučeních je také popsána varianta kodeku half-rate, jehož výstupní bitová rychlost je poloviční, tedy 6,5 kbit/s a umožňuje přenos dvojnásobného počtu hovorů. Dále je k dispozici kodek EFR (Enhanced Full Rate), což je kodek s plnou rychlostí avšak výrazně lepšími parametry (kvalita hlasu).
- K ušetření až 90 % energie je u sytému SM použito tzv. nespojitého (diskontinuálního přenosu). Princip spočívá ve faktu, že při běžné konverzaci mluvčí hovoří pouze přibližně 40 % času a během doby kdy nemluví není potřeba přenášet signál. K tomuto jsou mobilní stanice vybaveny obvodem detekce hlasové aktivity (Voice Activity Detection). Protože je systém GSM digitální, bylo by v době ticha na přijímací straně opravdu nepřirozené ticho a telefon by se jevil jako vypnutý či rozbitý. Z tohoto důvodu je v těchto časových okamžicích generován šum.

9.3.6.7.2. Prokládání a formátování burstů

Pro dosažení co nejvyšší odolnosti systému GSM vůči rušivému působení shluků chyb je kanálové konvoluční kódování doplněno ještě tzv. diagonálním prokládáním (interleaving). Každý bitový blok o délce 456 bitů vytvořený při předchozím kanálovém kódování, je rozdělen na 8 skupin po 57 bitech. Ty jsou poté metodou diagonálního prokládání proloženy s posledními čtyřmi skupinami z předchozího bloku a prvními čtyřmi skupinami z bloku následujícího. V takto upraveném signálu již dvě sousední skupiny 57 bitů nenáleží jedinému, ale dvěma různým blokům.

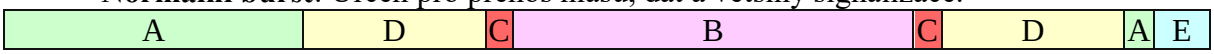
Dvě sousední bitové skupiny délky 2 x 57 bitů se označují termínem burst. Po doplnění tréninkové sekvence, využívané pro účely ekvalizace, a několika dalších pomocných bitů mají bursty dobu trvaní 0,577 ms, což je doba trvání jednoho časového intervalu. Takto upravený burst se nazývá normální burst a je základní jednotkou přenosu v systému GSM. Prokládání má velký význam v tom, že je při přenosu změněno pořadí bitových skupin a dojde li tedy v průběhu přenosu k typickému shluku chyb, budou postižené bity náležet do různých bitových skupin. Po inverzní operaci v přijímači se tak získá signál, u něhož chybně přenesené bity nesousedí a shluk chyb je tak převeden na rozptýlené osamocené chyby, které nejsou již tolik závažné a je možno je korigovat konvolučním dekodérem.



Kompletní normální burst je situován do časového intervalu o délce 0,577 ms. Obsahuje celkem 2 skupiny po 57 bitech, nesoucí vlastní informaci a dále tréninkovou ekvalizační sekvenci o 26 bitech. Typ ekvalizéru není specifikacemi GSM pevně stanoven – jeho provedení záleží na výrobci mobilní stanice. Zbytek burstu je tvořen 6 okrajových bitů, dva kontrolní bity a 8,25 bitů ochranných. Takto vytvořený burst má délku 156,25 bitů a dobu trvání 0,577 ms. Spojením 8 burstů se vytváří časový rámec TDMA o době trvání 4,615 ms. Dalším spojením 26 časových rámců TDMA vzniká jeden multirámec, spojením 51 multirámců vznikne superrámec a nejvýše v této hierarchii stojí hyperrámec.

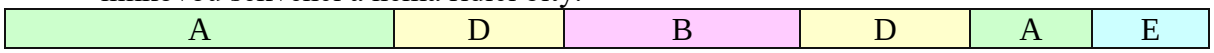
Kromě normálních burstů se v systému GSM používají ještě další typy burstů, všechny jsou popsané dále:

- **Normální burst**: Určen pro přenos hlasu, dat a většiny signalizace.



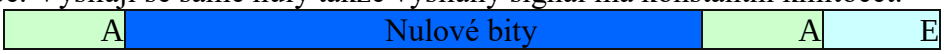
A (3 bity) - okrajové bity, **B** (26 bitů) - tréninková sekvence, **C** (1 bit) - řídicí bity, **D** (57 bitů) - data, **E** (8,25 bitu) - ochranná doba

- **Synchronizační burst**: Užívá se pro synchronizaci mobilní stanice, má rozšířenou tréninkovou sekvenci a nemá řídicí bity.



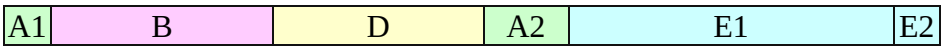
A (3 bity) - okrajové bity, **B** (64 bitů) - tréninková sekvence, **D** (39 bitů) - data, **E** (8,25 bitu) - ochranná doba

- **Burst pro kmitočtovou korekci**: Užívá se pro kmitočtovou synchronizaci mobilní stanice. Vysílají se samé nuly takže vysílaný signál má konstantní kmitočet.



A (3 bity) - okrajové bity, **E** (8,25 bitu) - ochranná doba

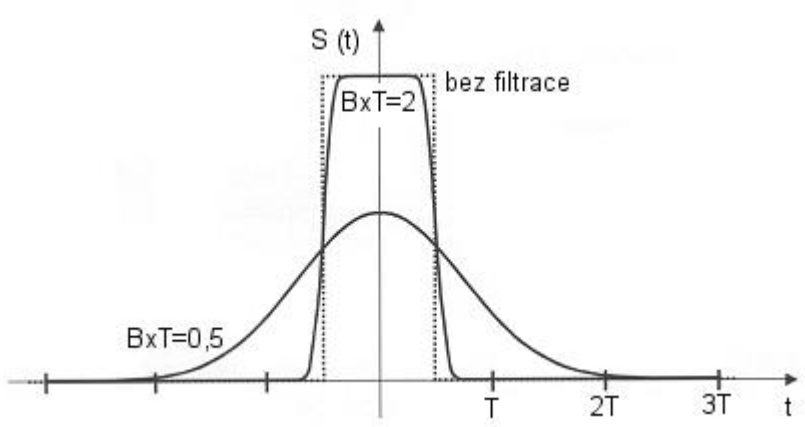
- **Přístupový burst**: Užívá se pro náhodný přístup (žádost mobilní stanice o spojení). Protože ještě není znám vzájemný časový posun mobilní a základnové stanice, má delší ochrannou dobu.



A1 (8 bitů), **A2** (3bity) - okrajové bity, **B** (41 bitů) - tréninková sekvence, **D** (36 bitů) - data, **E1** (60 bitů) - přídavná ochranná doba, **A2** (8,25 bitu) - ochranná doba

9.3.6.7.3. Modulace a demodulace

Zakódovaný digitální modulační signál se přivádí do modulátoru vysílače, kde se jím moduluje nosná vlna. Pro radiotelefonní systém GSM byla jako nejvhodnější modulační metoda vybrána gaussovská modulace s minimálním zdvihem, značená zkratkou **GMSK** (Gaussian Minimum Shift Keying). Jedná se o upravenou variantu modulace MSK. Signál modulovaný metodou MSK zaujímá relativně malé kmitočtové pásmo, zůstává však nepříjemné vyzařování do sousedních pásem, které lze odstranit filtrací. Provádí se tedy předfiltrace datového signálu pomocí gaussovské dolní propusti, odtud pochází název této modulace. Protože časový průběh filtrovaného signálu neobsahuje skokové změny, nemění se skokově ani vysílaný kmitočet. U této modulace je důležitý modulační index (B x T), což je součin šířky pásma gaussovské propusti a doby trvání bitu. Pro malé hodnoty modulačního indexu klesají postranní laloky, ale roste mezi symbolová interference, pro velké hodnoty je tomu naopak.



U systému GSM je použita GMSK s filtrací modulačního signálu gaussovskou dolní propustí s relativní šířkou pásma B x T = 0,3.

Frekvenční skoky nosné vlny

Dvě sousední nosné vlny mají v systému GSM odstup 200 kHz, přičemž každá z nich přenáší 8 nezávislých účastnických kanálů sdružených metodou TDMA. Jeden kanál však nevyužívá v průběhu hovoru jednu a tutéž frekvenci, ale v určitých časových intervalech "přeskakuje" mezi více nosnými frekvencemi. Systém GSM tedy používá pomalé frekvenční skoky **SFH** (Slow Frequency Hopping). Pomalé frekvenční skoky tak přinášejí dvě hlavní výhody.

- Frekvenční diverzita: Pomáhá redukovat ztráty rádiového spojení v oblastech s rychlým Rayleighovým únikem.
- Interferenční diverzita: Zvyšuje účinnost metody opětovného využití frekvencí rádiových kanálů.

Časový předstih ve vysílání

Pro správné fungování multiplexu TDMA je nutné, aby jednotlivé bursty používající sousední časové intervaly dospěly k základnové stanici ve správném čase a nedocházelo tak k překrývání, které vede k nárůstu chybovosti. Závažnost tohoto problému narůstá v okamžiku, kdy je mobilní stanice vzdálena od základnové stanice natolik, že nelze zanedbat dobu šíření rádiových vln. V klidovém stavu je mobilním stanice na příjmu, sleduje návěstní kanál (příchozí hovory) a její časování je oproti časování základnové stanice posunuto. V případě komunikace (odchozí nebo příchozí hovor) požádá pomocí přístupového burstu o přidělení řídicího kanálu. Přístupový burst má prodlouženou ochrannou dobu (252 ms) a umožňuje komunikaci na 37,8 km. Základnová stanice změří zpoždění příchozího signálu a spočítá parametr **TA** (Timing Advance, "předstih"), který pošle mobilní stanici. Mobilní stanic tento parametr používá tak, že signál vysílá v předstihu, daném právě tímto parametrem, který může nabývat hodnot 0 bitů (mobilní stanice velmi blízko u základnové stanice) až 63 bitů. Mobilní stanice tedy nevysílá přesně po uplynutí 3 časových intervalů od příjmu (3 x 165,25 bitů), ale po době kratší (3 x 165,25 - TA bitů). Z tohoto plyne i dosah systému GSM 35 km. Tohoto parametru lze využít i pro lokalizaci mobilní stanice, kdy je pomocí TA určena přibližná vzdálenost mobilní stanice od základnové a v případě porovnání výsledků od více základnových stanic lze dojí k poměrně kvalitnímu určení polohy mobilní stanice.

9.3.6.8. Zabezpečení systému GSM proti zneužití

V systému GSM je třeba, stejně jako v jiných telefonních sítích zabránit nejen odposlechu, ale také zneužití ztraceného telefonu, volání na cizí účet a podobně. Bezpečnost v sytému GSM je obecně rozdělena do dvou hlavních kategorií. První z nich je ověření totožnosti účastníka, mobilní stanice a SIM karty, druhou kategorií je samotný proces šifrování přenášených dat.

Ověření totožnosti uživatele SIM karty probíhá pomocí zadávání číselných kódů PIN (PIN2) a PUK. Ověření totožnosti mobilní stanice může probíhat (záleží na operátorovi) pomocí IMEI (International Mobile Equipment Identity), což je číslo uložené v mobilní stanic a dále v registru EIR. Do registru VLR je zasláno toto číslo, které je pak ověřeno a zařazeno do jednoho ze seznamů:

- White list ("bílý seznam") - stanice, jímž je přístup povolen
- Black list ("černý seznam") - kradené mobilní stanice
- Grey list ("šedý seznam") - porouchané stanice, nebo stanice nepodporující určité specifikace

Jediné slabé místo při přenosu jakýchkoli dat systémem GSM se nachází v jeho rádiové části. Data přenášená mezi mobilní stanicí a základnovou stanicí na jedné straně umožňují téměř neomezenou volnost uživatele mobilního telefonu, na straně druhé však znamenají potenciální nebezpečí odposlechu nebo například telefonování na účet majitele. Systém GSM proto používá řadu mechanizmů, jak tomuto předejít. Jedná se o použité protokoly a formáty dat, digitální modulaci GMSK, neustálé přelaďování stanice na různé frekvence, kanálové kódování a výše popsané ověření totožnosti účastníka a mobilní stanice. Další důležité procesy (ověření totožnosti SIM karty a samotné šifrování dat) jsou popsány v samostatných kapitolách.

9.3.6.8.1. Ověření totožnosti

Proces ověření totožnosti sim karty

Po zapnutí požádá mobilní stanice o přístup do sítě. Zašle tedy své IMSI (International Mobile Subscriber Identity) a toto je jediný okamžik, kdy je používáno toto číslo IMSI. Anonymita je poté zajištěna přidělením tzv. dočasné identifikace TMSI (Temporary Mobile Subscriber Identity), která je uložena na SIM kartě a v registru VLR.

V registru HLR (AuC) je vygenerováno náhodné číslo RAND. Na základě tohoto čísla je pak s pomocí autentifikačního klíče Ki podle šifrovacího algoritmu A3 spočítána odezva SRES (Signed Response) na toto náhodné číslo. Zároveň je podle šifrovacího algoritmu A8 (označovaný též A38) vygenerován šifrovací klíč Kc (viz. šifrování). Tato trojice (RAND - 128 bitů, SRES - 32 bitů a Kc - 64 bitů) je předána do registru VLR, kde je po dobu spojení uchována. Registr VLR dále přepošle náhodné číslo RAND mobilní stanici, která v SIM kartě na základě znalosti klíče Ki a čísla RAND spočte odpověď SRES. Tato odpověd je poslána zpět do VLR, kde je porovnána s odpovědí vygenerovanou v HLR a určí, zda stanice má přístup do sítě. Algoritmus A3 a klíč Ki jsou tedy zabezpečeně uloženy jednak v registru HLR a také v SIM kartě. Při komunikaci tedy rádiovým rozhraním prochází pouze náhodné číslo RAND a odpověď SRES. Algoritmus A3 je vytvořen tak, aby bylo relativně snadné spočítat SRES z údajů Ki a RAND, ale velmi náročné vypočíst Ki z údajů RAND a SRES. Animace: http://tomas.richtr.cz/mobil/gsm-aut.htm

9.3.6.9.2. Šifrovací proces

Šifrování probíhá tak, že je na základě klíče (kombinace znaků) pomocí šifrovacího algoritmu vygenerována bitová posloupnost. Tato šifrovací posloupnost se pak přičítá k přenášené bitové posloupnosti.

V registru HLR (AuC) je vygenerováno náhodné číslo RAND. Na základě tohoto čísla je pak s pomocí autentifikačního klíče Ki podle šifrovacího algoritmu A8 vygenerován šifrovací klíč Kc (zároveň je pomocí algoritmu A3 spočítána odezva SRES - viz. autentikace). Tato trojice (RAND - 128 bitů, SRES - 32 bitů a Kc - 64 bitů) je předána do registru VLR, kde je po dobu spojení uchována. Registr VLR dále přepošle náhodné číslo RAND mobilní stanici, která v SIM kartě na základě znalosti Ki a algoritmu A8 vygeneruje šifrovací klíč Kc. Šifrovací

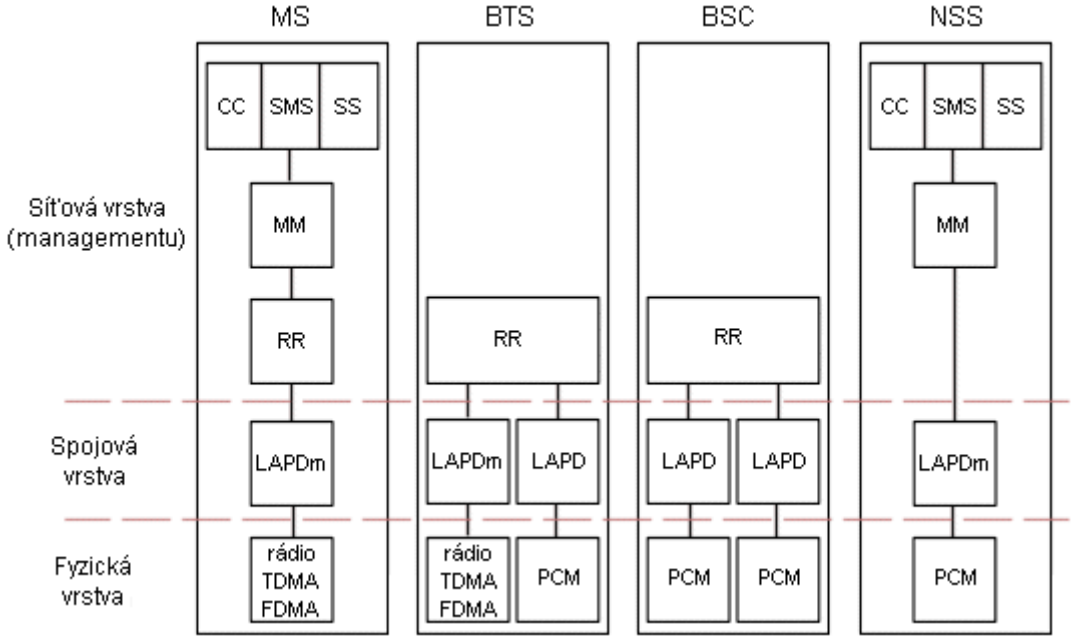
posloupnost, kterou k přenášeným datům přičítáme, je generována pomocí algoritmu A5. Vstupními údaji pro tento algoritmus je šifrovací klíč Kc (64 bitů, generovaný zvlášť pro každé spojení) a číslo TDMA rámce (22 bitů měnících se každých 4,615 ms). Šifrovací algoritmus již není z důvodu dostatečné výpočetní kapacity a rychlosti uložen na SIM kartě, ale je implementován přímo v mobilní stanici. Pokud během hovoru dojde k handoveru, šifrovací klíč Kc se nemění. Ani při šifrování tedy nedochází k přenosu žádného klíče a rádiovým rozhraním jsou přenášena pouze náhodná čísla RAND a šifrovaná data.

Algoritmy A3 a A8 nejsou specifikovány v doporučení, ale GSM MoU (Memorandum of Understnding) a je zde ponechána určitá volnost, ponechávající možnost domluvy mezi provozovatelem sítě a výrobcem SIM karet.

9.3.6.9.3. Vrstvový model, logické kanály

Vrstvový model

Na následujícím obrázku je zobrazen vrstvový model systému GSM, který je definován pro tři spodní vrstvy modelu OSI. Každá vrstva je tvořena skupinou tzv. entit, což jsou všechny do ní spadající funkční jednotky.



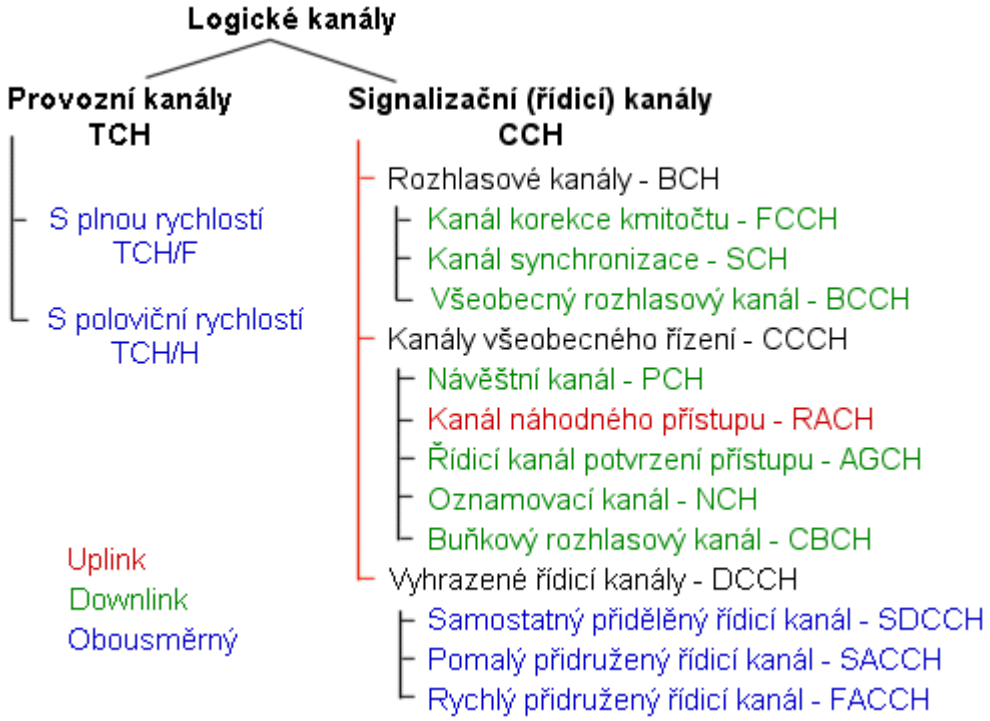
V dalších úvahách je třeba požít další pojmy – fyzické a logické kanály. Fyzickým kanálem je kombinace rádiového kanálu o šířce 200 kHz (označeného pomocí ARFCN) a časového intervalu (označené příslušným číslem 0-7). Do každého takto vytvořeného fyzického kanálu potom může být v rozdílných časech procesem mapování vložen různý logický kanál.

- Fyzická vrstva **PH** (Physical Layer): Definuje rádiové fyzické kanály, tedy parametry TDMA, FDMA, modulace a mapuje do těchto kanálů kanály logické.
- Spojová vrstav **DL** (Data Link Layer): Na této vrstvě je použit protokol LAPDm, což je varianta protokolu LAPD, přizpůsobená pro použití v systému GSM. Na této vrstvě jsou vytvářeny logické kanály.
- Síťová vrstva **NL** (Network Layer): Je dále rozdělena na tři podvrstvy:
- Management rádiových zdrojů **RR** (Radio Resource Management), který zajišťuje spojení mezi mobilní stanicí a subsystémem základnových stanic BSS, řídí handover, výkony vysílačů apod. Relace této podvrstvy existují mezi MS a BSS
- Management mobility **MM** (Mobility Management), zajišťující funkce vycházející z mobility účastníků. Na starost tedy má aktualizaci polohy, identifikaci a ověření účastníka. Vazby na této podvrstvě existují mezi MS a MSC/VLR, HLR a AuC.

- Management komunikace **CM** (Comunication Management) je opět podrobněji dělen :
- Řízení hovoru **CC** (Call Control): Zajišťuje vytváření, udržení a ukončení hovorů. Komunikace zde probíhá mezi MS a MSC.
- Doplňkové služby **SS** (Suplementary Services): Má na starost registraci, řízení a rušení doplňkových služeb, poskytovaných danou sítí. SS procedury existují mezi MS a MSC/VR, HLR a Auc.
- Služba krátkých textových zpráv **SMS** (Short Message Service): Zajišťuje mobilní stanici možnost přijímání a odesílaní krátkých textových zpráv. Komunikace probíhá mezi MS a SMS Centrem **SMSC** (Short Message Service Centre).

Logické kanály

Struktrua časových slotů vytváří na nosné frekvenci fyzické kanály. Fyzickým kanálem se tedy rozumí kombinace přiděleného rádiového kanálu o šířce 200 kHz (označeného pomocí ARFCN) a časového intervalu (označené příslušným číslem 0-7). Do každého takto vytvořeného fyzického kanálu potom může být v rozdílných časech procesem mapování vložen různý **logický kanál.** Specifikace GSM definují širokou řadu logických kanálů, které jsou v závislosti na účelu tohoto kanálu tvořeny různými typy burstů.



- **Provozní kanály TCH (Taffic Channels)**
  Tyto kanály jsou určené k přenosu digitalizovaných hovorů nebo datových signálů
- Provozní kanály s plnou rychlostí **TCH/F** (Full Rate Traffic Channels)
- Provozní kanály s poloviční rychlostí **TCH/H** (Half Rate Traffic Channels)
- **Signalizační (řídicí) kanály CCH (Control Channles)**
  Tyto kanály jsou určeny pro přenosu signalizace a dále je dělíme :
  Rozhlasové kanály BCH (Broadcast Channels)
- Kanál korekce kmitočtu **FCCH** (Frequencu Correction Channel): Nese informace umožňující korekci naladění mobilní stanice a identifikaci kmitočtu nesoucího signalizační kanály. Tvoří ho bursty pro kmitočtovou korekci.
- Kanál synchronizace **SCH** (Synchronization Channel): Nese informace pro rámcovou signalizaci mobilní stanice (číslo rámce) a identifikaci základnové stanice. Tvořen je synchronizačním burstem.

- Všeobecný rozhlasový kanál **BCCH** (Broadcast Common Channel): Nese informace o aktuálním způsobu mapování signalizačních kanálů, o výzvách k mobilní stanici (korekce výkonu, ...), o lokalizační oblasti apod. Je tvořen normálním burstem a je sledován každou mobilní stanicí.

  Kanály všeobecného řízení CCCH (Common Control Channel)
- Návěstní kanál **PCH** (Paging Channel): Slouží k předání informace o příchozím hovoru. Tento kanál je sledován každou mobilní stanicí ve stavu pohotovosti, je tvořen normálním burstem
- Kanál náhodného přístupu **RACH** (Random Access Channel): Jedná se o vzestupný kanál, který slouží pro vyžádání samostatného řídicího kanálu pro další signalizaci. Z důvodu náhodné aktivity požadavků mobilních stanic je komunikace v tomto kanále realizována principem ALOHA. Kanál je tvořen přístupovým burstem.
- Řídicí kanál potvrzení přístupu **AGCH** (Access Grant Control Channel) : Slouží pro přidělení samostatného řídicího kanálu mobilní stanici, která předtím o toto požádala. Tvoří jej normální burst.
- Oznamovací kanál **NCH** (Notification Channel) a Buňkový rozhlasový kanál **CBCH** (Cell Broadcast Channel) byly přidány až do fáze 2+ GSM

  Vyhrazené řídicí kanály DCCH (Dedicated Control Channel)
- Samostatný přidělený řídicí kanál **SDCCH** (Stand Alone Dedicated Control Channel): Slouží pro obousměrnou komunikaci mezi mobilní a základnovou stanicí před přidělením provozního kanálu.
- Pomalý přidružený řídicí kanál **SACCH** (Slow Associated Control Channel): Zajišťuje přenos signalizace k exitujícímu spojení.
- Rychlý přidružený řídicí kanál **FACCH** (Fast Associated Control Channel): Podobný účel jako SACCH, vzniká a zaniká však podle potřeby.

9.3.6.9. Přenos dat v systému GSM

9.3.6.9.1. Klasický přenos dat

Mobilní digitální sítě sice byly vyvinuty pro potřebu přenosu hlasu, nicméně lidský hlas přenášejí v digitálním tvaru. Díky tomu je pro ně relativně snadné přenášet místo hlasu i obecná data. Přesto však existují určitá omezení, týkající se hlavně dosažitelné přenosové rychlosti. V předchozích kapitolách bylo uvedeno, že z principu fungování sítě GSM je pro každý časový interval (hovorový kanál) k dispozici právě 22,8 kbit/s, přestože samotný rádiový kanál nabízí 33,8 kbit/s. Zbylých 11 kbit/s je však použito jako režijní přenosová kapacita, zajišťující funkci sítě GSM.Těchto 22,8 kbit/s však nelze využít výhradně k přenosu samotných uživatelských dat (stejně jako u přenosu hovoru je původní bitová rychlost 13 kbit/s doplněna o další potřebné údaje a vzroste na 22,8 kbit/s), protože i zde je potřeba přidat určitý počet režijních bitů. První cestou, kterou bylo data možno přenášet, bylo použití velmi podobných technik a mechanizmů pro zajištění spolehlivosti datových provozů jako u hlasových přenosů. Datové přenosy tak bylo možno realizovat rychlostí 9,6 kbit/s a zbývajících 13,2 kbit/s bylo využíto na zajištění spolehlivosti přenosu, ošetření chyb a výpadků. Tento princip přenosu dat je stále velmi často používán a je označován jako **CSD** (Circuit Switched Data), což znamená, že se jedná o přenos dat na principu přepojování okruhů.



Po určité době bylo úspěšně odzkoušeno, že datové přenosy nepotřebují tak vysoce robustní ochranné mechanismy a je možné tyto procedury oslabit a naopak zvýšit rychlost přenášených uživatelských dat. Konkrétně se rychlost přenášených dat zvýšila na hodnotu 14,4 kbit/s, ovšem za cenu toho, že datové přenosy touto rychlostí vyžadují kvalitnější signál. V okrajových částech buňky a stejně tak v místech se slabým signálem se pak efektivní využitelnost těchto datových přenosů zmenšuje.



Při každém přenosu dat je nutností schopnost vyrovnat se s případnými chybami při přenosu – nejinak je tomu u přenosů dat sítí GSM. Při klasickém přenosu hlasu je požito kódování, které z původních 13 kbit/s vytvoří 22,8 kbit/s a přijímající strana je pak schopna provést opravu některých přijatých dat. Podobně je tomu u přenosu obecných dat. Použité mechanizmy se sice od přenosu hlasu liší, ale zajišťují také dostatečné zabezpečení proti chybám. Tyto mechanizmy, stejně jako u přenosu hlasu, přidávají k přenášeným datům určité doplňující informace, které umožňují opravu přijímaných dat bez nutnosti přenášet tuto poškozenou část znovu. Toto řešení sice nezpůsobuje nepravidelnosti toku dat a zároveň nezpomaluje tok dat, na druhou stranu je náročný na přenosovou kapacitu. Čím více chyb má protokol odstranit, tím více ochranných bitů je třeba k datům přidávat.

Další zvýšení spolehlivosti datových přenosů touto cestou přidávání informací k přenášeným již není vhodné. Místo toho se používá řešení, využívající zpětnou vazbu mezi odesílatelem a příjemcem. Pokud příjemce přijme dta poškozená, pošle odesílateli zprávu a ten na tuto žádost data opětovně pošle. Tento proces samozřejmě vyžaduje, aby byly obě strany domluveny na tomto postupu a je tedy nutno definovat protokol. Tento protokol se nazývá **RLP** (Radio Link Protocol) a je odvozený od linkového protokolu HDLC. Bitový tok je zde rozdělen na rámce po 200 bitech a k nim jsou přidány bity pro detekci chyb a číslování rámců. Na přijímací straně je testováno správné přijetí každého rámce a v případě chybného přenosu je požadováno opětovné poslání. Protokol RLP je implementován v koncových bodech sítě GSM – jednak v samotném mobilním terminálu a na straně druhé v ústředně MSC, resp. na ní navazující jednotku spolupráce s externími sítěmi **IWF**, která zajišťuje propojení s ostatními sítěmi. Režim, kdy je tento protokol používán se nazývá **netransparentní spojení (přenos)**. Jeho výhodou je nízká chybovost, záporem je větší rozptyl přenosové doby. Jestliže se pro přenos dat nepoužívá protokol RLP, jedná se **transparentní spojení**. Existence tohoto režimu je nutná z toho důvodu, že některé mobilní stanice a jiné koncové zařízení nemusí protokol RLP podporovat.

9.3.6.9.2. HSCSD

HSCSD (High Speed Circuit Switched Data) je řešení pro rychlejší přenos dat, které ve stávající síti GSM využívá více časových intervalů. Jedná se o přenos dat s přepojováním okruhů a oproti klasickému přenosu přináší výrazné zrychlení. Na jednom časovém slotu jsou data přenášena buď rychlostí 9,6 kbit/s, ale s pomocí zefektivnění zabezpečovacího mechanismu a omezení prostoru pro opravné bity je možno z 22,8 kbit/s, připadajících na jeden interval, využívat 14,4 kbit/s. Toto je maximum, které lze přenášet po jednom slotu. Další směr, kterým je možno zvýšit rychlost přenášených dat tedy vede přes současné použití více časových slotů. Animace: http://tomas.richtr.cz/mobil/gsm-hs.htm

Jak je vidět z obrázku, komunikace mobilního terminálu a základnové stanice BTS probíhá na více časových slotech současně podle toho, kolik je jich stanici přiděleno. Přidělení slotů je závislé na momentálním počtu dostupných kanálů a dále na schopnosti samotné mobilní stanice. Mezi BTS a základnovou řídicí jednotkou BSC (rozhraní Abis) jsou data přenášena v kanálech s rychlostí 16 kbit/s (v celkové přenosové rychlosti 2 Mbit/s je 32 kanálů rychlých 64 kbit/s, z nichž každý je dále rozdělen na 4 x 16 kbit/s. Těchto 16 kbit/s může být využito k přenosu hovorových dat (13 kbit/s) nebo právě pro přenos dat rychlostí14,4 kbit/s. Ke sloučení do kanálu s přenosovou rychlostí 64 kbit/s dochází v BSC, případně jednotce **TRAU** (Transcoder and Rate Adaptation Unit). Tato jednotka má za úkol převádět kódovaná hovorová data13 kbit/s do standardních 64 kbit/s hovorových kanálů (PCM) a dále upravovat rychlost přenášených dat rovněž na 64 bit/s. Odtud také vychází maximální přenosová rychlost 64 kbit/s, které je možno pomocí této technologie dosáhnout.

Přenos tímto způsobem může být (a ve většině případech tomu tak skutečně je) asymetrický, tzn. ve směru od mobilní stanice k síti jsou přiděleny například tři časové intervaly a ve směru k mobilní stanici je přidělen interval jeden. Tento způsob přidělování je velmi často používán a je vhodný například pro připojení k internetu, kdy data směřují převážně směrem od sítě k uživateli. Standard, který HSCSD definuje, rozděluje možné režimy do 18 tříd podle toho, kolik kanálů je možno v jednotlivých směrech použít. Některé třídy jsou uvedeny v tabulce.

Třída HSCSD	Maximální počet slotů		
	Příjem	Vysílání	Celkem
1	1	1	2
2	2	1	3
3	2	2	3
4	3	1	4
5	2	2	4
6	3	2	4
9	3	2	5
10	4	2	5
12	4	4	5
13	3	3	6
18	8	8	16

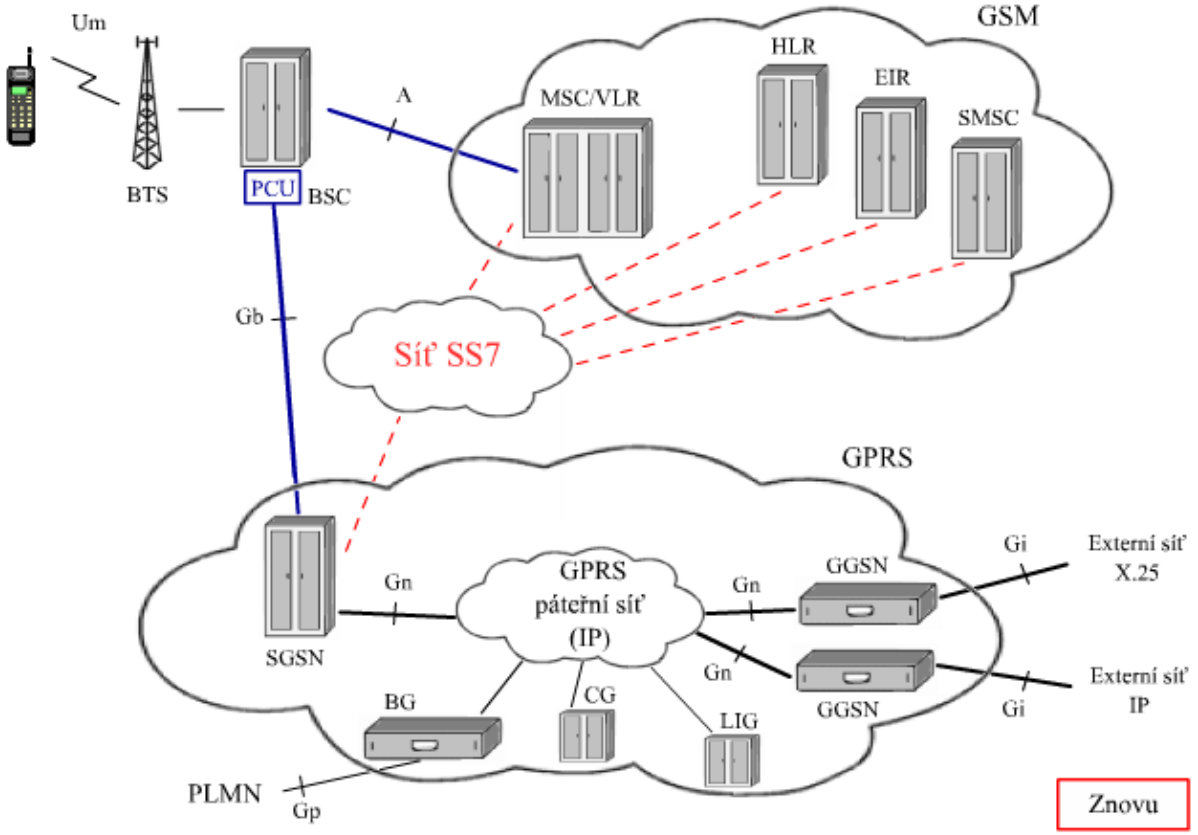
Tento standard je jednou z nejmladších modifikací systémy GSM (únor 1997). V současné době provozuje v České republice službu HSCSD pouze společnost Eurotel pod obchodním jménem HSCSD FAST. Ostatní operátoři tuto technologii nevyužívají a rozhodli se přejít rovnou na rychlé přenosy dat technologií GPRS. V síti Eurotel je možno použít až 4 sloty

současně, jedné se tedy o 6. třídu z výše uvedené tabulky. Na uživateli je, aby rozhodl zda použije přenos symetrický (2 + 2 sloty, neboli 22,8 kbit/s v obou směrech) nebo přenos asymetrický (3+1 slot, odpovídající 43,2 kbit/s na příjmu a 14,4 kbit/s pro vysílání). Z uvedeného vyplývá, že Eurotel používá na jednom kanálu přenos dat rychlostí 14,4 kbit/s.

9.3.6.9.3. GPRS

GPRS (General Packet Radio System) je zatím nejmodernější používaná technologie přenosu dat prostřednictvím sítě GSM. Celý přenos dat touto technologií spočívá na principu přepojování paketů, což přináší do mobilní komunikace řadu novinek. Přenosové kapacity zde nejsou nikomu trvale vyhrazovány, ale jsou k dispozici všem uživatelům zároveň. Takovéto řešení má několik zásadních výhod. První výhodou je maximálně efektivní využití přenosové kapacity těmi uživateli, kteří skutečně potřebují data v daný okamžik přenášet. To je zásadní odlišnost od předchozí varianty s přepojováním okruhů, kdy byla každému uživateli vyčleněna taková přenosová kapacita o kterou požádal, bez ohledu na její skutečné využívání. Další podstatnou výhodou, která vyplývá z principu přepojování paketů, je způsob zpoplatňování. Technologie GPRS umožňuje platbu nikoliv za čas (dobu spojení), ale podle objemu skutečně přenesených dat.

Zavedení technologie GPRS do sytému GSM vyžaduje poměrně velký zásah do existující mobilní sítě, protože ta byla budována pouze pro přenos dat na bázi přepojování okruhů. Na následujícím obrázku je znázorněna cesta od klasické sítě GSM po síť s implementovanou technologií GPRS.



Celý proces zavedení GPRS do sítě GSM si lze v jednoduchosti představit jako "přeložení" další sítě (fungující na principu přepojování paketů) přes stávající pevnou část sítě GSM a provázání těchto dvou sítí s řídicími prvky. Změny v původní síti se týkají jednak programového vybavení, v některých případech i samotného zařízení. Důležitou změnou je rozšíření základnové řídicí jednotky BSC o blok **PCU** (Packet Controller Unit), který je bodem kde se

obě sítě střetávají. Novými prvky v celé architektuře jsou potom především uzly pro podporu GPRS, tzv. **GSN** (GPRS Support Node).

- **SGSN** (Serving GPRS Support Node): Jedná se vlastně o analogii MSC, pracuje však s pakety. Plní především následující funkce:
    - SGSN směruje v obou směrech datové pakety a obsluhuje všechny uživatele přihlášené do GPRS sítě kteří se nacházejí v jeho dosahu. Ve směru k uživateli je provoz směrován ze SGSN přes BSC na BTS a odtud do mobilní stanice.
    - Provádí autentizaci, šifrování a kontrolu IMEI (v klasické síti probíhá šifrování mezi telefonem a BTS)
    - Funkce mobility managementu (což je součást 3. vrstvy vrstvového modelu GSM, plnící funkce autentifikace, lokalizace atd.).
    - Řízení logických kanálů směrem k mobilní stanici.
    - Spojení s HLR, MSC.
- **GGSN** (Gateway GPRS Support Node): Poskytuje rozhraní mezi GPRS sítí a externími sítěmi pracující podle standardu IP nebo X.25, je tedy jakousi branou mezi GSM a vnějšími sítěmi. V GGSN jsou také obsaženy funkce tzv. mobility managementu. Dalšími funkcemi GGSN je např. směrování paketů na správný uzel SGSN, konverze protokolů mezi GPRS a vnějšími sítěmi a může také pracovat jako firewall, tzn. ochrana sítě před nežádoucím vniknutím zvenku. Mezi sebou pak uzly SGSN a GGSN komunikují prostřednictvím protokolu GTP (GPRS Tunelling Protocol), který lze považovat za aplikační protokol rodiny protokolů TCP/IP - sám využívá ke svému fungování transportní protokoly UDP nebo TCP, pod kterými je provozován protokol IP.

Dalšími jednotkami " překryvné" sítě GPRS jsou :

- **BG** (Border Gateway): Jedná se o bránu mezi GPRS sítěmi různých operátorů, umožňující funkci roamingu.
- **CG** (Charging Gateway): Jednotka, která uchovává a vyhodnocuje zúčtovací data, která přicházejí od SGSN a GGSN. Po příslušném zpracování jsou tato data odeslána do celkového zúčtovacího systému sítě GSM.
- **LIG** (Legal Interception Gateway): Jednotka odposlechu, která v některých zemích musí být dle zákona nainstalována.

Přenosové rychlosti jsou opět nejvíce limitovány rádiovým rozhraním GSM, kde na každý slot připadá přenosová rychlost 33,8 kbit/s. Když se do ní započítá režie připadající na fungování sítě, zbývá přenosová rychlost 22,8 kbit/s. Z této rychlosti se pak u standardního přenosu odečítá ještě dalších 13,2 kbit/s připadajících na zajištění samotného přenosu, zejména jeho spolehlivosti (u klasických a HSCSD přenosů rychlostí 14,4 kbit/s je tato režie jen 8,4 kbit/s). I v případě GPRS je možné vyjít opět jen z oněch 22,8 kbit/s a snažit se minimalizovat počet dalších režijních bitů připadajících na zajištění přenosů. Zde velmi záleží na konkrétních podmínkách šíření signálu - při optimálních podmínkách je možné výrazněji oslabit režii na zajištění přenosu, a při horších podmínkách šíření je naopak nutné ji zase zvětšit. GPRS proto zavádí čtyři různé třídy, které označuje jako kódovací schémata (coding scheme), jejichž přehled je uveden v následující tabulce.

Třída	"Užitečná" přenosová rychlost
CS-1	9,05 kbit/s
CS-2	13,4 kbit/s
CS-3	15,6 kbit/s
CS-4	21,4 kbit/s

Pokud by GPRS mohlo využít maximální počet 8 slotů současně, pak by to při optimálních podmínkách šíření signálu (kdy lze použít CS-4) odpovídalo rychlosti 8 x 21,4, neboli 171,2 kbit/s. Je ale třeba zdůraznit, že se jedná skutečně spíše jen o teoretické maximum, protože jde o souběh dvou optimálních případů – optimálních podmínek šíření signálu a možnosti alokovat všech 8 slotů současně. V praxi bude takovýto souběh nastávat jen velmi vzácně, navíc lze počítat s tím že operátoři alespoň z počátku budou sami omezovat maximální dosažitelnou rychlost, zejména kvůli počtu obsazených slotů, takže skutečná nabídka bude s největší pravděpodobností výrazně nižší.
Pro GPRS byly definovány tyto třídy terminálů:

- **A** – terminály této třídy jsou schopny současně ovládat služby na základě spojování okruhů i spojování paketů. Oba druhy služeb jsou ovládány nezávisle na sobě.
- **B – tyto** terminály ovládají v jeden okamžik jen jednu z těchto služeb, ale je možné automaticky přecházet mezi těmito dvěma režimy. Může přerušit přenos paketů při příchozím hovoru a pokračovat později.
- **C–u** těchto terminálů je nutné nastavit režim (paketový nebo okruhový). Při nastavení na spojování okruhů nemůžete využívat GPRS a naopak.

Dále GPRS vychází vstříc skutečnosti, že různé druhy datových přenosů mohou mít různé požadavky na jejich kvalitu, resp. kvalitativní parametry. GPRS nabízí různé úrovně kvality služeb **QoS** (Quality of Service) pokud jde o:

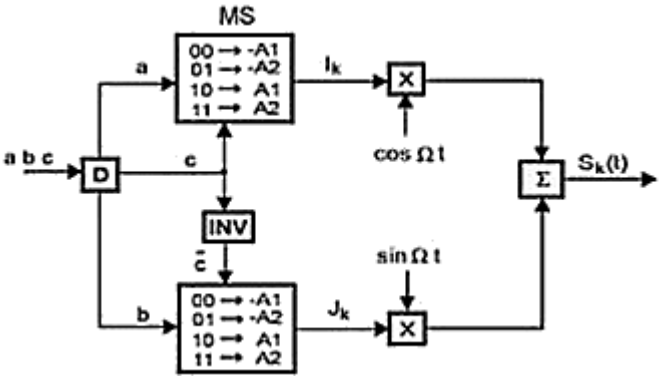
- **Prioritu** – zde jsou definovány tři úrovně priority: vysoká, střední a nízká.
- **Spolehlivost** – zde jsou definovány opět tři varianty, resp. třídy spolehlivosti, které definují určité kombinace pravděpodobnosti toho, že dojde ke ztrátě paketu, k přijetí duplikátu, k poškození paketu či jeho doručení mimo pořadí.
- **Zpoždění – definovány** jsou 4 třídy vztažené k průměrnému zpoždění a ke zpoždění 95 % přenášených paketů.
- **Propustnost** – zde je definována maximální (špičková) a střední přenosová rychlost.

GPRS tedy přináší uživatelům mnoho nových možností a nabízí:

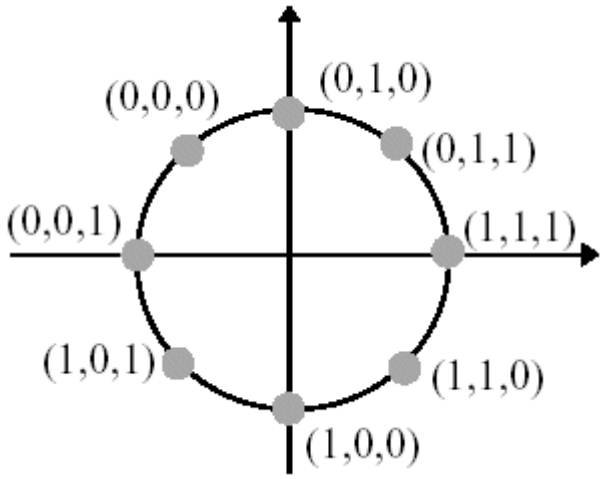
- Levnější přístup do datových sítí
- Vyšší přenosové rychlosti
- Kratší doba sestavení spojení
- Nepřetržité spojení s vnější datovou sítí
- Lepší podpora při roamingu
- Lepší podpora přístupu do různých datových sítích Internet, podnikové sítě atd.

9.3.6.9.4. EDGE

Nová technologie, nesoucí zkratku **EDGE** (Enhanced Data Rates for GSM Evolution) zvýší kapacitu sítě a datové rychlosti a týká se obou druhů nově zaváděných technologií, založených buď na technice přepojování okruhů – HSCSD, nebo přepojování paketů – GPRS. Cesta k větší efektivnosti přes dokonalejší kódování a sdružování hovorových kanálů (slotů) je již vyčerpána, a tak při snaze zachovat přidělená frekvenční pásma, jejich rozdělení na frekvenční kanály o šířce 200 kHz a jejich další členění pomocí techniky časového multiplexu (TDMA) zbývá již jen poslední možnost – použít jinou techniku modulace. Způsob, jakým EDGE dosahuje zvýšených rychlostí vychází z použití jiného (dokonalejšího) způsobu modulace, a to osmistavové fázové modulace **8 PSK** (Phase Shift Keying) místo původní modulace GMSK (Gaussian minimum Shift Keying).



Na vstup modulátoru přicházejí bloky bitů (v tomto případě tzv. tribity), které se podle schématu rozdělí do dvou větví a potom putují do měničů signálu. Zde je každé dvojici bitů přiřazena napěťová úroveň +- Ax. V tomto případě jsou v každé větvi 4 úrovně, protože kombinujeme vždy dva bity. Bity a a b určují polaritu úrovní I_k a J_k a bit c velikost úrovně. Výstupem z měničů signálu jsou pak napětí o amplitudách I_k, J_k, která jsou následně přiřazena nosným vlnám cos Ωt a sin Ωt. V další části dochází k vektorovému součtu těchto nosných a výsledkem jsou fázově posunuté hodnoty, jak je vidět na následujícím obrázku.



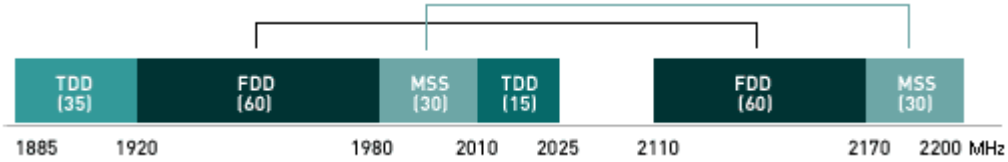
Symbolová rychlost na jednom frekvenčním pásmu 200 kHz zůstává stejná jako dříve (270,833 ksymbolů/s), protože však používáme osmistavovou modulaci (oproti dvoustavové GMSK), je nyní bitová rychlost trojnásobná. Maximální uváděná využitelná přenosová rychlost po odečtení bitů zajišťujících datové přenosy je potom 384 kbit/s. To však platí při současném využití všech 8 slotů a při dobrých podmínkách pro šíření signálu (nepříliš velké vzdálenosti od základnové stanice).

Nasazení této technologie bude vyžadovat především přidání nového transcieveru do každé buňky (BTS), další úpravy by měly týkat především programového vybavení. Velmi výhodná se jeví kombinace technologie EDGE spolu s technologiemi GPRS a HSCDS, kde stávající princip přenosu rozšířený o nový způsob modulace přináší výrazné zrychlení datových přenosů. Takovýmto společným použitím vznikají principy nazvané ECSD (Enhanced Circuit Switched Data) a EGPRS (Enhanced General Packet Radio System). EDGE je obecně považován za poslední krok "vylepšování" systému GSM před zaváděním sítí 3. generace.

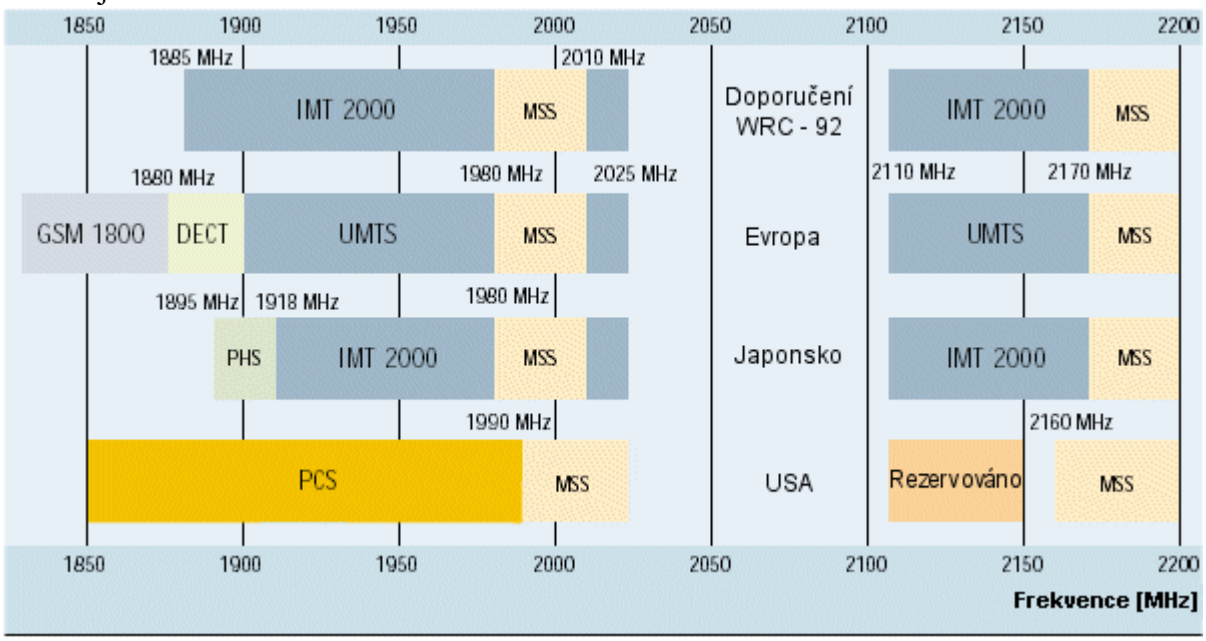
9.3.6.10. IMT-2000

Od počátku devadesátých let intenzivně pracují mezinárodní telekomunikační orgány na vývoji systémů třetí generace, určených pro personální komunikaci. První iniciativy **ITU** (International Telecommunication Union) se objevili již v roce 1986, tedy ještě před dokončením standardizace systémů 2. generace. Pracemi na systému zvaném **FPLMTS** (Future Public

Land Mobile Telecommunication System – budoucí systém pro pozemní mobilní telekomunikaci) byl v rámci ITU pověřen **CCIR** (International Radio Consultative Committee). Dosti důležitým milníkem v této oblasti byl rok 1992. V dubnu tohoto roku **WARC** (World Administrative Radio Conference – Světový administrativní kongres pro rádio, později zkráceno na WRC) vymezil pro FPLMTS kmitočtové pásmo 230 MHz (1885-2025 MHz a 2110-2200 MHz). Část tohoto pásma (1980-2010 MHz a 2170-2200 MHz) je určena pro družicovou část systému **MSS** (Mobile Satellite Service – Mobilní družicová služba).



Ačkoliv je toto frekvenční spektrum doporučeno pro IMT-2000, v jednotlivých regionech světa jsou již některé frekvence používány současnými systémy. Situace je zobrazena na následujícím obrázku.



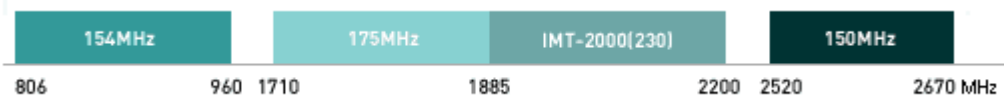
V Evropě je situace nejjednodušší, dochází pouze ke kolizi se systémem DECT (1880-1900 MHz), který však lze považovat za součást systému UMTS, což je evropský model systému IMT-2000. V Japonsku je situace podobná, největší problém nastává v USA. Současný systém PCS 1900 zde kompletně překrývá doporučené spektrum, řešením tedy může být použití technologie, umožňující koexistenci obou těchto systémů.

V roce 1992 byla také dohodnuta nová struktura ITU. Původní poradní sbory CCIR a CCITT (International Telphony and Telegraphy Consultative Committee) byly pozměněny na **ITU-R** (přebírá část činnosti CCIR) a **ITU-T** (přebírá část činnosti CCIR a činnost CCITT). Došlo také k přejmenování projektu FPLMTS na **IMT-2000** (International Mobile Telecommunications - 2000), tedy na Mezinárodní mobilní telekomunikaci - 2000. Číslovka 2000 se zde vyskytuje z několika důvodů.

- Doporučení mají být k dispozici kolem roku 2000
- Systém bude pracovat v oblasti kmitočtů kolem 2000 MHz
- Maximální přenosová rychlost systému bude nabývat hodnot kolem 2000 kbit/s.

Později, v roce 2000, byly k již uvedeným frekvenčním pásmům pro IMT-2000 schváleny ještě další. Stalo se tak na konferenci WRC-2000 v Istanbulu. Jedná se o nová pásma 1710-1885 MHz a 2520-2670 MHz. Dále bylo schválen záměr použít pro IMT-2000 také frekvence, které jsou již dnes vyhrazeny pro mobilní komunikace – jde o frekvence v pásmu 806-

960 MHz. Tyto nové frekvence jsou rezervovány z důvodu předpokladu velkého zájmu o mobilní komunikaci v budoucnu a mohou být použity globálně nebo pouze v jednotlivých regionech.

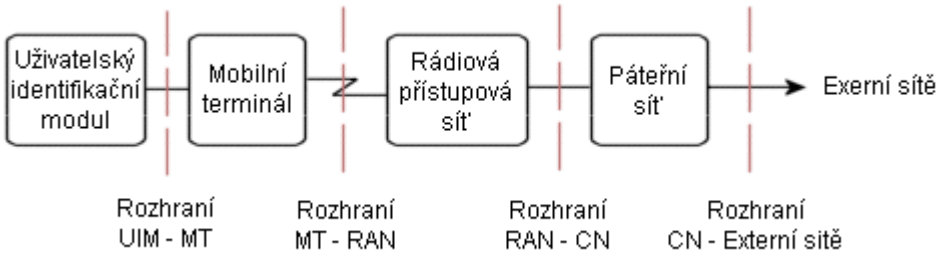


Zde jsou uvedeny některé požadavky na systém IMT-2000 :

- Zvýšení přenosových rychlostí současných systémů 2. generace na hodnoty až 2 Mbit/s
  - Pro uživatele s vysokým stupněm mobility minimálně 144 kbit/s, snaha je však o 384 kbit/s. Tato přenosová rychlost by měla být dostupná při velkém pokrytí území.
  - Pro uživatele s malým stupněm mobility 2 Mbit/s. Tato rychlost má být dostupná pouze lokálně, tzn. v obytných čtvrtích a centrech měst.
- Celosvětová kompatibilita jak na uživatelské úrovni (mezinárodní roaming), tak na úrovni operátorů (spolupráce zařízení různých výrobců). Pro zajištění tohoto požadavku je třeba definovat v celém systému několik rozhraní, viz. obrázek dále.
- Lepší využití rádiových frekvencí (vyšší spektrální účinnost modulací, účinnější zdrojové a kanálové kódování)
- Malé a levné terminály pro různé typy aplikací
- Identifikace účastníka nezávisle na mobilním terminálu pomocí **UIM** (User Identification Module – Uživatelský identifikační modul), což je obdoba SIM karty u systému GSM
- Široké spektrum služeb s různým stupněm kvality

Systém je pro účely návrhu obecně rozdělen na dvě hlavní části:

- Rádiová přístupová síť **RAN** (Radio Access Network), která zabezpečuje přístup mobilního uživatele k páteřní síti pomocí rádiového prostředí. V porovnání se systémem GSM se jedná o obdobu subsystému základnových stanic.
- Páteřní síť **CN** (Core Network) nebo také jádro sítě, které provádí spojovací funkce (propojení účastníků, směrování paketů), udržuje a aktualizuje důležité uživatelské informace (poloha, bezpečnost, účtování) a zajišťuje spojení do dalších sítí (ISDN, X.25, PSTN, Internet, ...). Obdobou v síti GSM je síťový spojovací subsystém NSS.



Uživatel je jednoznačně identifikován vložením UIM do mobilního terminálu. Pro přenositelnost UIM na různé typy mobilních terminálů tedy musí být definované rozhraní UIM-MT. Mobilní terminál poté komunikuje přes rádiové rozhraní MT-RAN s rádiovou přístupovou sítí. Definice tohoto rozhraní je důležitá především z důvodu mezinárodního roamingu. Dále je nutné definovat rozhraní RAN-CN, a to z důvodu umožnění spolupráce zařízení různých výrobců.

Na vývoji systému splňujícího podmínky IMT-2000 se pracuje v několika regionech světa a do roku 1998 bylo předloženo 10 návrhů pro pozemní mobilní komunikaci.

Evropa (organizace ETSI) vyvíjí systém **UMTS** (Universal Mobile Telecommunication System – Univerzální systém pro mobilní telekomunikaci) a podány byly následující návrhy:

- **UTRA** (UMTS Terrestrial Radio Access) definuje pro duplex FDD metodu **WB-CDMA**, tj. širokopásmové asynchronní DS-CDMA a pro TDD **TD-CDMA**, tj. širokopásmové TDMA s rozprostřením.
- **DECT**

Japonsko, tedy organizace ARIB (Association of Radio Industries and Business) podalo návrh:

- **W-CDMA** se stejnými parametry jako UTRA (dva režimy – FDD a TDD)

USA předložili tyto návrhy:

- **cdma2000** – jedná se o širokopásmovou variantu systému IS-95 (cdmaOne)
- **UWC-136** (Universla Wireless Communication) - podporuje tři různé šířky pásma: 30 kHz, tj. vylepšený systém IS-136, 200 kHz, a 1,6 MHz (širokopásmové TDMA) pro použití uvnitř budov
- **WIMS W-CDMA** (Wireless Multimedia and Messaging Services Wideband CDMA) - bezdrátové služby přenosu multimédií a zpráv určené pro bezdrátové účastnické smyčky WLL
- **NA: W-CDMA** (North American WB-CDMA)

Jižní Korea, zastoupená organizací TTA (Telecommunications Technology Association) s návrhy:

- **CDMA I** – jedná se o synchronní DS-CDMA (obdoba cdma2000 z USA)
- **CDMA II** – asynchronní DS-CDMA (obdoba Evropsko/Japonského W-CDMA)

Čína, tedy oranizace **CATT** (China Academy of Telecommunication and Technology) podal návrh:

- **TD-SCDMA** (Time Division – Synchronous CDMA) - čínská varianta širokopásmové CDMA.

Na jednání ITU v roce 1999 pak došlo ke schválení kompletní sady specifikací rádiových rozhraní pro IMT-2000, z původních deseti návrhů byly vybrány tyto:

- **IMT-DS** (Direct Spread) - UTRA FDD (W-CDMA) - Evropsko/Japonský návrh
- **IMT-MC** (Multi Carrier) - cdma2000 – USA
- **IMT-TC** (Time Code) - UTRA TDD (TD-CDMA) - Evropsko/Japonský návrh
- **IMT-SC** (Single Carrier) - UWC-136 - USA
- **IMT-FT** (Frequency Time) - DECT – Evropa

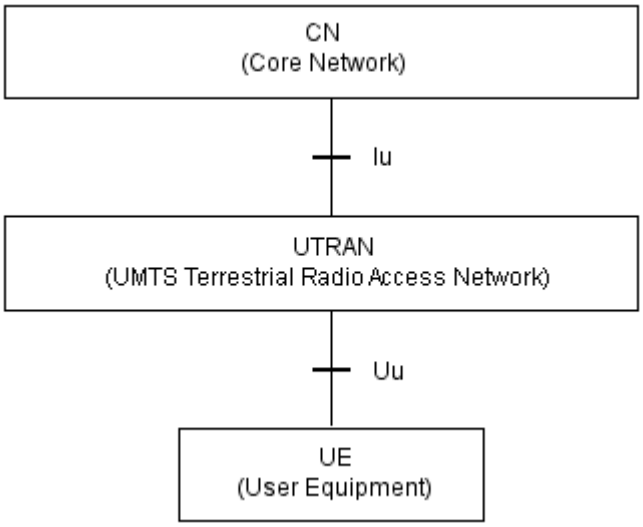
9.3.6.11. Základní struktura UMTS

Sítě třetí generace budou implementovány do existujících sítí druhé generace. Kromě Japonska by nikde jinde na světě neměla existovat pouze síť 3G bez předchozí funkční sítě druhé generace. Koncept využití mobilních sítí třetí generace předpokládá alespoň zpočátku úplné pokrytí území systémem GSM a malé oblasti pokryté systémem UMTS. Pro hlasové služby se tedy alespoň v blízké budoucnosti bude stále používat systém GSM. Pro sítě třetí generace byla zvolena technologie **CDMA** (Code Division Multiple Access), což je přístupová metoda kódového dělení. Pro UMTS je použita její varianta **WB-CDMA** (Wideband Code Division Multiple Access), širokopásmová přístupová metoda. V CDMA neexistuje žádné časové dělení a všichni uživatelé používají přidělené frekvenční pásmo po celou dobu. K rozeznání různých uživatelů, kteří používají jedno frekvenční pásmo současně, se používá uživateli přidělený binární kód. Spektrum se skládá z jednoho párového pásma (1920-1980 MHz + 2110-2170 MHz) a jednoho nepárového pásma (1910-1920 MHz + 2010-2025 MHz). Duplexní metody pro WB-CDMA jsou dvě:

- FDD (Frequency Division Duplex) pro párové pásmo;
- TDD (Time Division Duplex) pro pásmo nepárové.

Systémy druhé generace jsou primárně zaměřeny na přenos hlasu, zatímco UMTS je systém přinášející především vysokorychlostní data. Původní myšlenkou UMTS bylo nabídnout tuto službu tam, kde se koncentrují potenciální UMTS uživatelé, tedy ve velkých městech a v hustě obydlených oblastech. Například ve Švédsku, které je celoplošně pokryto sítí GSM/GPRS, se staví malé ostrůvky UMTS pokrývající pouze největší švédská města. S malými modifikacemi by takový model měl, alespoň zpočátku po zavedení UMTS, fungovat v celé Evropě, tedy i v ČR. Předpokladem pro tento přechod je implementovaná technologie GPRS v síti GSM. Zatím se tedy nepočítá s výstavbou žádné čistě samotné UMTS sítě (mimo Japonska), i když taková možnost samozřejmě v praxi existuje.

Na následujícím obrázku je velmi jednoduše zobrazena struktura systému UMTS.



Na nejvyšší úrovni bude použita ATM páteřní síť **CN** (Core Network), dále pak směrem k uživatelům radiová přístupová síť **UTRAN** (UMTS Terrestrial Radio Access Network), a konečně uživatelé budou přistupovat k UMTS síti pomocí uživatelských terminálů **UE** (User Equipment). Mezi těmito entitami byly definovány rozhraní **Iu** (mezi CN a UTRAN) a **Uu** (mezi UTRAN a UE). V porovnání se systémem GSM odpovídá přístupové síti UTRAN subsystém základnových stanic BSS a páteřní síti (jádru) CN odpovídá síťový spojovací subsystém NSS.

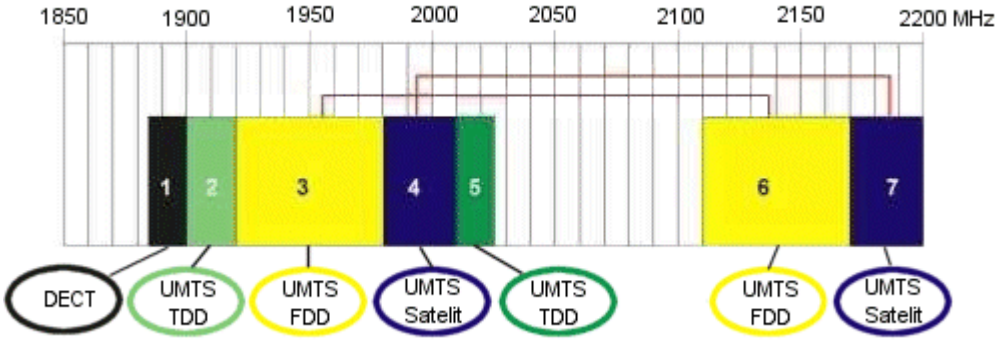
Hlavní funkcí CN je spojování hovorů a směrování paketů. Existuje několik možných provedení CN, jednotným požadavkem však je dostatečná přenosová kapacita. Součástí této páteřní sítě jsou také databázové funkce a funkce síťového managementu. Přístupová síť UTRAN je částí systému, se kterým prostřednictvím rádiového rozhraní komunikují jednotlivé uživatelské terminály. Samozřejmě ani v systému UMTS není možná komunikace jednotlivým mobilních stanic přímo mezi sebou. Uživatelský terminál UE nepatří do UTRAN ani do žádné jiné části systému, ale jedná se o samostatnou součást systému UMTS.

9.3.6.12. Rádiové rozhraní – alokace spektra

Jak již bylo popsáno dříve, pro sítě 3. generace byla zvolena technologie CDMA (Code Division Multiple Access). Pro přenos využívají sítě UMTS dvou duplexních technik neboli přenosové módy FDD a TDD.

9.3.6.12.1. Alokace spektra UMTS

Spektrum se skládá z jednoho párového pásma (1920-1980 MHz + 2110-2170 MHz) a jednoho nepárového pásma (1910-1920 MHz + 2010-2025 MHz).



Pro nepárová pásma (časově dělený duplex TDD) byla pro UMTS vybrána technologie TD-CDMA, která je vhodná pro asymetrické vysokorychlostní datové přenosy, pokrytí hlavně uvnitř budov. Pro párová pásma (s frekvenčním duplexem FDD) byla vybrána pro UMTS technologie W-CDMA, která je zase vhodnější pro velkoplošné pokrytí a symetrické středně rychlé datové služby, ale je mnohem náročnější na regulaci výkonu na straně mobilního telefonu i základnové stanice Node B. Tyto dvě technologie dohromady tvoří společné rádiové rozhraní označované zkratkou **UTRA** (UMTS Terrestrial Radio Access)

Číslo na obrázku	Frekvenční rozsah	Šířka pásma	Použitá technologie
pásmo č. 1	1885 - 1900 MHz	15 MHz	DECT
pásmo č. 2	1900 - 1920 MHz	20 MHz	TD-CDMA
pásmo č. 3	1920 - 1980 MHz	60 MHz	W-CDMA - Uplink
pásmo č. 4	1980 - 2010 MHz	30 MHz	Satelitní složka
pásmo č. 5	2010 - 2025 MHz	15 MHz	TD-CDMA
pásmo č.6	2110 - 2170 MHz	60 MHz	W-CDMA - Down-link
pásmo č. 7	2170 - 2200 MHz	30 MHz	Satelitní složka

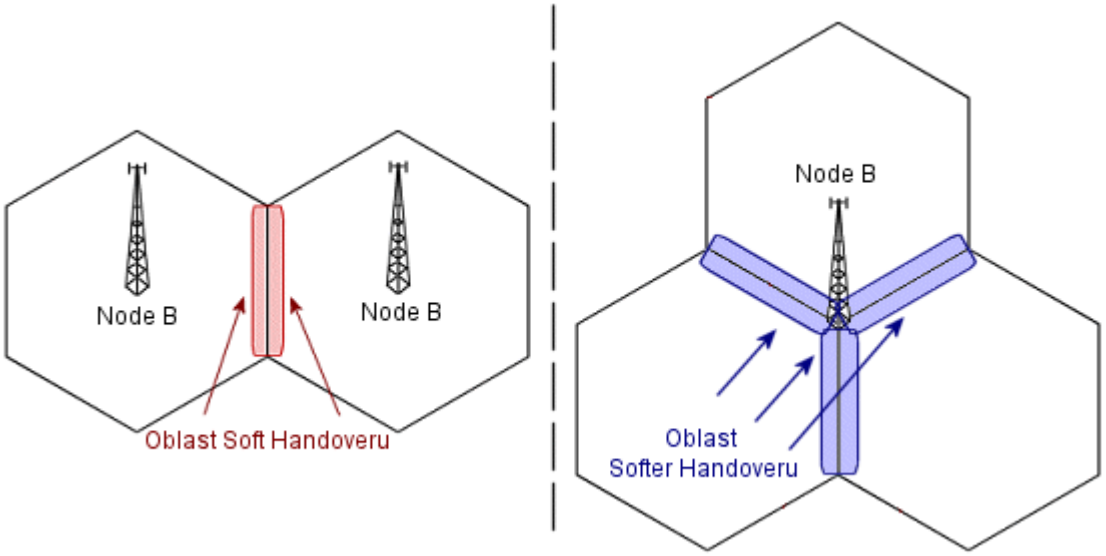
9.3.6.13. Handover v systému UMTS

V systému UMTS je stejně jako u jiných celulárních telekomunikačních systémů účastníkům umožněn volný pohyb. Pro splnění této vlastnosti musí síť v průběhu právě probíhajícího spojení umožňovat přechod mobilní stanice z jedné buňky do druhé, tedy handover. Handover mezi buňkami, jež používají stejnou rádiovou frekvenci, se nazývá **Intra-frequency handover**, handover mezi buňkami pracujícími na jiných frekvencích **Inter-frequency handover.** Systém UMTS využívá několika typů handoveru:

- **Hard handover**: Během tohoto handoveru je nutností změnit vysílací a přijímací kanál (frekvenci) a v určitém čase je tedy možná komunikace pouze s jednou základnovou stanicí. Tento handover je používán vždy pro Inter-frequency handover a využívá jej např. systém GSM nebo přístupová metoda UTRA-TDD.
- **Soft handover**: Tento handover probíhá v UMTS mezi dvěma různými Node B. V průběhu tohoto handoveru komunikuje mobilní stanice se dvěma Node B a nedchází tak k žádnému výpadku hovoru. Mobilní stanice průběžně vyhledává nové Node B a na stávající frekvenci a monitoruje výkon přijímaného signálu. Tyto informace jsou zasílány zpět do sítě a na jejich základě RNC mobilní stanici nařizuje přidat nebo naopak ubrat některé Node B ze své aktivní sady. Aktivní sada tedy obsahuje všechny nejbližší Node B, vhodné pro handover. V případě vysílání mobilní stanice vzniká výsledná podoba signálu až v RNC, v případě vysílání Node B vysílají různé základnové

stanice stejná data. Tento typ handoveru přináší vyšší kvalitu signálu, z důvodu současné komunikace s několika Node B však více zatěžuje rádiový sektor systému. Tento handover je využit v přístupové metodě UTRA-FDD.

- **Softer handover**: Jedná se o speciální případ soft handoveru. Mobilní stanice zde přechází mezi jednotlivými sektory patřící jedné základnové stanici Node B. Tento handover je řešen podobně jako soft handover, hlavní rozdíly existují zejména na úrovni implementace uvnitř sítě a je zde využíváno technologie zvané Rake Reciever



- **Mezisystémový handover mezi systémem UMTS a GSM**: V kombinovaných sítích UMTS a GSM může docházet k handoveru mezi Node B a základnovou stanicí BTS. Nutností je samozřejmě mobilní stanice spolupracující s oběma systémy. Tento handover pak probíhá vždy jako hard handover.

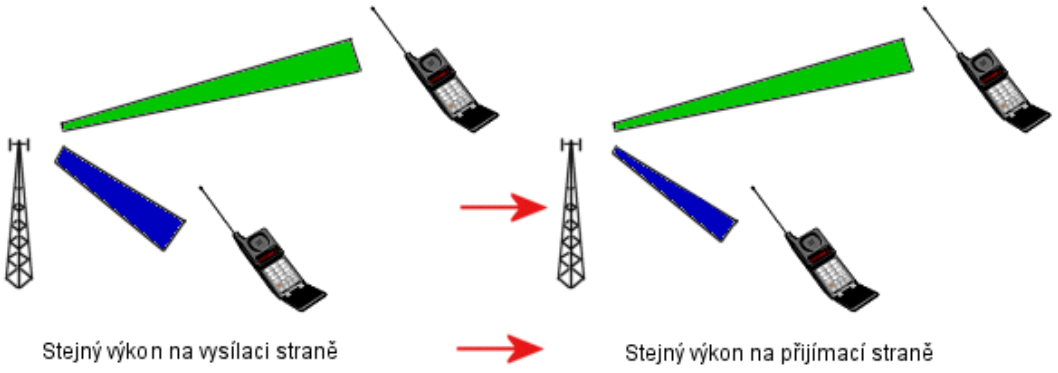
9.3.6.14. Řízení výkonu v UMTS

Velmi důležitou činností uvnitř systému UMTS je řízení vysílacího výkonu (Power Control) mobilních stanic a základnové stanice Node B. Z důvodů možného vzájemného rušení jednotlivých mobilních účastníků hraje řízení výkonu dosti důležitou roli. Vysílací výkon může být regulován ve dvou směrech (uplink – vysílá mobilní stanice, downlink - vysílá Node B) a existují dva zkladní principy řízení výkonu:

- Zpětná uzavřená smyčka (Closed Loop Power Control)
- Zpětná otevřená smyčka (Open Loop Power Control)

9.3.6.14.1. Řízení výkonu – Uplink

Jeden z důvodů, proč je kontrola výkonu tak důležitá je efekt nazývaný **near-far problem** (problém blízké a vzdálené stanice). Popsán je následujícím obrázkem.



Mobilní stanice používající CDMA pracují na stejné frekvenci, a proto se přímo mezi nimi může vyskytovat rušení. Kdyby dvě stanice, různě vzdálené od základnové stanice, vysílaly stejným vysílacím výkonem, uživatelé blíže základnové stanici by nad vzdálenějšími dominovali a signál vzdálených uživatelů by byl velmi silně rušen. Tento problém je nutno vyřešit právě vhodným řízením vysílacího výkonu jednotlivým mobilních stanic. Dalším přínosem je potom menší spotřeba energie mobilní stanicí a zvýšení kapacity sítě. Řízení výkonu pro uplink je prováděno následujícími technikami:

- **Uplink Closed Loop Power Control**: Základnová stanice měří SIR (Signal to Interference Ratio) přijímaného signálu od všech mobilních stanic a porovnává jej s cílovou hodnotou SIR_{target}. Po tomto porovnání UTRAN vysílá mobilní stanici pokyny ke zvýšení nebo naopak snížení vysílacího výkonu.
- **Outer Loop Power Control**: Při této metodě RNC určuje jednotlivým Node B cílovou hodnotu $SIR_{target.}$ RNC tuto hodnotu určuje v závislosti na požadované QoS.
- **Uplink Open Loop Power Control**: Mobilní stanice nastavuje svůj vysílací výkon podle úrovně přijatého pilotního signálu od základnové stanice.

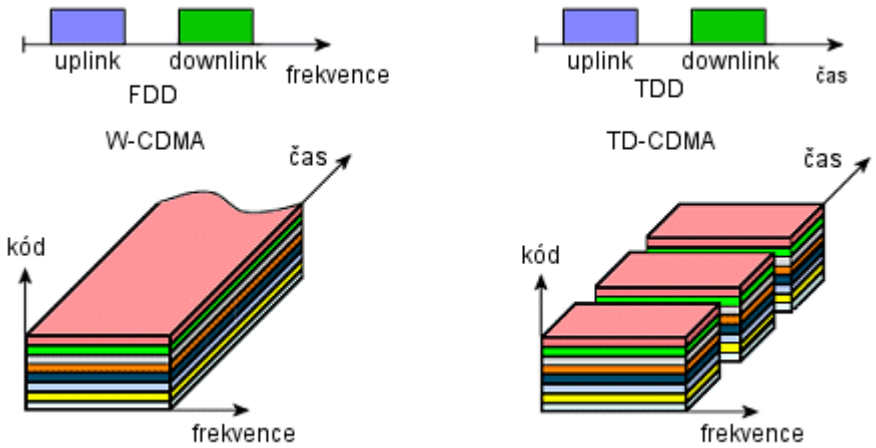
9.3.6.14.2. Řízení výkonu – Downlink

- **Downlink Closed Loop Power Control**: Řízení výkonu při vysílání Node B probíhá pouze touto jedinou metodou. Jedná se o stejný princip jako v opačném směru, vysílací výkon pro Node B nyní určuje mobilní stanice.

9.3.6.15. FDD

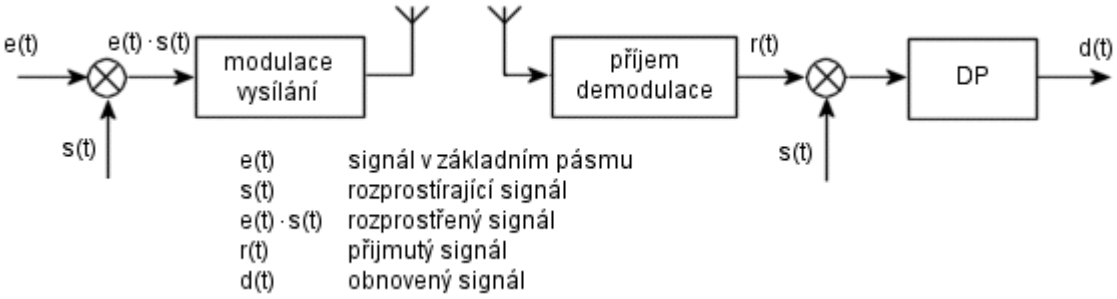
9.3.6.15.1. DS-CDMA

UTRA (UMTS Terrestrial Radio Access – UMTS pozemský rádiový přístup) pro duplexní metodu FDD využívá jako metodu mnohonásobného přístupu širokopásmovou **W-CDMA** (Wideband CDMA) s rozprostíráním přímou posloupnosti **DS-CDMA** (Direct Sequence CDMA). Návrh je harmonizován mezi evropským ETSI a japonským ARIB – hlavní parametry na sestupné trase pocházejí z návrhu ARIB, parametry vzestupné trasy vzešly z návrhu ETSI.

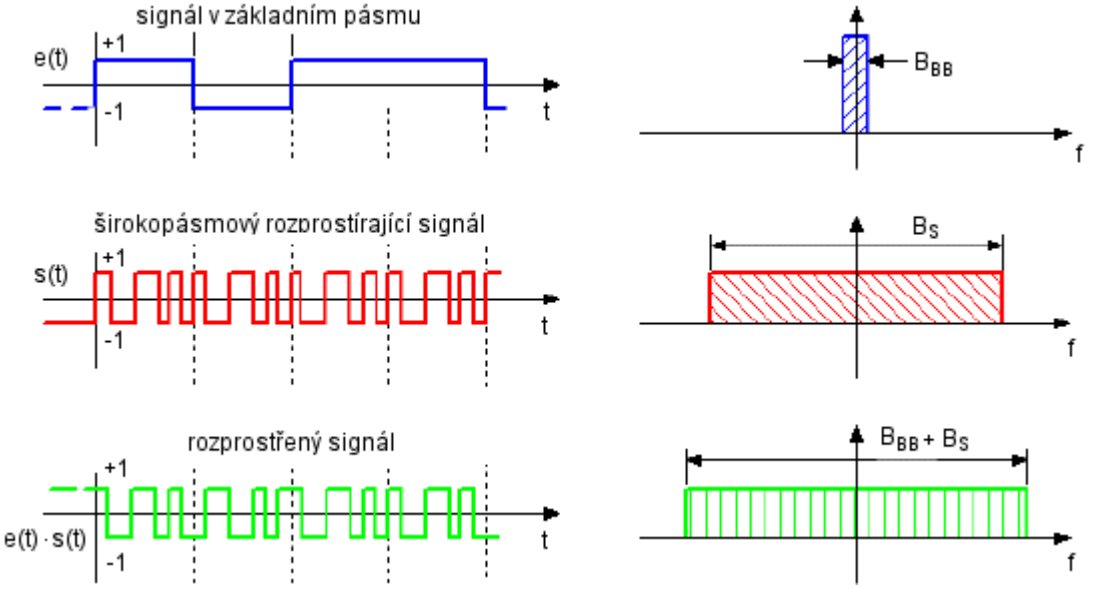


Pomocí následujících obrázků je stručně nastíněn princip fungování techniky DS-CDMA. Datový signál v základním pásmu je pomocí kombinace s rozprostírajícím signálem rozprostřen do širšího frekvenčního pásma. Tento rozprostírající signál má mnohem vyšší bitovou rychlost než signál kódovaný. U rozprostírajícího signálu se však nehovoří o bitech ale o čipech (chip) a o čipové rychlosti (chip rate). Poměr čipové rychlosti rozprostírajícího kódu a bitové rychlosti kódovaného signálu se nazývá **rozprostírající faktor** (spreading factor). Rozprostřený signál je dále modulován a vysílán. Na straně přijímače je přijat rozprostřený

signál a pomocí stejného (známého) kódu, jakým byl signál rozprostřen je opět obnoven. Důležitou podmínkou při obnově signálu je správná synchronizace přijatého signálu a kódu.



Na následujícím obrázku je výše popsaný princip zobrazen ve frekvenční oblasti. Zobrazen je pouze případ, kdy dochází k rozprostírání signálu. V případě obnovy signálu by byla situace obdobná, proces by probíhal opačným směrem a bylo by nutné brát v úvahu výše zmíněnou správnou synchronizaci.

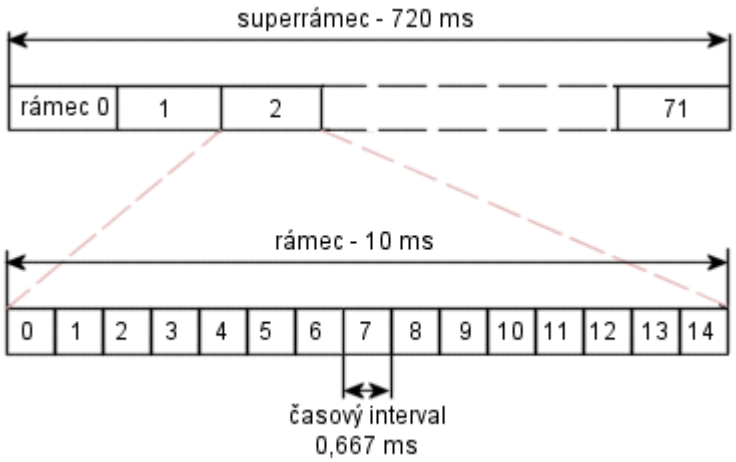


Výše popsaným způsobem je zakódováno více uživatelských kanálů najednou a po následující multiplexaci jsou vysílány všechny najednou ve stejném, širokém pásmu. Každý přijímač tedy přijme všechny zakódované kanály a pomocí svého, unikátního, kódu obnoví pouze signál jemu příslušející.

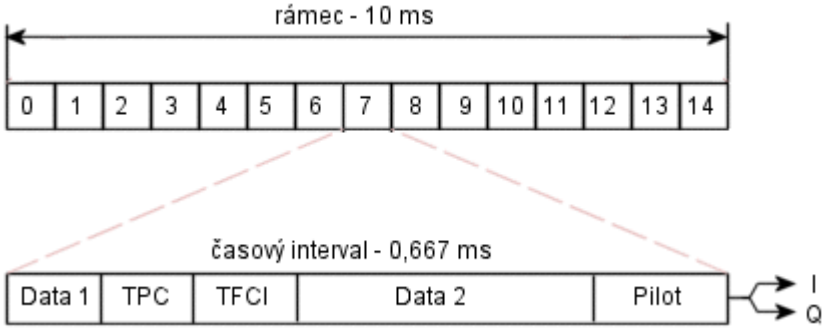
Režim FDD poskytuje maximální přenosovou rychlost 384 kbit/s. Využit bude pro pokrytí městské a venkovské zástavby a nabízí vysoký stupeň mobility. Přenosová rychlost módu FDD se mění na základě aktuální potřeby uživatele. Většinu spektra pro UMTS využívá právě režim FDD, doporučeno však je kombinovat párové a nepárové frekvence.

9.3.6.15.2. Struktura přenášených dat na fyzické vrstvě

Veškeré informace přenášené na sestupné i vzestupné trase jsou v režimu FDD uspořádány do rámců délky 10 ms a tyto rámce jsou dále rozděleny na 15 časových intervalů (time slot) o délce 0,667 ms. 72 rámců tvoří jeden superrámec, trvající 720 ms.

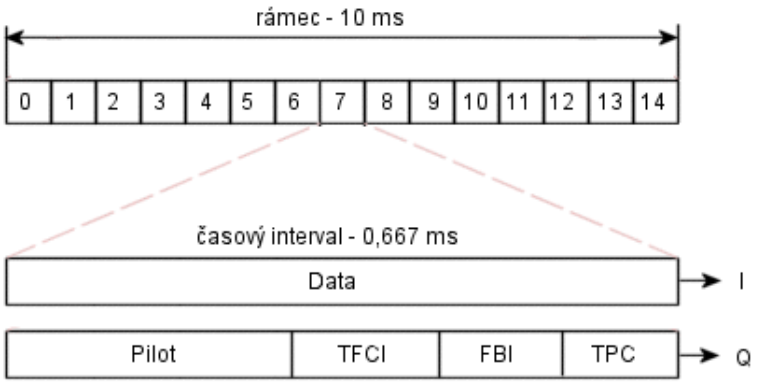


Pro další popis je třeba rozlišit směr komunikace. Na sestupné trase (**downlink**) je fyzický kanál vytvářen časovým multiplexováním dat a řídicích informací do jednoho časového intervalu. Počet obsažených bitů je proměnný v rozmezí 20-1280 bitů (v hodnotách 20 x 2^k, k=0...7), konkrétní uspořádání je popsáno obrázkem.



Časový interval obsahuje pilotní bity pro synchronizaci a odhad parametrů kanálu, bity TPC (Transmit Power Control) pro řízení vysílacího výkonu, bity TFCI (Transport Format Combination Indicator) pro řízení přenosové rychlosti a samozřejmě bity nesoucí uživatelská data. Datový tok je poté rozdělen na dvě větve a obě jsou rozprostřeny posloupností. Dále je provedeno skramblování a před modulací je datový tok filtrován filtrem s roll-off faktorem a=0,22.

Situace na vzestupné trase (**uplink**) je poněkud odlišná. Definovány jsou dva typy fyzických kanálů, které jsou přenášeny současně. Délka tentokrát může nabývat 10-640 bitů (opět v hodnotách 20 x 2^k, k=0...6). Oproti předcházejícímu případu se zde navíc přenášejí bity FBI (Feedback Information). Protože zde jsou přenášeny v podstatě dva různé datové toky o poloviční rychlosti, neprovádí se dělení na toky dva, ale do jedné větve vstupují data a do druhé řídicí informace. Následuje opět skramblování, filtrování a modulace. Situace je znázorněna na obrázku.



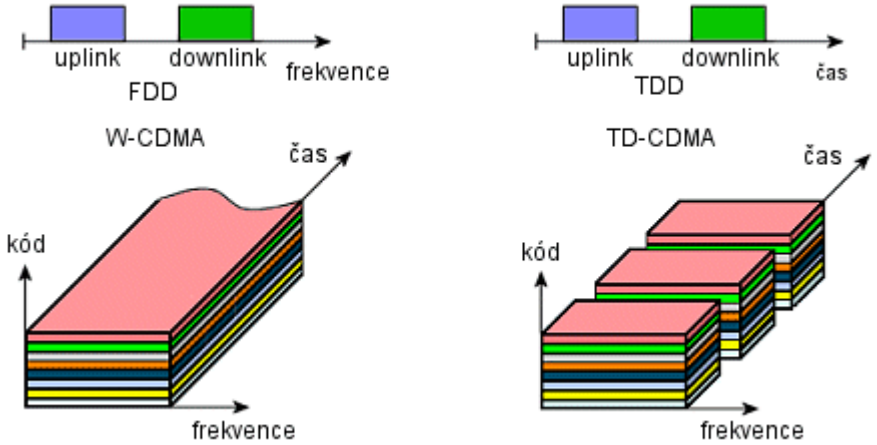
9.3.6.15.3. Přehled parametrů režimu FDD

Mnohonásobný přístup	W-CDMA (DS-CDMA)
Šířka pásma	5 MHz
Bezpečnostní vzdálenost (Frequency reuse distance)	1
Modulace	QPSK
Čipová rychlost (Chip rate)	3,84 Mčipů/s
Max. přenosová rychlost	384 kbit/s
Rozprostírající faktor (Spreading Factor)	4-256
Handover	Soft, softer (mezi frekvencemi: hard)
Délka rámce	10 ms
Trvání časového interalu	667 µs
Předmodulační filtr	Root raised cosine roll-off faktor 0,22

9.3.6.16. TDD

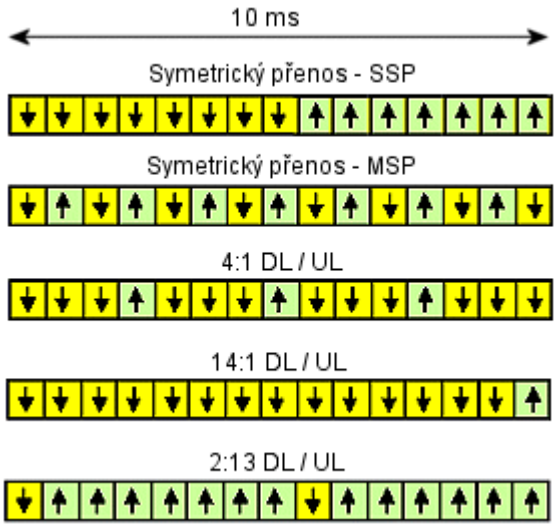
9.3.6.16.1. TD-CDMA

Pro režim TDD využívá **UTRA** (UMTS Terrestrial Radio Acces) jako metodu mnohonásobného přístupu širokopásmovou W-CDMA v kombinaci s časovým rozdělením vzestupných a sestupných tras. Jde tedy o kombinaci TDMA/CDMA a metoda je nazvána **TD-CDMA** (Time Division CDMA). Mnoho parametrů (délka rámce, počet časových intervalů, ...) je zde shodných s režimem FDD, některé se odlišují. Výhodou tohoto režimu je jednoduché vytvoření asymetrické komunikace, kdy je v každém směru použita jiná přenosová rychlost.

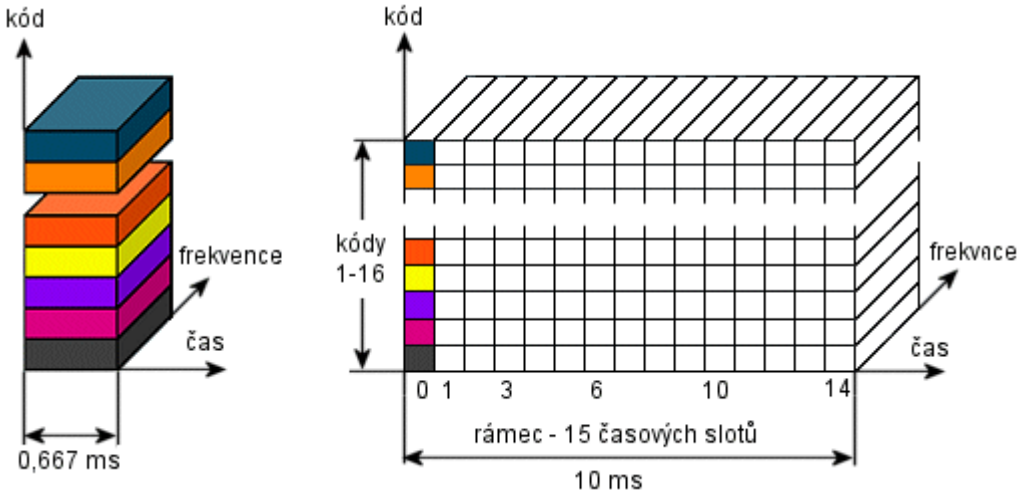


I zde (stejně jako v režimu FDD) jsou informace přenášeny v rámcích o době trvání 10 ms. Rámce jsou dále rozděleny na 15 časových intervalů (slotů) o délce 0,667 ms, které jsou přiděleny různým uživatelům. Protože je zde využita duplexní technika TDD, mohou být některé časové intervaly v rámci použity pro sestupnou trasu, jiné pro trasu vzestupnou. Dělení mezi těmito směry je proměnlivé a je vždy nastavováno podle aktuální situace, pouze první

časový slot slouží vždy pro trasu sestupnou. V celém rámci se může nacházet jeden bod změny směru (Single Switching Point) nebo může být směr komunikace měněn několikrát (Multiple Switching Points). Na obrázku je znázorněno několik příkladů rozdělení časových intervalů pro jednotlivé směry komunikace včetně dvou krajních případů.



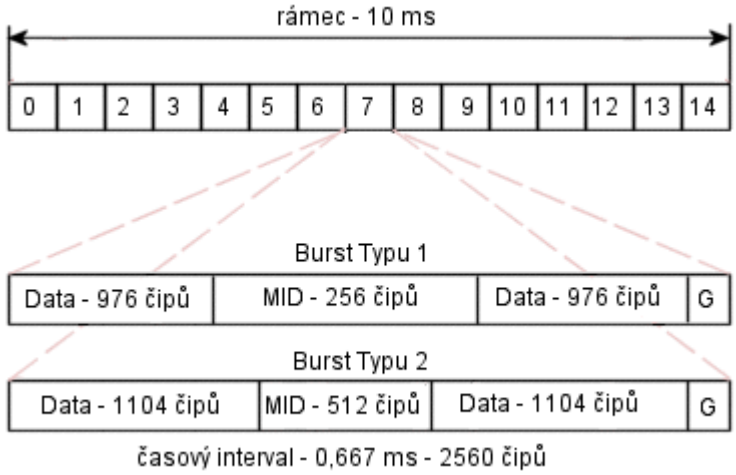
Vedle tohoto časového dělení je zde uvnitř časových intervalů pro další oddělení jednotlivých kanálů použito rozprostírajících kódů (maximálně 16). V rámci jednoho slotu je tak umožněna současná komunikace na více uživatelských kanálech.



Existují dvě možnosti přidělování kódu uživatelům:

- Více kódů s konstantním rozprostřením (**Multi Code with Fixed Spreading**): Přenosová rychlost v rámci jednoho časového intervalu je stále stejná a přenosová kapacita pro konkrétního uživatele v tomto slotu je zvyšována přidělováním dalších kódů, které jsou pak používány současně. Tyto kódy mohou být přiděleny jednomu uživateli nebo mohou být rozděleny mezi uživatelů více.
- Jeden kód s proměnlivým rozprostřením (**Single Code with Variable Spreading**): Opět jsou různým uživatelům v rámci jednoho časového slotu přidělovány různé rozprostírající kódy, ale jeden uživatel má vždy přidělen pouze jediný kód. Jestliže uživatel požaduje větší přenosovou rychlost, je zvýšena přenosová rychlost tohoto časového intervalu, tedy snížen rozprostírající faktor přiděleného kódu.

Přenášené bity jsou tedy v režimu TDD opět děleny do rámců a časových intervalů, jak ukazuje následující obrázek. V jednotlivých časových intervalech jsou dále bity přenášeny v blocích, označovaných pojmem burst. Každý burst obsahuje řídicí informace a uživatelská data rozprostřená rozprostírajícím kódem. Na závěr každého burstu je vložena ochranná doba (Guard Period). Počet přenášených bitů tedy závisí na rozprostírajícím faktoru, definován je pouze počet čipů. Definovány jsou dva základní typy burstů podle počtu přenášené informace v datové části a ve vložené části pro řídicí informace (midamble).



- Burst typu 1 - může být použit na vzestupné i sestupné trase nezávisle na počtu uživatelů v časovém intervalu
- Burst typu 2 - může být použit na vzestupné trase při počtu uživatelů v časovém slotu menším než 4 a sestupné trase nezávisle na počtu uživatelů v časovém intervalu

Režim TDD je určen především pro pokrytí městské zástavby, míst kde je vysoká koncentrace uživatelů a kde je potřeba nabídnout vysokou přenosovou rychlost. Pokrývá tedy většinou pouze malé oblasti (mikrobuňky a pikobuňky). Ideálně se hodí pro vyšší přenosové rychlosti až do 2 Mbit/s při malé mobilitě. Díky schopnosti zvládnout asymetrický tok dat se tento režim dokonale hodí pro bezdrátový přístup do podnikových sítí.

9.3.6.16.2. Přehled parametrů režimu TDD

Mnohonásobný přístup	**TD-CDMA**
Šířka pásma	**5 MHz**
Bezpečnostní vzdálenost (Frequency reuse distance)	**1**
Modulace	**QPSK**
Čipová rychlost (Chip rate)	**3,84 Mčipů/s**
Max. přenosová rychlost	**2 Mbit/s**
Rozprostírající faktor (Spreading Factor)	**1, 2, 4, 8, 16**
Handover	**Hard**
Délka rámce	**10 ms**
Trvání časového interalu	**667 µs**
Předmodulační filtr	**Root raised cosine roll-off faktor 0,22**

10. Bezdrátové technologie

10.1. IrDA

IrDA je standard vytvořený **IrDA** konsorciem (Infrared Data Association), který definuje bezdrátový přenos dat pomocí infračerveného záření. IrDA ve svých specifikacích definuje standardy, jak fyzických koncových zařízeni a protokolů, jimiž komunikuji IrDA zařízení. Tento standard vznikl z potřeby mobilně propojit různé zařízení mezi sebou (hlavní využití IrDA je pro spojení přenosných počítačů, různých personálních komunikátorů a mobilních telefonů, ale IrDA rozhraním jsou vybaveny například i videokamery). IrDA konsorcium vzniklo v roce 1993 za účasti 20 různých společností a dalo si za úkol navrhnout standard pro bezdrátový přenos dat na krátkou vzdálenost pomocí infračerveného záření. Již za rok, tedy v roce 1994, byla vydána první verze IrDA standardu.



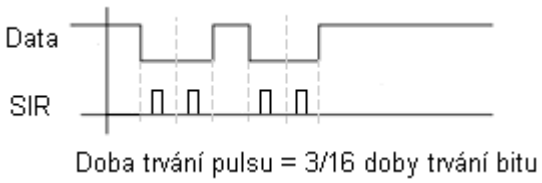
Dnes tento standard popisuje fyzickou vrstvu a protokoly nutné pro komunikaci dvou zařízení. První standard, označený IrDA 1.0, popisuje přenos dat rychlostmi od 2400 bit/s do 115200 bit/s, novější verze IrDA 1.1 obsahuje jistá vylepšení na fyzické vrstvě a umožňuje přenos dat rychlostí až 4 Mbit/s.

10.1.1. Fyzická vrstva standardu IrDA

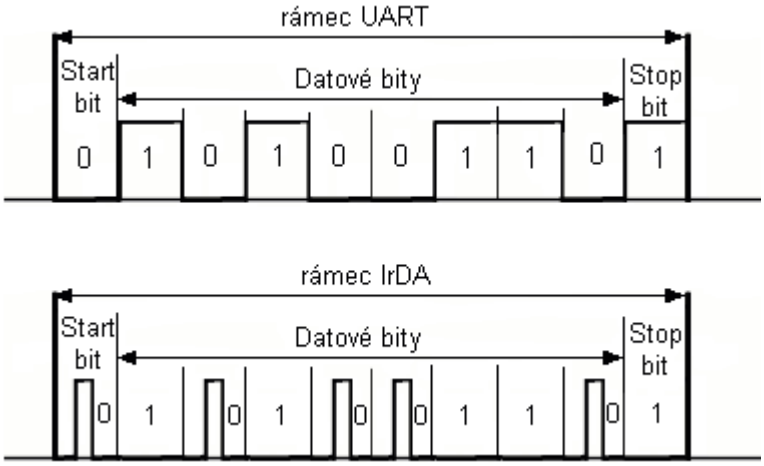
IrDA zařízení komunikují pomocí infračervených LED diod s vlnovými délkami vyzařovaného světla 875 nm (tolerance výroby asi 30nm). Přijímačem jsou PIN fotodiody, které pracuji v generačním režimu (při dopadu světla na přijímač světlo "vyrazí" elektrony, které se odvádí do elektronického filtru, který propustí jen ty frekvence, které jsou povoleny pro daný typ IrDA modulace). Existuje přímá úměra mezi energií dopadnutého záření a nábojem, který optická část přijímače vygeneruje. IrDA zařízení dle normy IrDA 1.0 a 1.1 pracují do vzdálenosti 1.0 m při bitové chybovosti BER 10^{-9} a maximální úrovni okolního osvětlení 10 klux (denní svit slunce). Tyto hodnoty jsou definovány pro nesouosost vysílače a přijímače 15 stupňů, pro jednotlivé optické prvky se měří výkon do 30 stupňů. Existují i směrové vysílače pro větší vzdálenosti, ty však nedodržují přesně definovaný standard.

10.1.1.1. IrDA 1.0

Přenosové rychlosti používané pro IrDA 1.0 jsou od 2400 do 115200 kbit/s (2400 bit/s, 9600 bit/s, 19,2 kbit/s, 38,4 kbit/s, 57,6 kbit/s, 115,2 kbit/s). Prvotní komunikace probíhá vždy rychlostí 9600 bit/s a proto je podpora této rychlosti oproti ostatním povinná. Používá se pulsní modulace, doba vyzařování světelné energie odpovídá 3/16 délky původní doby trvání bitu a pulsy odpovídají bitům s nulovou hodnotou. Tato metoda se také nazývá **SIR** (Serial InfraRed).

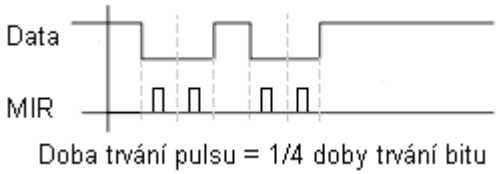


Formát dat je stejný jako na sériovém portu, tedy asynchronně vysílané slovo uvozené startovacím prvkem.



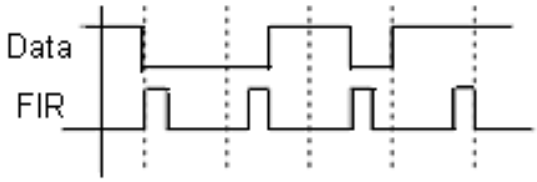
10.1.1.2. IrDA v.1.1

IrDA 1.1 definuje navíc rychlosti 0,576 Mbit/s a 1,152 Mbit/s s pulsní modulací 1/4 délky doby trvání původního bitu. Metoda se někdy označuje jako **MIR** (Medium Infrared), jedná se však o neoficiální název.



Při těchto rychlostech je již základní jednotka (paket) vysílán synchronně a je uvozen startovacíc sekvencí (2 x 8 bitů). Následuje cílová adresa (8 bitů), data ukončená 16 bity ochranného kódu CRC a koncová sekvence (8 bitů). Startovací a koncová sekvence se nikde jinde v toku dat nemůže vyskytnout.

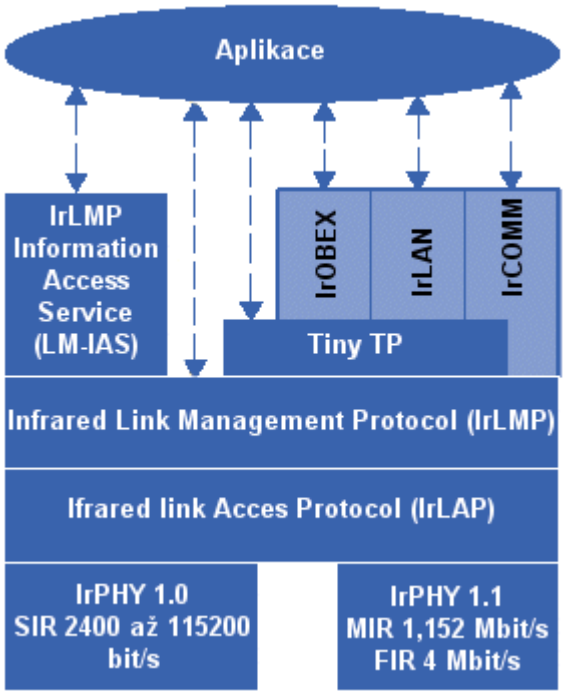
Pro rychlost 4 Mbit/s se používá impulsová polohová modulace **PPM** (Pulse Position Modulation), v níž se 2 bity informace zakódují do pulsu v jedné ze čtyř možných časových pozic – nositelem informace je zde tedy pozice pulsu v čase namísto existence pulsu jako u předchozích modulací. Například bity 00 se vyšlou jako sekvence 1000, bity 01 jako 0100. Tato metoda jse nazývá **FIR** (Fast Infrared). Pakety při této modulaci mají na rozdíl od rychlosti 0,576 a 1,152 Mbps použit kontrolní kód CRC-32.



Dvojice bitů	PPM
00	1000
01	0100
10	0010
11	0001

10.1.2. Protokoly technologie IrDA

Prvním IrDA standardem byl v roce 1994 popis fyzické vrstvy **IrPHY** (Infrared Physical Layer), pojednávající o přenosu dat pomocí infračerveného záření rychlostí do 115200 bit/s. Tato specifikace byla krátce poté následována a zveřejněním protokolu **IRLAP** (Infrared Link Access Protocol), což je protokol založený na protokolu HDLC uzpůsobený potřebám IrDA komunikace. Základní funkcí je řízení přístupu k přenosovému médiu a vytvoření spolehlivého spojení. Později byl přidán protokol **IrLMP** (Infrared Link Management Protocol), který má dvě základní funkce. První z nich je mechanizmus, zajišťující aby na vrstvě IrLAP mohlo probíhat více spojení zároveň. Tato funkce se nazýv8 LM-MUX (Link Management Multiplexer). Druhým úkolem je detekce přítomnost jiných IrDA zařízení a služeb, které tyto zařízení poskytují. Tato část nese název **LM-IAS** (Link Management Information Access Service). **Tiny TP** (Tiny Transport Protocol) je protokol, který udržuje virtuální kanál mezi zařízeními, zajišťuje řízení toku dat (flow control) a provádí rozdělení dat do paketů a jejich znovusestavení (SAR – Segmentation And Reassembly). Všechny doposud zmíněné protokoly jsou nutné pro správnou funkci IrDA zařízení a IrDA kompatibilní zařízení je musí podporovat.



Pro použití IrDA zařízení v běžné praxi bylo vytvořeno několik dalších protokolů, zpřístupňující tuto technologii mnoha aplikacím.

- **IrOBEX** (IrDA Object Exchange Protocol): Protokol založený na HTTP (HyperText Transfer Protocol), který slouží k přenosu informací (telefonní čísla, elektronické vizitky, údaje v kalendáři) mezi jednotlivými aplikacemi.
- **IrCOMM**: Protokol sloužící k emulaci sériového a paralelního portu. Mnoho existujících aplikací tak dokáže pracovat s IrDA zařízením bez jakékoliv změny.
- **IrLAN**: Tento protokol byl vyvinut, aby umožnil IrDA zařízením bezdrátový přístup do LAN pomocí infračerveného záření.
- **IrTran-P** (Infrared Transfer Picture): Protokol vyvinutý výrobci digitálních fotoaparátů, určený k přenosu digitálních obrázků.

10.2. Bluetooth

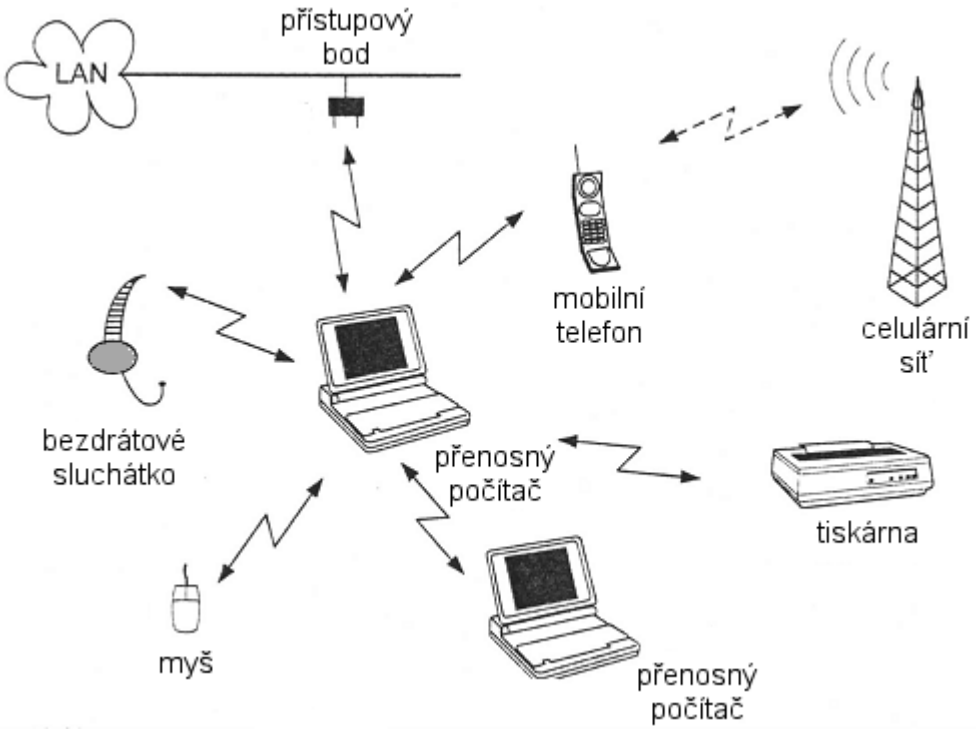
10.2.1. Základní charakteristika technologie Bluetooth

Perspektivní bezdrátový systém pochází od firmy Ericsson a nese označení Bluetooth. V roce 1998 zmíněná firma za podpory dalších společností IBM, Nokia, Toshiba a Intel, vytvořila speciální zájmovou skupinu **Bluetooth SIG** (Special Interest Group), kterou během několika následujících měsíců podpořilo více jak tisíc institucí z celého světa. Podle odborných prognóz z přelomu století potom má v roce 2002 technologii Bluetooth používat již asi 100 miliónů uživatelů počítačů, mobilních telefonů a dalších přidružených elektronických zařízení.

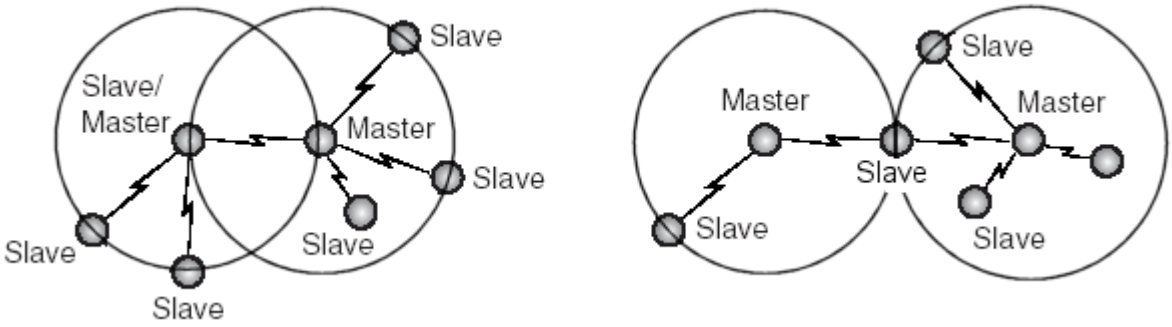
Bluetooth je univerzální rádiový systém (rozhraní), skládající se z velmi malých síťových struktur, označovaných pojmem pikonet (pikosíť). Z nich každá umožňuje, za pomoci miniaturních terminálů obsahujících rádiový transciever (vysílač/přijímač) a procesor základního pásma, současnou bezdrátovou datovou i řečovou komunikaci na krátké vzdálenosti. Každé zařízení pracuje v jednom z následujících režimů : master nebo slave. Jestliže zařízení jako první iniciuje sestavení sítě, pracuje v režimu master a plní funkci řídicí jednotky. Ta spočívá v identifikaci účastníků, zajištění jejich vzájemné synchronizace apod. Ostatní účastníci představují podřízené jednotky (slave). Tyto funkce však jsou dočasné a zanikají po zrušení dané sítě, takže v příštím cyklu může funkci řídicí jednotky plnit libovolný jiný účastník. V rámci jediné pikonetové struktury se této komunikace může účastnit celkem až osm terminálů (1 x master, 7 x slave), které jsou zabudovány přímo do hostitelských systémů, mezi něž patří např. osobní počítač, tiskárna, mobilní telefon apod. Veškerá komunikace probíhá přes jednotku master a není možné přímé rádiové spojení mezi jednotkami slave. Jestliže se v jedné síti pikonet nachází 1 slave a 1 master, jedná se o komunikaci bod-bod (point to point), je-li přítomno více zařízení jedná se o komunikaci bod-více bodů (point to multipoint).



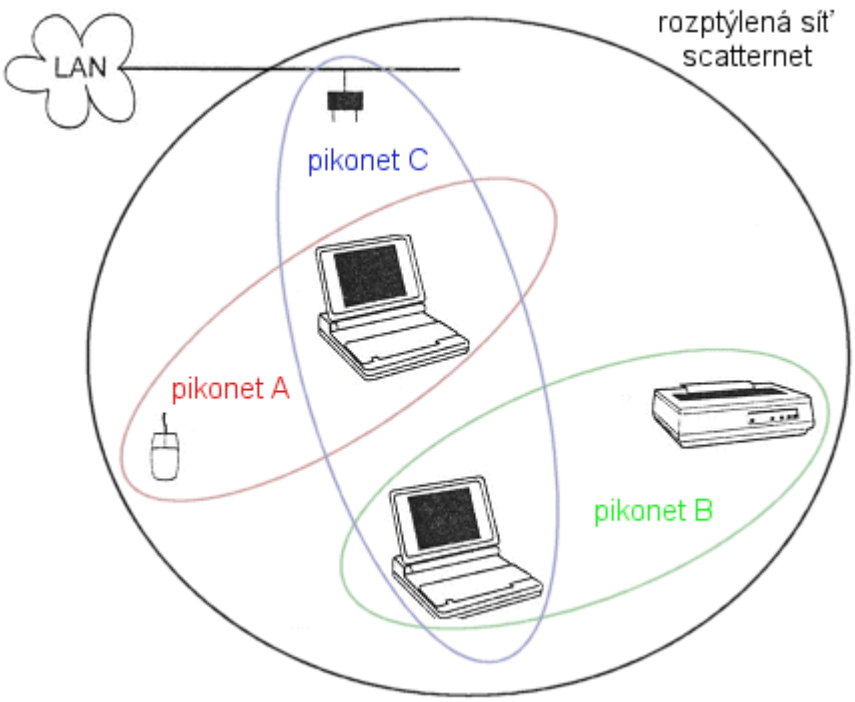
Příklad jedné samostatné pikosítě je na následujícím obrázku. Je zde znázorněna možnost spolupráce technologie Bluetooth s dalšími, fixními nebo mobilními sítěmi. Přenosný počítač zde plní funkci řídicí jednotky (master) a ostatní, s ním komunikující zařízení pracují v režimu slave.



V téže lokalitě nebo jejím okolí však mohou být přítomna další Bluetooth zařízení, převyšující počet 8, případně další sítě pikonet. Pro propojení více zařízení nebo pro propojení více samostatných pikosítí je použita struktura nazvaná scatternet (rozptýlená síť). V této struktuře jsou některá zařízení obsažena ve více pikosítích a zajišťují tak jejich propojení. Na obrázku jsou dva příklady rozptýlené sítě. V prvním případě je jedno zařízení zároveň master v jedné pikosíti a slave ve druhé, v dalším případě je zařízení v režimu slave pro obě propojované pikosítě.

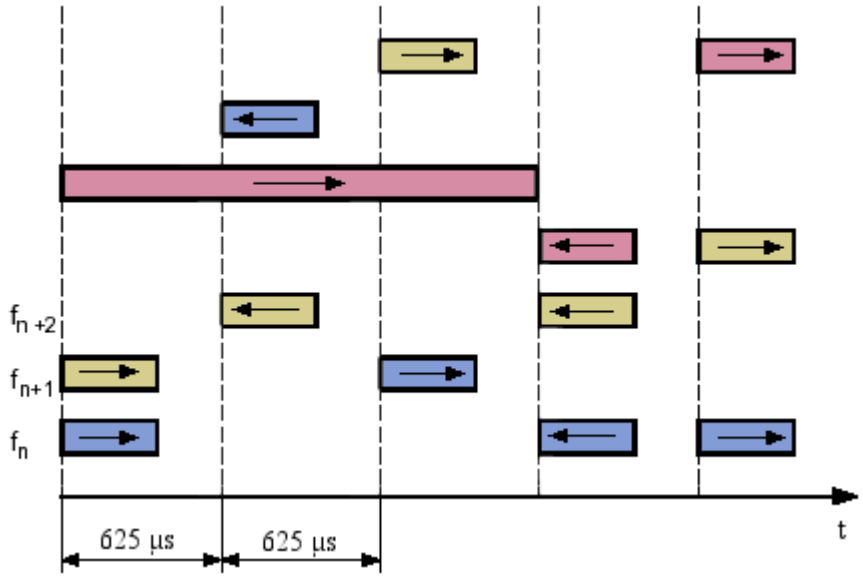


Opět je zde zobrazen příklad této struktury. Jedna rozptýlená síť (scatternet) se skládá z pikosítí A, B a C. Dvě z nich, A a B, slouží pro připojení periferních zařízení k přenosnému počítači a pikosíť C těmto počítačům poskytuje bezdrátové připojení do lokální počítačové sítě.

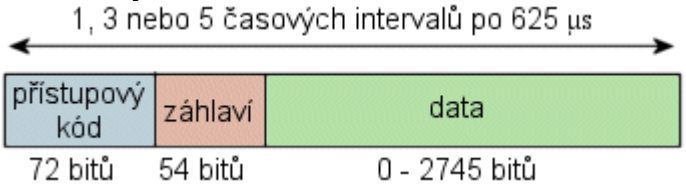


10.2.2. Zpracování signálu

K rádiovému přenosu využívá systém Bluetooth nelicencované pásmo **ISM** (Industrial, Scientific, Medical), určené pro průmyslové, vědecké a lékařské aplikace. V USA a v Evropě (kromě Francie a Španělska) se toto pásmo rozkládá mezi 2400 a 2483,5 MHz, v Japonsku mezi 2471 a 2497 MHz. Aby se minimalizovala možnost vzájemného rušení s jinými rádiovými zařízeními pracujícími v tomto pásmu nebo v jeho okolí (rádiové bezpečnostní systémy, kuchyňské mikrovlnné trouby apod.), používají terminály Bluetooth přenosu s rozprostřeným spektrem (s kódovým multiplexem). Jako optimální se v uvažované aplikaci ukázala varianta s kmitočtovým skákáním nosné vlny **FH-SS** (Frequency Hopping Spread Spectrum). Ta využívá ve státech s pásmem ISM o šířce 80 MHz celkem 79 kmitočtů, jinde 23 kmitočtů, vždy se vzájemným odstupem 1 MHz. Po nich přeskakuje podle vhodné pseudonáhodné sekvence nosná vlna, modulovaná binární modulací GFSK. Kmitočet přeskoků, tj. jejich počet za sekundu, je f = 1600, takže intervaly aktivního přenosu na dílčích nosných vlnách mají vždy dobu trvání T = 1/f = 1/1600 = 625 µs. V časové oblasti je přenos členěn do intervalů, které mají dobu trvání rovněž 625 µs. Toto ovšem platí pouze pro základní pakety – existují i pakety o trojnásobné a pětinásobné délce, které vyžadují delší časový interval, a tedy i pomalejší změnu nosné frekvence. Aby jednotlivá zařízení v rámci jedné pikosítě nezačala vysílat ve stejný okamžik na stejném kanále, je nutností vnést do přidělování frekvencí určitá pravidla. Právě o to se v každé pikosíti stará zařízení, které pracuje v režimu master. Přenos na jednotlivých rádiových kanálech je plně duplexní s aplikací časového duplexu (systém FHS/TDD). Použitý duplexní kanál využívá po sobě jdoucí sloty střídavě pro vysílání a příjem.



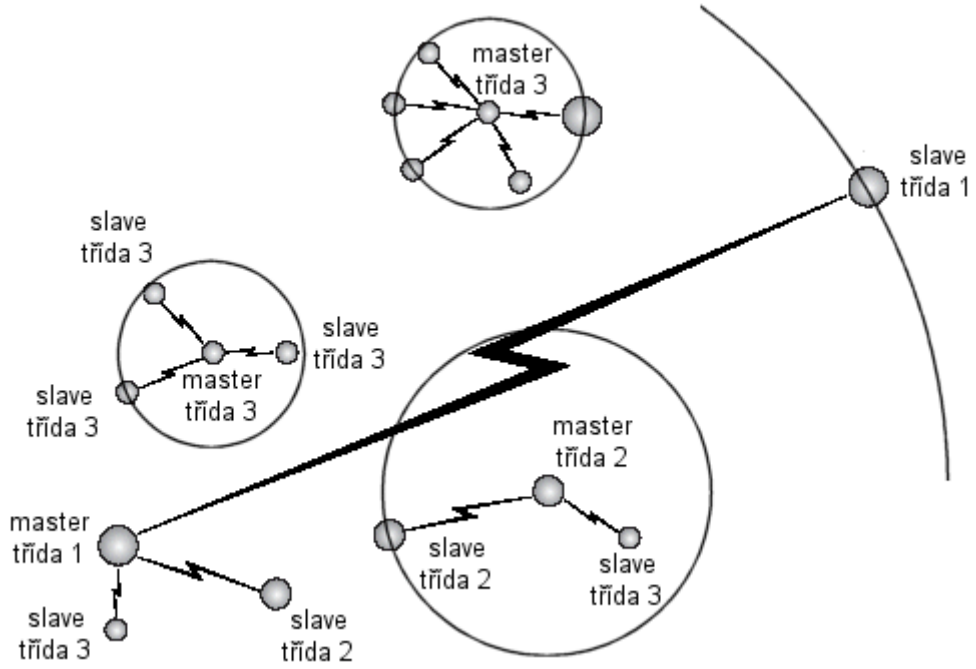
Přenos dat probíhá formou přepojování paketů v asynchronním režimu **ACL** (Asynchronous Connectionless Link), kde každý paket může mít délku jednoho, tří nebo pěti časových intervalů. V každém z těchto případů je paket přenášen na jediné frekvenci, určené prvním časovým intervalem. Nejdelší paket, označený DH5, obsahuje 2871 bitů, z toho 2745 přenáší uživatelská data. Po vyslání tohoto paketu musí vždy dojít k přenosu zpětné informace, jejíž minimální délka je jeden časový interval. Z tohoto plyne maximální přenosová rychlost při asymetrickém režimu 723,2 kbit/s v jednom směru a 57,6 kbit/s ve směru druhém. Přenos řeči se uskutečňuje v synchronním režimu **SCO** (Synchronous Connection Oriented). Digitalizovaný hovorový signál je zde rovněž členěn do časových intervalů, v každém intervalu je přenesen jeden paket a práce s delšími pakety zde není možná. Přenosová rychlost zde je 64 kbit/s v obou směrech. Každý paket začíná přístupovým kódem o délce 72 bitů, který je jedinečný pro každou pikosíť. Jeho úkolem je umožnit komunikaci pouze autorizovaným zařízením. Následuje záhlaví o 18 bitech, jeho délka se však ochranným kódováním zvyšuje na 54 bitů. Za záhlavím jsou již vlastní uživatelská data, která mohou obsahovat 0 až 2745 bitů. Celková délka paketu tak může být v rozmezí 126 až 2871 bitů.



Standard Bluetooth definuje podle maximálního vysílacího výkonu tři třídy zařízení.

- Třída 1–100 mW – komunikace v okruhu zhruba 100 m
- Třída 2 - 2,5 mW
- Třída 3–1 mW – komunikace v okruhu zhruba 10 m

Z následujícího obrázku je zřejmé, že v rámci jedné pikosítě je komunikace umožněna všem zařízením různých tříd, pokud jejich výkon postačí pro navázání kvalitní komunikace. Pikosítě se mohou, jak je také znázorněno, vzájemně překrývat, protože každá z nich pracuje s jinou sekvencí frekvenčních přeskoků. V případě kolize na jedné frekvenci se paket pošle znovu v následujícím časovém intervalu, kdy, již bude frekvence obou vysílačů s velkou pravděpodobností odlišná.



Rádiový přenos je v systému Bluetooth důkladně chráněn proti chybovosti. Ochrana hovorového modulačního signálu je zajištěna hlavně aplikací robustního zdrojového kódování, využívajícího modulaci delta s proměnnou strmostí **CVSD** (Continuous Variable Slope Delta Modulation). Toto opatření výrazně zvětšuje imunitu systému vůči interferencím, které jsou v nelicencovaném pásmu ISM často velice intenzivní. Možnost rušení vzájemně si blízkých sítí pikonet je potom výrazně zmenšena využitím některých principů protokolu mnohonásobného stochastického přístupu ALOHA. V systému je také důsledně kontrolována totožnost účastníků a přenos může být chráněn šifrováním před zneužitím.

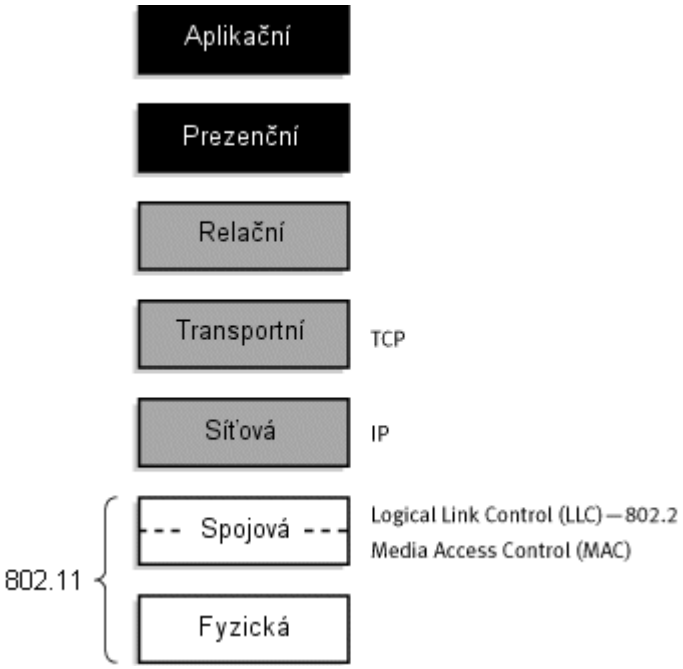
10.3. IEEE 802.11(Wi-Fi)

Americký standard 802.11 byl zveřejněn v roce 1997 americkým institutem **IEEE** (The Institute of Electrical and Electronics Engineers). Jedná se o specifikaci bezdrátové lokální sítě WLAN (Wireless Local Area Network), která nachází široké uplatnění v místech, kde je ekonomicky, technicky či jinak nevýhodné provozovat počítačové sítě s použitím metalických či optických kabelů. Další výhodou bezdrátových LAN je jistý stupeň mobility, která je uživatelům k dispozici. Původní standard 802.11 pracuje s přenosovou rychlostí 1 nebo 2 Mbit/s. V roce 1999 schválený standard **802.11b** navíc přidává přenosové rychlosti 5,5 a 11 Mbit/s, poslední modifikací je varianta **802.11a** s přenosovou rychlostí až 54 Mbit/s, která již přináší výkon srovnatelný s klasickými počítačovými sítěmi.

Standard 802.11 je zaměřen na dvě nejnižší vrstvy modelu OSI, fyzickou a spojovou vrstvu. Jakákoliv síťová aplikace, síťový operační systém nebo protokol (včetně nejznámějších TCP/IP a Novell NetWare) pracuje na síti kompatibilní s 802.11 stejným způsobem, jako při použití Ethernetu. Architektura systému, jeho vlastnosti a služby jsou popsány standardem IEEE 802.11, IEEE 802.11a a 802.11b přináší vyšší přenosové rychlosti a upravují pouze fyzickou vrstvu.

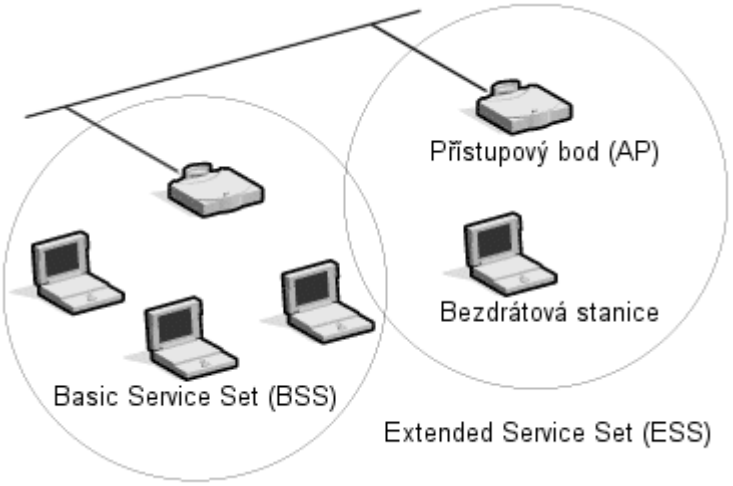
Standard 802.11 definuje dvě různá zařízení. Prvním je bezdrátová stanice, což je ve většině případů počítač vybavený vhodnou síťovou kartou NIC (Network Interface Card), druhým je přístupový bod (AP, Access Point), který slouží jako most (bridge) mezi bezdrátovou a fixní částí lokální počítačové sítě. S přístupovým bodem komunikují stanice, které vyžadují přístup k prostředkům, nacházejícím se ve fixní části sítě nebo v jiné, navazující síti (Internet).

Bezdrátové sítě tohoto standardu mohou pracovat ve dvou různých režimech podle použité architektury.



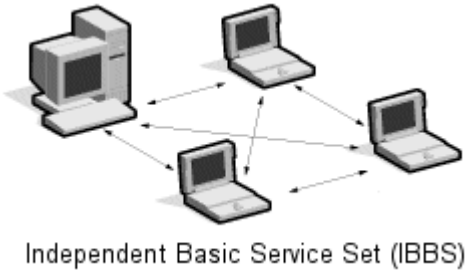
10.3.1. Infrastruktura

V tomto režimu se celá bezdrátová síť skládá z minimálně jednoho přístupového bodu AP napojeného na lokální počítačovou síť a dále několika bezdrátových stanic. Tato struktura se nazývá také **BSS** (Basic Service Set). Spojením několika BSS vznikne **ESS** (Extended Service Set), tvořící jedinou bezdrátovou lokální síť. Z důvodu častého požadavku přístupu bezdrátových stanic k tiskárně, internetu nebo souborovému serveru je tato struktura v současné době nejpoužívanější. Klasickým příkladem použití je bezdrátová počítačová síť uvnitř budovy s několika přístupovými body, poskytujícími přístup ke sdíleným prostředkům a internetu.



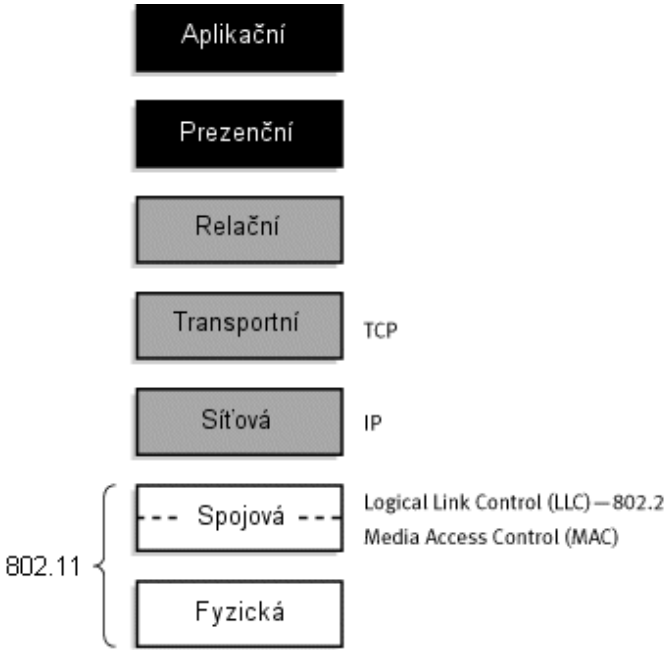
10.3.1.1. Ad hoc

Síť v tomto režimu je složena pouze z bezdrátových stanic a neobsahuje žádný přístupový bod do fixní počítačové sítě. Tato struktura může vzniknout na libovolném místě, není zde potřeba žádná fixní složka a všechna zařízení spolu komunikují navzájem. Tento režim komunikace se nazývá také peer to peer nebo **IBSS** (Independent Basic Service Set).



10.3.1.2. Fyzická a spojová vrstva standardu IEEE 802.11

Jak již bylo zmíněno, standard 802.11 definuje dvě nejnižší vrstvy modelu OSI – fyzickou a spojovou. Dále budou funkce a vlastnosti těchto vrstev stručně popsány.



10.3.1.3. Fyzická vrstva standardu 802.11

Původní standard 802.11 z roku 1997 definuje tři různé fyzické vrstvy. Dvě jsou založeny na rádiovém přenosu, třetí využívá infračerveného záření a všechny umožňují přenos dat rychlostí 1 nebo 2 Mbit/s. V případě rádiové komunikace je použito pásma pro průmyslové, vědecké a lékařské účely **ISM** (Industrial, Science and Medical band). Toto pásmo lze využít pro různé aplikace bez nutnosti získání licence a například v České republice se jedná konkrétně o 2400-2483,5 MHz. V obou případech je použito techniky rozprostřeného spektra (Spread Spectrum). První varianta rádiového přenosu využívá metodu frekvenčního přeskakování **FHSS** (Frequency Hopping Spread Spectrum), druhou možností je technika rozprostírání přímou posloupností **DSSS** (Direct Sequence Spread Spectrum). FHSS a DSSS jsou naprosto odlišné technologie a není mezi nimi možná přímá spolupráce.

- V případě použití frekvenčního přeskakování (FHSS) je pásmo 2,4 GHz rozděleno na 75 rádiových kanálů po 1 MHz. Při komunikaci dochází ke změnám těchto kanálů podle předem domluvené sekvence. Sekvence jsou vytvořeny tak, aby byla minimalizována možnost vysílání dvou zařízení na jednom rádiovém kanále současně. Tato technika umožňuje relativně jednoduchou realizaci, ale nedovoluje přenosové rychlosti vyšší než 2 Mbit/s.
- Při použití techniky rozprostírání přímou posloupností (DSSS) je pásmo rozděleno na 14 rádiových kanálů, v USA na 11. V prvním kroku je datová posloupnost vynásobena pseudonáhodnou posloupností, která má mnohem větší tzv. čipovou rychlost (chip rate), než je bitová rychlost dat. Poté je takto rozprostřená posloupnost modulována na nosnou frekvenci pomocí PSK modulace.
- Poslední možností přenosu dat je použití infračerveného záření na frekvencích 300-428 THz. Tento princip přináší bezpečnější komunikaci z hlediska možného odposlechu a rušení, na druhou stranu nevýhodou je nutnost přímé viditelnosti mezi komunikujícími zařízeními.

V roce 1999 byl standard rozšířen o variantu **802.11b**, která umožňuje komunikaci přenosovou rychlostí 5,5 a 11 Mbit/s, opět v pásmu ISM. Pro tyto rychlosti je nutné použití technologie DSSS. Systémy 802.11b tak jsou zpětně kompatibilní pouze se systémy 802.11 založenými na technice DSSS a nikoliv FHSS. Pro umožnění práce v prostředí, které je silně rušené, používá systém 802.11b princip dynamického přizpůsobování přenosové rychlosti. V

horších podmínkách je tedy přenosová rychlost automaticky snižována až na původních 1 Mbit/s, při návratu do oblasti s vhodnými podmínkami je možný návrat na 5,5 nebo 11 Mbit/s.

Modifikace **802.11a** opět definuje pouze vlastnosti fyzické vrstvy a umožňuje přenos dat rychlostí až 54 Mbit/s. Pracuje v nelicencovaném pásmu 5150-5350 MHz nazvaném **UNII** (Unlicensed National Information Infrastructure), které je rozděleno na 8 subpásem po 20 MHz. Použita je zde technologie **OFDM** (Orthogonal Frequency Division Multiplex). Tato technika nejdříve rozdělí rychlý datový tok na několik pomalejších datových toků, které se potom modulují na stejný počet nosných vln a jsou přenášeny paralelně. V tomto případě je pásmo 20 MHz rozděleno na 52 kanálů po 300 kHz, z nichž slouží 48 pro přenos dat a ostatní pro signalizaci a detekci chyb. V závislosti na použité modulaci lze potom dosáhnout přenosové rychlosti až 54 MBit/s.

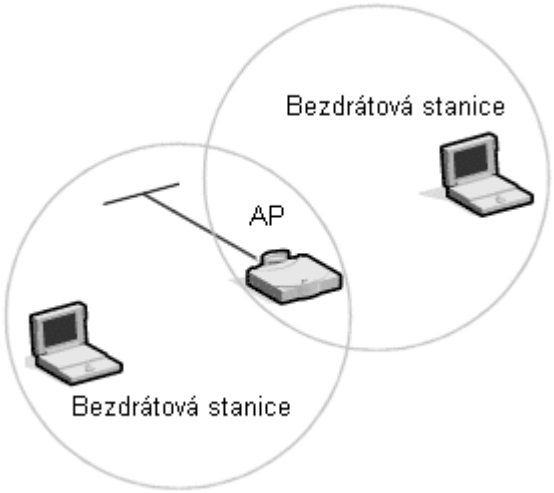
Použitá modulace	Přenosová rychlost
BPSK	6 Mbit/s
QPSK	12Mbit/s
QAM (16 stavů)	24 Mbit/s
QAM (64 stavů)	54 Mbits

10.3.1.4. Spojová vrstva standardu 802.11

Spojová vrstva (Data Link Layer) standardu 802.11 se skládá ze dvou podvrstev:

- LLC (Logical Link Control): Je zde použito protokolu 802.2, shodného pro všechny typy lokální datové komunikace dle standardů IEEE 802.
- MAC (Media Access Control): Velice podobný protokol jako u Ethernetu (802.3). Vrstva má na starost především řízení přístupu více uživatelů ke sdílenému médiu.

Na rozdíl od Ethernetu dle specifikace 802.3, který na vrstvě MAC používá metodu mnohonásobného přístupu ke sdílenému médiu s příposlechem nosné a detekcí kolizí **CSMA-CD** (Carier Sense Multiple Access - Collision Detection), standard 802.11 využívá principu **CSMA-CA** (Carier Sense Multiple Access - Collision Avoidance). Tato metoda kolize nedetekuje, ale snaží se jim vyhnout. Vysílající stanice "poslouchá" a když po určitou dobu nezjistí žádnou aktivitu, začne vysílat. Jestliže přijímající strana přijme rámec v pořádku, odpoví rámcem ACK, který říká že přenos proběhl bezchybně. V případě neúspěšného doručení rámce ACK vysílací stanici (ať z důvodu neúspěšného přenosu datového rámce nebo rámce ACK) se tato stanice pokusí o přenos dat znovu až po náhodně zvoleném časovém intervalu. Další problém, který je řešen na MAC vrstvě, je tzv. problém skrytého uzlu (hidden node problem), který je znázorněn na následujícím obrázku.



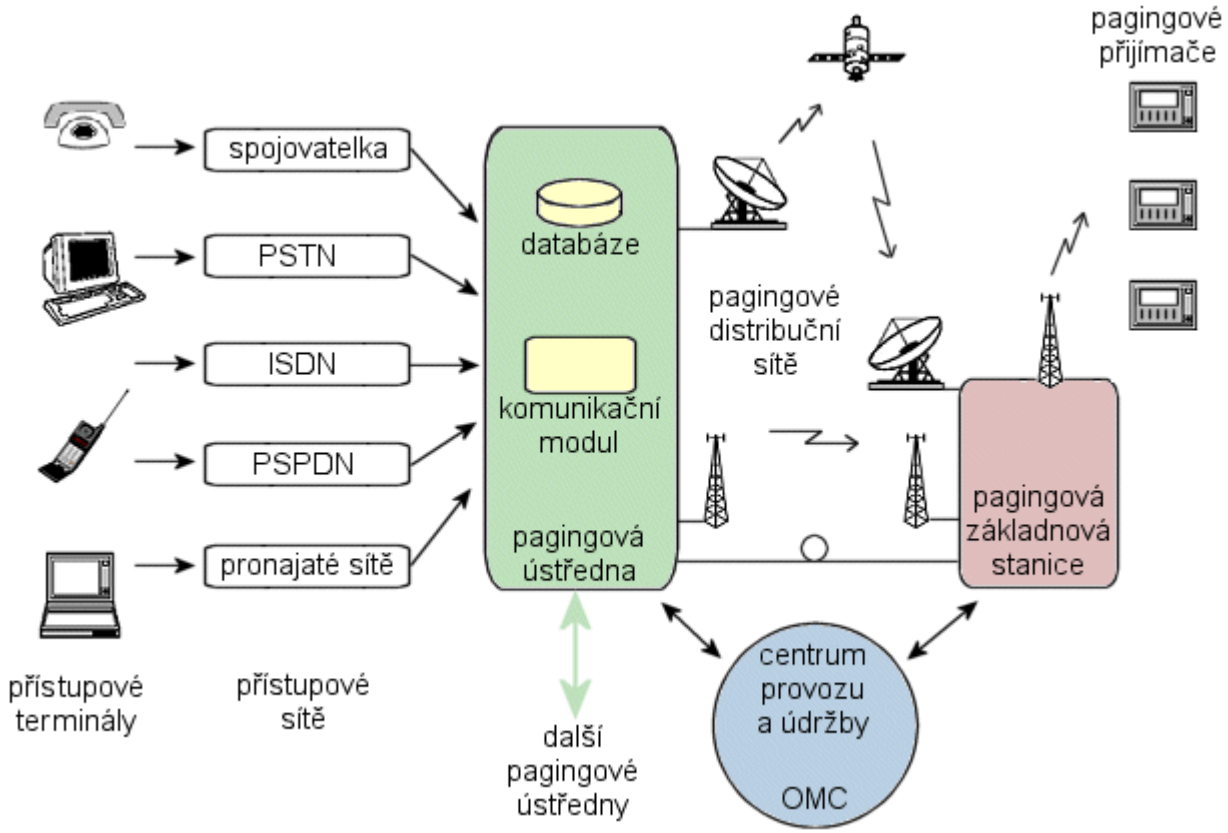
Protože se dvě stanice v rámci jednoho přístupového bodu AP nemusí navzájem "slyšet", je především při přenosu velkých paketů použit mechanizmus **RTS/CTS** (Ready To Send/Clear To Send). Stanice před začátkem vlastního vysílání požádá přístupový bod o přidělení kanálu krátkým rámcem RTS a dostane odpověď CTS. Ten již přijímají všechny stanice v dosahu přístupového bodu a na jistou dobu omezí své vysílání.

10.4. Paging

Paging (též označovaný jako systém pro vyhledávání osob, systém jednosměrného návěštění nebo radiokontaktní služba) je jeden z moderních prostředků rádiové mobilní komunikace. Veřejný rádiový paging je systém pro předávání krátkých zpráv z veřejných telekomunikačních sítí na alfanumerické displeje kapesních pagingových přijímačů (pager), většinou pracujících v pásmech velmi krátkých vln. Přitom zprávu je možné adresovat zvolenému příjemci nebo skupině příjemců. Přijímače jsou velice malá zařízení, protože nemusí obsahovat mikrofon, sluchátko, klávesnici ani vysílací část. V současné době jsou odhady rozvoje pagingových sítí spíše konzervativní z důvodu prudkého rozvoje veřejných mobilních systémů, které poskytují službu přenosu krátkých zpráv SMS.
Předávané zprávy mohou být:

- Tónové – přenáší se určité posloupnosti tónů, které odpovídají předem definovaným zprávám
- Numerické – vyjadřuje telefonní číslo pro zpětné volání nebo zakódovanou zprávu
- Alfanumerické – zpráva o délce zpravidla několika desítek znaků
- Datové – pro přenos obecných datových informací
- Hovorové – použito zpravidla v neveřejných systémech

Na následujícím obrázku je znázorněna základní architektura pagingového systému.



Zpráva, která má být doručena, se nejdříve do pagingového systému zadá pomocí vhodného přístupového terminálu (access trminal). Toto sdělení se poté přes přístupové sítě (access network) dostane k vlastnímu pagingovému systému. V prvním stádiu používání této technologie byl jako přístupový terminál využíván pouze běžný analogový telefonní přístroj, který ovšem umožňoval předání pouze numerických zpráv. Pro zajištění alfanumerických zpráv je v systému nutná přítomnost spojovatelky, která hovorové sdělení přijme a převede do podoby odpovídající komunikačnímu protokolu. Stejným způsobem lze využívat i mobilní telefony. Postupem času však přibyly další možnosti přístupu k pagingovým službám, a to především prostřednictvím rychle se rozvíjejících datových sítí. Uživatelé tak mohou zprávy zasílat elektronickou poštou, pomocí krátkých textových zpráv z mobilních telefonů a z internetových stránek.

Zadaná zpráva dále přichází do pagingové ústředny (**RPS** – Radio Paging Switch), která obsahuje jednak komunikační modul a dále databázi, která obsahuje informace o jednotlivých účastnících, povolených službách, přijímačích, geografické oblasti atd. Každé sdělení, přicházející do komunikačního modulu, se nejdříve na základě informací uložených v databázi analyzuje, ověří se identita účastníka a nárok na vyžadovanou službu. Zpráva je dále zařazena do řady čekajících na odeslání do dalšího funkčního bloku, kterým jsou distribuční sítě. Řazení do "fronty" není závislé pouze na času příchodu, ale také na prioritě odeslání. Priorita je u pagingových sytému běžným prvkem a zmenšuje problémy s poměrně malou přenosovou kapacitou těchto systémů. Zprávy s menší prioritou tak nejsou doručeny v reálném čase ale i se zpožděním, což je v některých případech postačující. Systém tak může obsloužit větší množství zákazníkům než dovoluje okamžitá přenosová kapacita, limitovaná především jeho rádiovými kanály.

Dalším blokem v celkové koncepci systému je pagingová distribuční síť (**PDN** – Paging Distribution Network). Ta předává pagingové zprávy ke stacionárním základnovým stanicím.

Distribuční síť využívá různé přenosové prostředky:

- Radioreléové spoje
- Metalické spoje
- Optické spoje
- Družicové spoje

Pagingové základnové stanice (**PBS** – Paging Base Station) přijímají vhodným přijímačem pagingové signály přicházející z distribuční sítě. Tyto signály jsou zde zesilovány, kmitočtově transponovány, demodulovány a upraveny do formátu vhodného pro následující vysílání. Po následující modulaci a zesílení jsou vysílány k pagingovým přijímačům (paging reciever, pager).

Samozřejmou součásti pagingového systému je operační a dohledové centrum (**OMC** – Operational and Maintenace Centre), které má na starost kontrolní a monitorovací funkce, řeší otázky tarifikace a odstraňuje případné technické poruchy.

10.4.1. Rádiové rozhraní

V následující části je obecně popsán rádiový sektor pagingových systémů, tzn. způsob jakým je signál přenášen rádiovým rozhrním od pagingové základnové stanice (PBS – Paging Base Station) k pagingovým přijímačům. V základnové stanici dochází k úpravám signálu, nutným pro následující vysílání. Při těchto úpravách se zajišťuje především přesná časová synchronizace vysílání zpráv, která je velmi důležitá zejména u moderních pagingových systémů. Upravené zprávy dále přicházejí do vlastní vysílací části základnové stanice, kde se modulují na nosné vlny, zesilují a vysílají.

Vysílání se uskutečňuje například v okolí frekvencí 170 MHz, 450-470 MHz a mnoho dalších, obecně však v řádu stovek MHz. Tato pásma se oproti systémům mobilních telefonů nacházejí níže, což se příznivě projevuje v lepším přirozeném pokrytí obsluhovaných oblastí pagingovým signálem, který se na těchto frekvencích díky snazšímu ohybu a lomu rádiových vln snáze dostává i do nepřístupných oblastí.

Všechny moderní pagingové systémy používají digitální modulaci nosných vln, nejčastěji dvoustavovou nebo čtyřstavovou **FSK** (Frequency Shift Keying – klíčování frekvenčním zdvihem). Tyto modulace se vyznačují dobrou imunitou vůči šumu a interferencím, přítomným v pozemních rádiových kanálech. Tyto modulace také umožňují realizaci poměrně jednoduchých a malých pagingových přijímačů

Výstupní výkony vysílačů základnových stanic bývají řádu několika desítek až stovek wattů. Díky tomu může být síť pagingových základnových stanic mnohem řidší, než je tomu u celulárních telefonních systémů. Tyto velké výkony s sebou však přinášejí nebezpečí vzájemného rušení s jinými radiokomunikačními službami. Toto nebezpečí však lze eliminovat vhodným frekvenčním plánováním a vhodným rozmístěním základnových stanic.

V následující tabulce jsou uvedeny základní parametry rádiového přenosy některých pagingových systémů (u obousměrných technologií se parametry týkají komunikace základnová stanic – pagingový přijímač).

	ERMES	POCSAG	FLEX	ReFLEX 25(50)	PACS
Frekvenční pásmo	169 MHz	Různé	Různé	929-941 MHz (940-941 MHz)	940-940,5 MHz
Šířka kanálu	25 kHz	12,5 nebo 25 kHz	25 kHz	25 kHz (50 kHz)	12,5 kHz
Modulace	4 - FSK	2 - FSK	2 nebo 4 - FSK	2 nebo 4 - FSK	GMSK

10.4.2. Pagingový systém ERMES

Evropský standardizační institut ETSI začal v roce 1989 vyvíjet evropský rádiový pagingový systém **ERMES** (European Radio Message System). Cílem bylo navrhnout pagingový systém splňující následující kritéria:

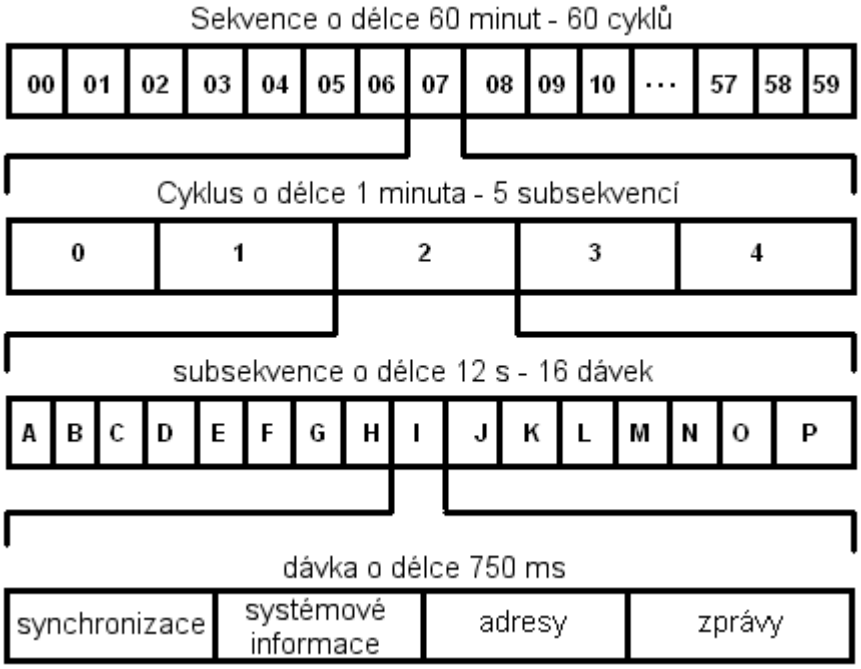
- Mezinárodní roaming v zemích EU
- Možnost přítomnosti více operátorů v každé zemi
- Vyšší přenosová rychlost než starší systémy
- Sjednocení poskytovaných služeb
- Použitelnost jediného zařízení v celé Evropě



Tento standard byl dokončen v roce 1992, od kdy postupně začal fungovat v mnoha evropských zemích.

Zprávy zadávané přístupovými terminály se v sytému ERMES dostávají přes ústřednu a distribuční siť do základnových stanic a odtud jsou vysílány rádiovými vysílači k jednotlivým pagingovým přijímačům. Toto vysílání probíhá v pásmu 169,4 MHz až 169,8 MHz, v němž je umístěno celkem 16 rádiových kanálů se vzájemným odstupem 25 kHz. K samotnému přenosu je použito čtyřstavové FSK, přenosová rychlost signálu je 6,25 kbit/s. Přenos je zabezpečen ochranným kanálovým konvolučním kódováním a je aplikována metoda prokládání (interleaving), zvyšující imunitu vůči shlukům chyb způsobeným vícecestným šířením.

Na následujícím obrázku je popsána struktura dat, přenášených v systému ERMES. Přenášená informace je uložena v kódových slovech o délce 30 bitů. Ty jsou dále hierarchicky řazeny do různých datových bloků.



Nejdelší jednotkou je sekvence, trvající přesně 1 hodinu. Všechny sítě ERMES startují sekvence přesně ve stejný okamžik, což je zajištěno napojením na UTC (Coordinated Universal Time). Sekvence je dále dělena na 60 cyklů, trvajících 1 minutu. Každý cyklus obsahuje 5 subsekvencí, po 12 s. Těchto 12 sekund obsahuje 16 dávek (batch) označených A až P. Každá z prvních 15 dávek obsahuje 154 kódových slov o délce 30 bitů, poslední dávka obsahje kódových slov 190. Jedna subsekvence tedy obsahuje 15 x 154 + 1 x 190 = 2500 kódových slov.)

Aby mohl přijímač skenovat všech 16 rádiových kanálů (v místech, která jsou pokryta signálem více sítí), probíhá vysílání tak, že např. dávka A je vysílána první v rádiovém kanálu č. 1, osmá v rádiovém kanálu č. 2 (rádiový kanál připadá jiné síti) a šestnáctá v rádiovém kanále č. 3. Protože je adresa každého pagingového přijímače "svázána" s určitou dávkou, přijímač "hledá" svou adresu pouze v okamžicích, odpovídajícím příslušející dávce a rádiovému kanálu.

Na začátku každé dávky jsou synchronizační bity (2 x 30 bitů), následují systémové informace (3 kódová slova po 30 bitech) a poté pole adres (proměnný počet kódových slov, maximálně 140). Adresy jsou zde řazeny sestupně, přijímač se proto může vypnout okamžitě, jestliže se v seznamu není jeho adresa. Jestliže se zde adresa vyskytne, přijímač přijímá dále a čeká na příchozí zprávu. Celý tento proces tedy výrazně šetří energii - protože přijímač je aktivní pouze v době, kdy by mu zpráva mohla přijít a ostatních okamžicích je vlastně vypnutý.

10.4.2.1. Služby sytému ERMES

Dříve popsaných 16 rádiových kanálů může být v určitém státě rozděleno až mezi 8 operátorů, z nichž každý je schopen provozovat pagingový systém při využití dvou kanálů pro 4,2 milióny uživatelů. Celkem lze tedy v rámci jednoho státu zajistit pagingový provoz až pro 32 miliónů účastníků. Systém je přizpůsoben pro použití mezinárodního roamingu. Protože však systém není schopen lokalizovat daný přijímač a zprávy vysílá na svém celém území, je nutno alespoň přibližně specifikovat oblast, ve které se uživatel nachází (v případě mezinárodního roamingu uživatel oznámí do které země cestuje a zprávy jsou poté zasílány do této oblasti).

ERMES podporuje všechny hlavní druhy pagingových služeb:

- Tónový paging osmi tónů
- Numerický paging do 20 numerických znaků
- Alfanumerický paging do 400 znaků
- Transparentní přenos dat

Další službou jsou takzvané skupinové zprávy, které je možno zaslat určité skupině příjemců. Odesílatel má také možnost zadat prioritu zprávy. ERMES definuje 3 stupně priority:

- Priorita 1 - zpráva je doručena do 1 minuty, pracuje pouze v rámci jednoho státu
- Priorita 2 - zpráva je doručena do 2,5 minuty v rámci státu a do 10 minut mezinárodně
- Priorita 3 - není garantován čas doručení

Tyto uvedené služby a režimy spolu s přenosovou rychlostí 6,25 kbit/s dovolují vytvářet řadu různých aplikací. ty nejsou vázány na přijímací straně nijak vázány na klasickou či jinou komunikační síť a lze je tedy používat i v odlehlých lokalitách. Zde jsou některé příklady použití:

- Dálkové ovládání různých strojů, zapínání a vypínání jistých zařízení, ovládání bezpečnostního zařízení
- Předávání aktuálních informací (dopravní informace, ...) do lokalit bez jiných komunikačních kanálů
- Pagingové přijímače zabudované do přenosných počítačů či jiných mobilních zařízení, do kterých lze i bez přítomnosti obsluhy přenášet data

Velkým konkurentem pagingovým systémům jsou v dnešní době veřejné mobilní celulární systémy, které poskytují službu přenosu krátkých zpráv SMS a nabízejí stejné, někdy i kvalitnější služby.

10.4.3. Ostatní pagingové systémy

10.4.3.1. AUTORUF

Jedná se o první veřejný systém pro přenos zpráv do vozidel, který začal pracovat v roce 1958 ve Švýcarsku. Pracoval na kmitočtu 80 MHz a používal amplitudovou modulaci.

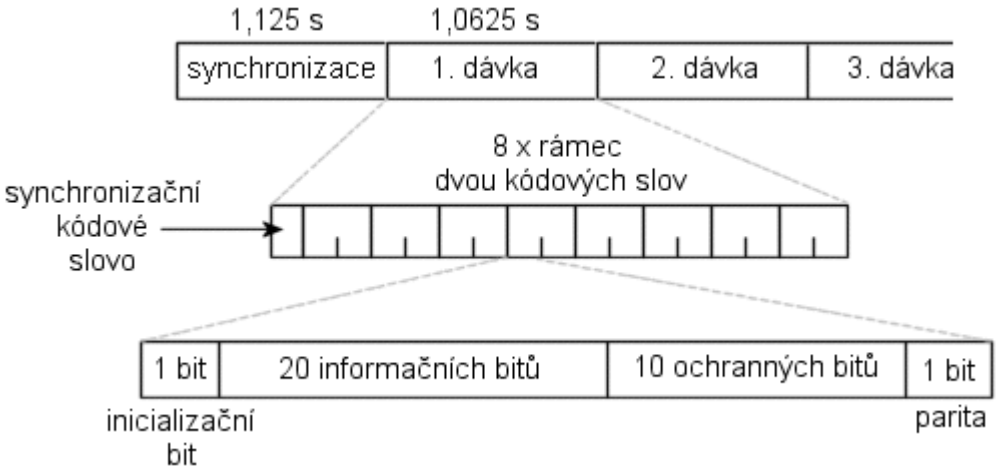
10.4.3.2. EUROSIGNAL

Systém, který byl původně navržen jako celoevropský, pracoval na podobném principu jako AUTORUF. K jeho rozšíření však nedošlo.

10.4.3.3. POCSAG (Post Office Code Standardization Advisory Group)

Tento systém byl vyvinut v letech 1975-1978 ve Velké Británii, později (1982) byl převzat do doporučení CCIR a používá se ve veřejných i neveřejných systémech. Každá země, která tento standard realizovala, používá jiné frekvenční pásmo a pagingové přijímače proto nejsou v různých zemích kompatibilní. Systém pracuje v pásmu 450 MHz od roku 1989 v Německu a ve Francii. Od roku 1991 tento standard používá mezinárodní systém EUROMESSAGE (Německo, Itálie, Velká Británie, Švýcarsko, Francie), který pracuje v pásmu 466 MHz.

Systém pomocí dvoustavové modulace FSK vysílá na nosné frekvenci data přenosovou rychlostí 512 bit/s (později byly specifikovánu varianty 1024 bit/s a 2048 bit/s). Struktura přenášených dat je na obrázku.



Každá skupina dávek začíná synchronizační sekvencí délky 1,125 s, obsahující 576 bitů (opakující se nuly a jedničky). Každá dávka obsahuje 544 bitů, doba jejího trvání je 1,0625 s a skládá se ze 17 kódových slov. První z nich je synchronizační a další jsou po dvou sdruženy do rámců a nesou informaci. Každé kódové slovo má délku 32 bitů. První, inicializační, bit udává zda je v informačním poli přenášena adresa pagingového přijímače, nebo zda je zde obsažena zpráva samotná. Poté následuje 20 informačních bitů, 10 bitů pro korekci a detekci chyb a následuje jeden bit paritní. Podobně jako u sytému ERMES je zde kladen důraz na úsporu energie. Poněvadž může být každý přijímač adresován pouze v určitém rámci, nemusí být aktivní celou dobu a tím výrazně šetří energii. Systém nabízí přenos tónových (4 tóny), numerických (14 znaků) a alfanumerických (80 znaků) zpráv.

10.4.3.4. FLEX (Flexible Speeded Wide Area Synchronous Protocol), ReFLEX, InFLEXion

Tento pagingový standard vyvinula a vlastní společnost Motorola a v současné době je konkurenčním systémem pro ERMES. Později byly zpracovány i další varianty tohoto pagingového systému (obousměrný pagingový systém ReFLEX a systém InFLEXion, přenášející hlasové zprávy). Dle požadavku provozovatele tohoto systému existují varianty systému FLEX, pracující s různými přenosovými rychlostmi. Během jedné periody, trvající 4 minuty, je přeneseno 128 rámců. Každý rámec obsahuje jeden synchronizační blok (115 ms) a 10 datových bloků (každý 160 ms).

Přenosová rychlost	Modulace	Počet kódových slov v bloku
1600 bit/s	2 - FSK	8
3200 bit/s	2 nebo 4 - FSK	16
6400 bit/s	4 - FSK	32

Úspora energie je zde opět řešena způsobem použitým u všech předchozích systémů. Pager přijímá pouze rámce, odpovídající jeho adrese, a ve zbylém čase je vypnut. Dalším vývojovým stupněm je systém ReFLEX, který umožňuje obousměrnou komunikaci. Varianta InFLEXion umožňuje přenášet hlasové zprávy na pagingový přijímač, kde je možné tyto zprávy poslouchat. Princip je zde v podstatě shodný s předchozími, zprávami jsou nyní pouze bloky digitalizovaného hlasu. Parametry rádiového přenosu jsou uvedeny v přehledu pagingových systémů

10.4.3.5. pACT (personal Air Communications Technology)

Pagingový standard vyvíjený pod patronací společností AT&T, Ericsson a NEC. Jedná se opět o obousměrný systém, narozdíl od ReFLEXu má v obou směrech stejnou přenosovou kapacitu - 8000 bit/s. Parametry rádiového přenosu jsou opět uvedeny v přehledu.

10.4.3.6. RDS (Radio Data System)

Systém využívající sítě FM VKV vysílačů (pásmo II, 87,5-108 MHz). Informační datový tok je modulován na subnosný kmitočet 57 kHz pomocí FM modulace, se zdvihem 4 kHz. Datová informace e přenáší pomocí tzv. skupin (group), z nichž každá obsahuje 4 bloky po 26 bitech (16 bitů informačních a 10 kontroních). Systém je dnes často používán pro přenos doplňkových informací k rozhlasovému vysílání (název vysílače, dopravní informace, počasí, přepínání hudba/řeč, ...), podporuje však všechny hlavní pagingové zprávy (tónové, numerické a alfanumerické). Velkou výhodou je poměrně jednoduchým způsobem dosažitelné vysoké pokrytí, blížící se absolutnímu pokrytí území. Systém je provozován i v České republice.



11. Družicová mobilní komunikace

Tato oblast komunikace prochází v posledních letech stádiem rychlého technického rozvoje, začíná tvořit dominantní součást družicových systémů a začleňuje se do systémů globálních telekomunikačních soustav. Jedním z hlavních cílů satelitních projektů je vytvoření podmínek pro realizaci spojení mezi účastníky na libovolném místě na Zemi pomoci mobilních stanic konstruovaných s minimálními rozměry, hmotností a spotřebou energie.

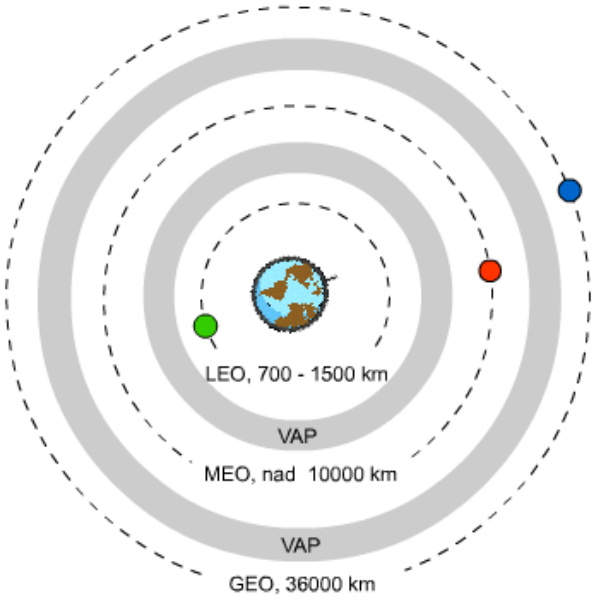
Mezi první systémy mobilní družicové komunikace, dnes nazývané systémy 1. generace, s opravdu celosvětovým dosahem patří systém **INMARSAT**. Tento systém využívá geostacionární družice na 4 pozicích a dodnes tvoří páteřní soustavu celosvětové mobilní služby v námořní, pozemní i letecké dopravě. Systém je v provozu od roku 1982, stále je používán a nadále je rozvíjen. Od roku 1992 jsou navíc původní analogové systémy doplněny o systémy digitálního přenosu.

V několika posledních letech se začíná výrazně rozvíjet nová aplikace družicových komunikací, konkrétně družicové personální komunikační služby **S-PCS** (Satellite-Personal Communications Services). Tyto systémy jsou dnes řazené k systémům druhé generace a převážně využívají nestacionárních družicových drah v různých výškách nad zemským povrchem. V roce 1998 byla takto uvedena do provozu družicová síť **IRIDIUM**, v provozu je již síť **GLOBALSTAR** a několik dalších systémů je ve stádiu příprav.

Vývoj v oblasti mobilních komunikací spěje k perspektivním univerzálním mobilním telekomunikačním systémům UMTS, jejichž součástí je také družicová složka **S-UMTS**. Tyto systémy jsou v současné době předmětem zájmu standardizačních organizací.

11.1. Oběžné dráhy družic

Satelitní komunikace využívá družice, obíhající kolem Země na třech různých oběžných drahách. Umístění družic na oběžných drahách nesmí být libovolné, neboť magnetické pole Země zachycuje elektrony a ionizované částice vyzařované sluncem a vytváří pásma s vysokou energií, takzvané Van Allenovy pásy (VAP). Magnetické pole tak chrání povrch Země před nepříznivými účinky, ale na druhou stranu přináší technická omezení pro umístění družic na oběžných drahách.



11.2. INMARSAT

INMARSAT je družicový komunikační systém, který byl nasazen do provozu v roce 1982. Z původního rozsahu služeb zaměřeného na námořní oblast (pomoc při ohrožení lodi, řízení námořní dopravy) se pole působnosti velmi rozšířilo a pokrývá nyní i pozemní a leteckou komunikaci. Původní analogové systémy jsou od roku 1992 doplněny i systémy digitálními. Systém INMARSAT používá 4 družice na geostacionárních drahách a skládá se z následujících částí.

Kosmický segment: Skládá se z družicových komunikačních transpondérů s příslušnými kmitočtovými pásmy (1,5/1,6 GHz a 4/6 GHz). Pozice jednotlivých družic jsou následující: atlantská zóna – dvě družice AOR-E a AOR-W, indická zóna IOR a pacifická zóna POR.



Pacific Ocean Region



Atlanic Ocean Region – East



Atlanic Ocean Region – West



Indian Ocean Region

Stanice MES (Mobile Earth Station): Mobilní pozemské stanice převozné (transportable) pro umístění na prostředky námořní, letecké nebo pozemní dopravy nebo přenosné (portable) pro personální komunikaci. S družicí komunikují v kmitočtovém pásmu 1,5/1,6 GHz.

Stanice LES (Land Earth station): Pevné pozemská stanice, které tvoří rozhraní vůči kosmickému segmentu. Systém nepoužívá přímou komunikaci mezi jednotlivými družicemi, spojení mobilní stanice se uskutečňuje vždy přes některou LES v dané zóně. Stanice LES pracují v pásmu 6/4 GHz a slouží zároveň pro komunikaci mezi mobilní stanicí a pozemskými telekomunikačními sítěmi.

Stanice NCS (Network Controll Station): Slouží pro celkovou koordinci sítí, pro účely kontroly a monitorování systému.

Systém INMARSAT definuje několik různých standardů, které se liší především podle nabízených služeb. Některé z nich jsou dále krátce popsány.

11.2.1. INMARSAT-A a INMARSAT-B

Služba INMARSAT-A byla realizována jako skutečně první globální družicová námořní pohyblivá služba v pásmu 1,5/1,6 GHz. Pohyblivé stanice MES se také vyskytují v modifikaci určené pro spojení ze souše. Z hlediska telefonního přenosu hlasu pracuje na analogových principech. Služba nabízí klasickou hlasovou službu a přenos dat. Vývoj kompresních algoritmů dovoluje přenášet v režimu **HSD** (High Speed Data) data rychlostí až 64 kbit/s. Váha běžného A-terminálu se pohybuje kolem 120 kg a přenosný A-terminál váží mezi 20 a 30 kg. Z důvodu přechodu na digitální způsoby přenosu je následníkem tohoto systému standard INMARSAT-B, který je používán taktéž převážně v námořních aplikacích, využití na souši je však také možné. Zde se již jedná o digitální systém, hlas je nejdříve digitalizován a poté zakódován pomocí kódování ALC (Adaptive Linear Coding), jehož výstupní tok má přenosovou rychlost 16 kbit/s. Maximální přenosová rychlost při přenosu dat je 64 kbit/s. Systém INMARSAT-B poskytuje také faksimilní služby CCITT G-3.

11.2.2. INMARSAT-C

INMARSAT-C nabízí pouze služby datových přenosů (telex, fax, data) s maximální přenosovou rychlostí 600 b/s a délkou zpráv do 32 kbyte (obousměrně). Protože metoda přenosu je založena na střadačovém principu (store and forward), může doba přenosu každé

zprávy od jejího vyslání až po její přijetí trvat pět až deset minut. Specialitou služby je propracované rozesílání zpráv více uživatelům současně (fax, data) a automatické generování aktuální polohy při nehodě či v nouzové situaci. Data jsou přenášená v malých paketech o velikosti 32 bytů. V rámci služeb INMARSAT existuje virtuální síť SafetyNET s ochranou přenášených dat proti zneužití pro pobřežní hlídky, meteorology a podobně. Druhá síť má název FleetNET a je určena pro poskytování zpráv určité skupině uživatelů prostřednictvím hromadného rozesílání aktuálních zpráv (burzovní zprávy, novinky obecně, sportovní výsledky, ...). Hmotnost přenosných C – terminálů se pohybuje okolo tří až čtyř kg, klasické terminály pro instalaci do lodních systémů dosahují hmotnosti asi 10 kg.

11.2.3. INMARSAT-D a D+

INMARSAT-D a D+ je družicový komunikační systém, který zajišťuje službu pagingu (varianta D) a obousměrného pagingu (varianta D+). Podporované zprávy jsou tónové, numerické i alfanumerické.

11.2.4. INMARSAT-M

INMARSAT-M byl uveden do provozu 1993. Jedná se o plně digitální systém, který poskytuje řízení vysílacího výkonu terminálu v závislosti na aktuálních podmínkách pro zajištění obousměrné komunikace. Komunikační kmitočty jsou shodné se systémem INMARSAT-A. Používá poměrně malé mobilní stanice MES, které umožňují přenos hovoru, dat a faxových zpráv s maximální přenosovou rychlostí 2400 b/s. Přenosná varianta (portabe) má tvar kufříku o hmotnost 5-8 kg, k dispozici jsou i větší, převozné (transportable) stanice.

11.2.5. INMARSAT-miniM

Systém INMARSAT-miniM nabízí mobilní družicovou komunikace pro hovor a přenos dat stejně jako INARSAT-M. Ke svému provozu využívá družic 3. generace, které umožňují provoz v úzkých svazcích (spotbeam) s vyšším vyzářeným výkonem na družicích. To pak umožňuje zmenšit rozměry MES zhruba na polovinu a jejich hmotnost snížit až na 2 kg. Tyto stanice potom již mohou být používány jako personální.

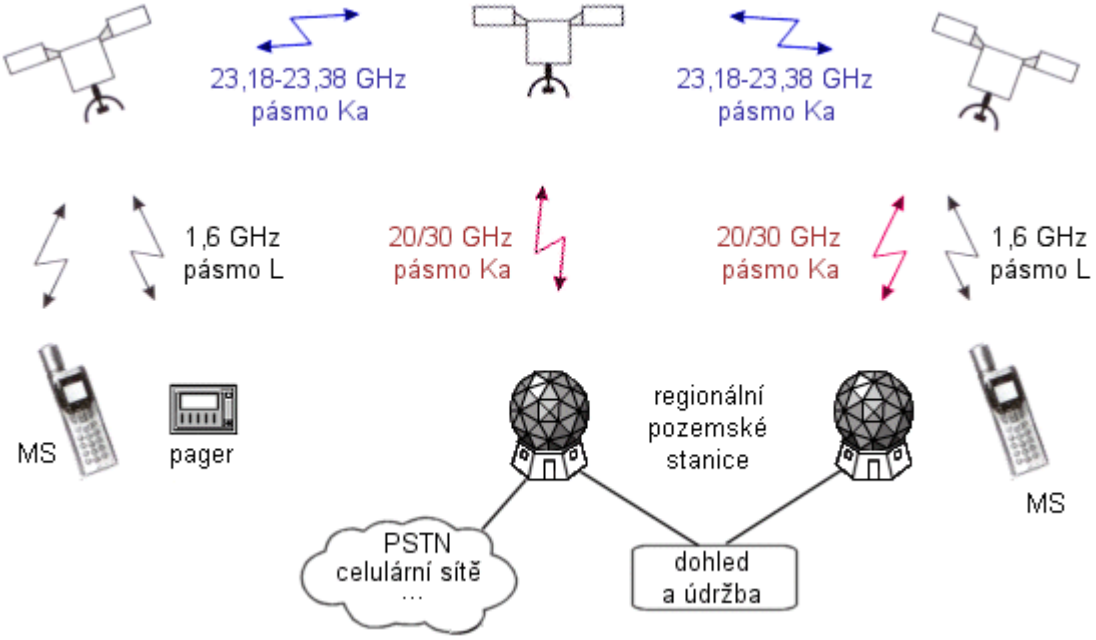
11.2.6. INMARSAT-E

Tento systém je konstruován jako elektronický námořní maják a nenabízí jiné telekomunikační služby.

11.3. IRIDIUM

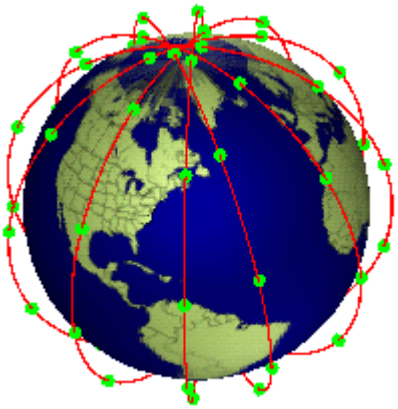
Iridium je družicový globální digitální personální komunikační systém, ve kterém uživatelé používají příruční mobilní stanice, které komunikují se soustavou 66 družic na oběžné dráze LEO. Název Iridium byl odvozen od skutečnosti, že prvek iridium obsahuje 77 valenčních elektronů, což byl počet původně plánovaných družic. V průběhu vývoje se však ukázalo, že pro zabezpečení provozu postačí 66 družic + 6 družic záložních. Původní název však zůstal v platnosti. Zajímavou vlastností systému Iridium je zpracování signálu na palubě družice (kromě kmitočtového přeložení a výkonového zesílení zde dochází i mezidružicové komunikaci. Systém zajišťuje hlasové i datové přenosy, paging a určení polohy. Za předpokladu přímé viditelnosti mezi mobilní stanicí a družicí systém zjišťuje pokrytí prakticky celého zemského povrchu. Hlavním tvůrcem tohoto systému je společnost Motorola, která měla na starost jeho vývoj. První družice byly na oběžnou dráhu vyneseny v roce 1997, projekt byl spuštěn v roce 1999. Především z přemrštěných finančních požadavků a nepříliš výrazného zájmu o poskytované služby měl ve své krátké historii tento systém řadu problémů. V současné době je sytém opět funkční a je provozován společností Iridium Satelite LLC.

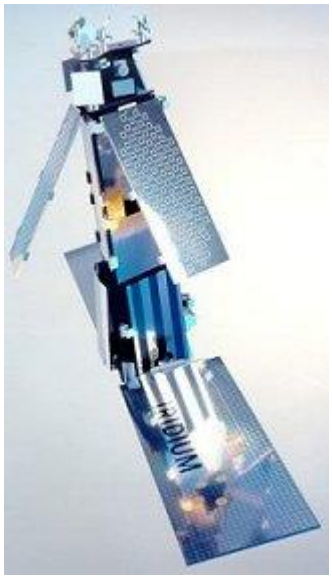
Systém se skládá z kosmického segmentu, uživatelského segmentu, regionálních pozemských stanic a řídicího segmentu.



11.3.1. Kosmický segment

Družice jsou umístěny ve výšce 780 km nad zemským povrchem na šesti orbitálních drahách v seskupení po jedenácti. Jedna váží 689 kg a kolem Země oběhne za 100 minut a 28 sekund. Každá družice Iridium má čtyři antény pro mezidružicové spoje, aby mohl být provoz směrován na družici, která je před ní a za ní na téže oběžné dráze, jakož i na sousední družice v přilehlých orbitálních rovinách.





V následující tabulce jsou kmitočtová pásma používaná systémem Iridium.

Komunikující strany	Pásmo
Mobilní stanice – družice	1616-1616,5 MHz (pásmo L)
Družice – mobilní stanice	1616-1616,5 MHz (pásmo L)
Regionální pozemská stanice – družice	29,1-29,3 GHz (pásmo Ka)
Družice – regionální pozemská stanice	19,4-19,6 GHz (pásmo Ka)
Mezidružicové spoje	23,18-23,38 GHz (pásmo Ka)

11.3.2. Uživatelský segment

Mobilní stanic v systému Iridium může být různého provedení. Především se jedná o přenosné mobilní stanice, dále existují mobilní stanice pro umístění na vozidla a specializované stanice pro použití na palubě letadla nebo lodi. Mobilní stanice umožňují hlasovou komunikaci (vokodér Motorola – přenosová rychlost 2,4 nebo 4,8 kbit/s) a přenos dat rychlostí 2,4 kbit/s. V případě přístupu k internetu v současné době Iridium nabízí využití vlastní přístupové brány a zvýšení přenosové rychlosti pomocí komprese až na 10 kbit/s. Pro využívání pagingové komponenty Iridum je samozřejmě nutností pagingový přijímač Iridium.

11.3.2.1. Regionální pozemské stanice

Tyto stanice nekomunikují s mobilními stanicemi přímo, ale přes družice. Úkolem těchto stanic je především sledování pohyby účastníka, řízení komunikace a zajištění komunikace mezi sítí Iridium a ostatními (pozemskými nebo družicovými) sítěmi.

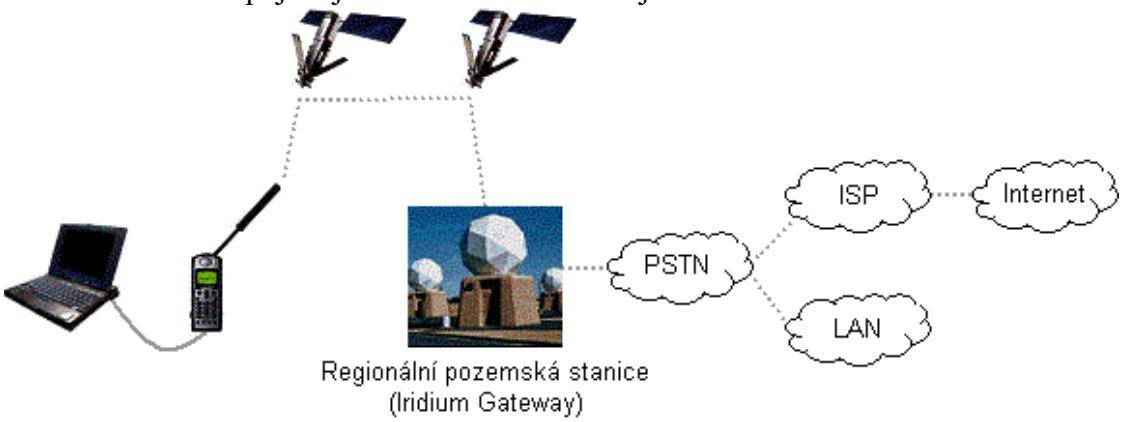
11.3.3. Řídicí segment

Řídicí centrum Iridium má na starost údržbu, detekci a opravy poruch, diagnostiku družic a veškeré činnosti spojené s administrativou a zpoplatňováním služeb.

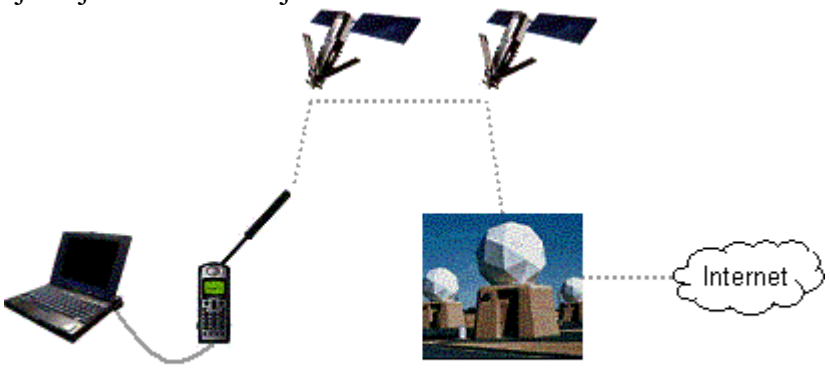
11.3.3.1. Služby systému IRIDIUM

Systém Iridium poskytuje následující telekomunikační služby:

- **Hlasová komunikace**: Zajištěn je vysoce kvalitní přenos hovorových dat a napojení na různé druhy pozemních telekomunikačních sítí. Digitalizovaný hlas je kódován a dekódován s použitím vokodéru VSELP Motorola s přenosovou rychlostí 2,4 kbit/s nebo 4,8 kbit/s.
- **Přenos dat**: V systému Iridium je umožněn přenos dat přenosovou rychlostí maximálně 2,4 kbit/s, jedná se o klasické komutované spojení s přepojováním okruhů. Proces takového datového spojení je znázorněn na následujícím obrázku.



Pro přímé připojení k Internetu dnes Iridium nabízí službu **Direct Internet**. Přístup do internetu je zajištěn přímo na straně provozovatele sítě Iridium a pomocí kompresních metod lze dosáhnout maximální přenosové rychlosti až 10 kbit/s. To je ovšem teoretické maximální přenosová rychlost, při přenosu hůře komprimovatelných dat bude jistě nižší. Služba Direct Internet je objasněna následujícím obrázkem.



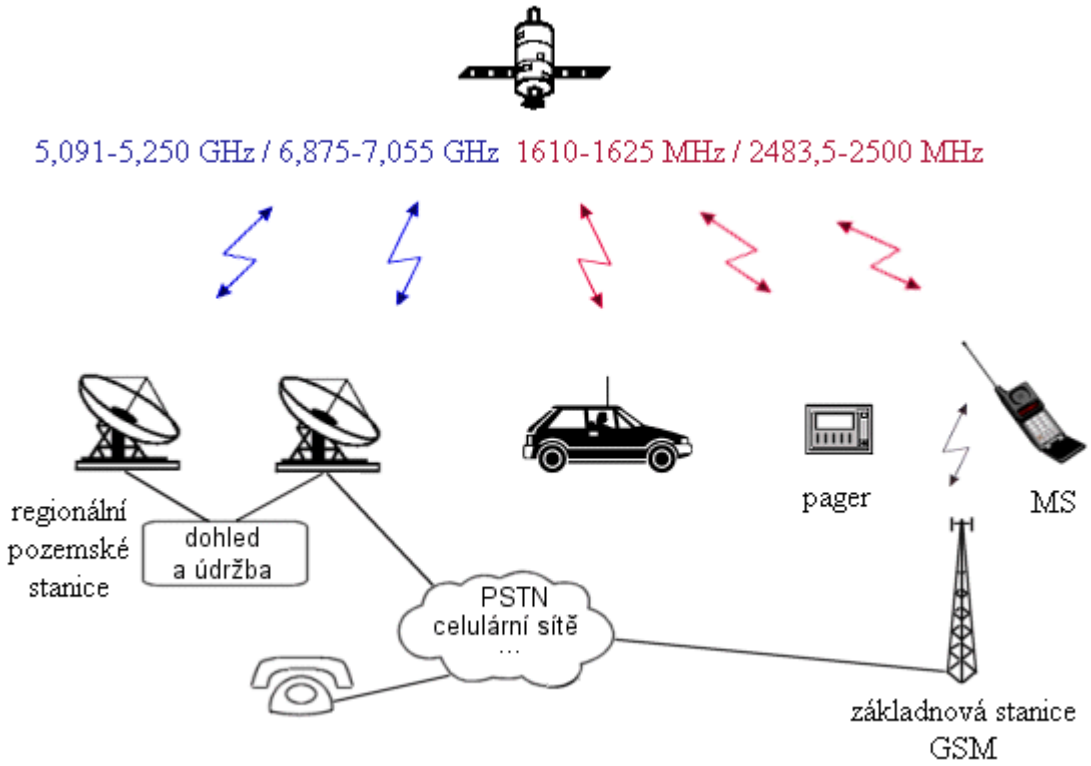
- **Paging**: Alfanumerický paging je zajišťován pro pagingové přijímače velikostí srovnatelné s klasickými zemskými pagingovými přijímači. Uživatelům mobilních telefonů systému Iridium je umožněna služba krátkých textových zpráv SMS. Zprávy mohou být odesílány mnoha způsoby včetně elektronické pošty nebo z internetového rozhraní.
- **Určení polohy**: Poloha je určována pomocí zjišťování doby časového zpoždění šířícího se signálu. Metoda dovoluje lokalizovat mobilní stanici s přesností okolo 1,5 km.

Přehled parametrů systému Iridium

Pokrytí	Globální
Typ dráhy družic	LEO
Výška družice / doba oběhu	780 km / 100 minut 28 sekund
Počet družic	66 (+ 6 záložních)
Počet rovin oběžbých drah	6
Hmotnost družice	689 kg
Životnost družice	7 let
Pásma pro komunikaci **regionální pozemská stanice – družice** vzestupná / sestupná dráha	19,4-19,6 GHz / 29,1-29,3 GHz
Pásma pro komunikaci **mobilní stanice – družice**	1616,0-1616,5 MHz
Počet bodových svazků družice	48
Mezidružicové spoje	4 na jednu družici přenosová rychlost 25 Mbit/s pásmo 23,18-23,38 GHz
Modulace	DE-QPSK, DE-BPSK
Mnohonásobný přístup	TDMA/TDD
Doba viditelnosti družice	9 minut
Umístění prvních družic	1997
Zahájení provozu	1999

11.4. GLOBALSTAR

Systém Globalstar zajišťuje družicové personální komunikační služby z jakéhokoliv bodu na povrchu Země s výjimkou polárních oblastí. Dráhy družic jsou optimalizovány tak, aby zajistily co nejvyšší dostupnost svých služeb v oblasti mezi 70 stupni jižní šířky a 70 stupni severní šířky. Systém vyvinuly především společnosti Loral a Qualcomm, dnes se na jeho podpoře podílí více společností. Stejně jako ostatní systémy družicové komunikace lze i tento rozdělit na kosmický segment, uživatelský segment a regionální pozemské stanice.



11.4.1. Kosmický segment

Globalstar používá soustavu 48 družic (+ další 4 záložní) obíhajících v osmi rovinách na oběžných drahách LEO. Družice mají hmotnost 462 kg a jsou umístěny ve výšce 1400 km nad zemským povrchem. Kromě kmitočtové konverze a zesílení výkonu se na palubě družice neprovádí žádné zpracování signálu, jednotlivé družice mezi sebou nekomunikují a veškerý provoz jde přes regionální pozemské stanice. V následující tabulce jsou uvedena kmitočtová pásma používaná systémem Globalstar.

Komunikující strany	Pásmo
Mobilní stanice – družice	1610-1625,5 MHz (pásmo L)
Družice – mobilní stanice	2483,5-2500 MHz (pásmo S)
Regionální pozemská stanice – družice	6,875-7,055 GHz (pásmo C)
Družice – regionální pozemská stanice	5,091-5,250 GHz (pásmo C)
Mezidružicové spoje	žádné

11.4.2. Uživatelský segment

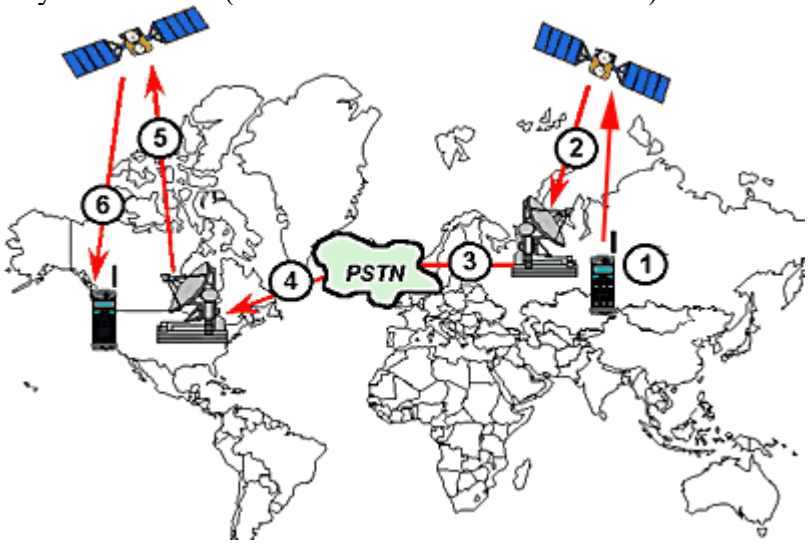
V systému Globalstar se používají tři typy uživatelských stanic:

- Přenosná ruční stanice: Velice podobná stanice jako mobilní stanice v systému GSM. Může být v provedení pouze pro systém Globalstar nebo může být dvoumódová v provedení pro Globalstar a GSM, CDMA nebo AMPS.
- Mobilní uživatelská stanice: Tato jednotka je připevněna na vozidle a skládá se z ruční jednotky vložené do adaptéru ve vozidle. Tyto stanice mají vyšší zisk antény a opět mohou existovat v dvoumódové varianty.
- Pevná uživatelská stanice: Tyto stanice mají funkce ekvivalentní mobilním uživatelským stanicím krom toho, že zisk antény a vysílaný výkon může být ještě vyšší.

Všechny uživatelské stanice systému Globalstar využívají pro identifikaci uživatelů SIM kartu. V případě dvoumódových stanic se stanice může přihlásit do lokálního celulárního systému a komunikovat za nižší cenu. Jestliže se toto přihlášení z nějakého důvodu nepodaří, stanice se přihlásí k systému Globalsatr.

11.4.3. Regionální pozemské stanice

Z důvodu absence přímé komunikace mezi jednotlivými družicemi plní tuto funkci regionální pozemské stanice, a i komunikace mezi dvěmi mobilními stanicemi tak vždy částečně probíhá přes fixní pozemskou síť. Sytém je samozřejmě navržen tak, aby byla umožněna spolupráce s veřejnými pozemními sítěmi PSTN, mobilními sítěmi PLMN a dalšími komunikačními sítěmi, což je dalším úkolem těchto pozemských stanic. Poloměr plochy regionu, který má na starost regionální pozemská stanice se mění se zeměpisnou šířkou od asi 200 km při zeměpisné šířce 15 stupňů po 5000 km pro vyšší zeměpisné šířky. Stanice tak jsou instalovány hustěji poblíž rovníku a v oblastech s vysokou intenzitou provozu. Stanice má několik (typicky 4) antén pro sledování družic, pracuje bez obsluhy a je monitorována z řídicího centra provozovatele systému SPCC (Service Provider Control Centre).



11.4.4. Služby

Systém samozřejmě umožňuje hlasovou komunikaci, kdy jsou digitalizovaná hovorová data přenášena rychlostí 2,4 kbit/s, 4,8 kbit/s nebo 9,6 kbit/s. Přenos dat je umožněn rychlostí maximálně 9,6 kbit/s. K podporovaným službám patří ještě alfanumerický paging, služba krátkých textových zpráv a určení polohy.

Přehled parametrů systému GLOBALSTAR

Pokrytí	70º severní šířky a 70º jižní šířky
Typ dráhy družic	LEO
Výška družice / doba oběhu	1400 km / 2 hodiny
Počet družic	48 (+ 4 záložní)
Počet rovin oběžbých drah	8
Hmotnost družice	426 kg
Životnost družice	7 let
Pásma pro komunikaci **regionální pozemská stanice – družice** vzestupná / sestupná dráha	5,091-5,250 GHz / 6,875-7,055 GHz
Pásma pro komunikaci **mobilní stanice – družice** vzestupná / sestupná dráha	1610-1625 / 2483,5-2500 MHz
Počet bodových svazků družice	16
Mezidružicové spoje	žádné
Modulace	QPSK, rozprostřené spektrum
Mnohonásobný přístup	CDMA
Doba viditelnosti družice	10-12 minut
Umístění prvních družic	1997
Zahájení provozu	2000

11.5. Další systémy 2. Generace

11.5.1. ICO (dříve označovaný Inmarsat P)

V globálním systému ICO je použito dvanáct družic uspořádaných do dvou orbitálních rovin skloněných vůči rovníku o 45°. Dráhy jsou typu MEO s družicemi ve výšce přibližně 10350 km. Doba oběhu družice je asi šest hodin a každá družice je vidět ze kteréhokoliv bodu na Zemi po dobu asi 20 min. Pro dosažení nepřerušené komunikace je nezbytné předávání mezi družicemi a mezi bodovými svazky družice, podobně jako v systémech IRIDIUM a GLOBAL-STAR. Komunikace družic s pozemskými stanicemi probíhá v kmitočtovém pásmu 5 GHz / 6GHz, s mobilními uživatelskými stanicemi v pásmu 1980-2010 MHz/2170-2200 MHz, použita je technologie CDMA.

11.5.2. Odyssey TRW

Tento systém je vyvíjen společností TRW Space and Electronic Group. Architektura sítě bude tvořena 12 družicemi na orbitálních drahách MEO ve výšce 10370 km. Systém používá techniku mnohonásobného přístupu CDMA a pracuje v pásmech 20GHz / 30GHz a 1610-1626,5 MHz / 2483,5-2500 MHz. Mezi poskytové služby patří hlasová komunikace a přenos dat, další služby nejsou vyloučeny.

11.5.3. TELEDISC

Síť Teledesc je tvořena 288 družicemi LEO. Družice obíhají ve výšce 1 400 km a jsou uspořádány do konstelace, která obsahuje dvanáct rovin po 24 družicích, využito je mezidružicové komunikace až osmi sousedních družic. Systém pokrývá téměř 100 % populace na zemském povrchu a je určen převážně pro přenosy dat mezi počítači a videokonferencemi. Na Zemi se počítá spíše s pevnými terminály, připojení mobilních účastníků však není vyloučeno.

V současné době již existuje nebo je ve fázi příprav mnoho dalších družicových telekomunikačních systému, pro úplnost jsou zde uvedeny alespoň názvy některých z nich:

- **Ellipso**
- **LEO One**
- **Celertri**
- **VITASAT**
- **Orbcomm**
- **Leostat**
- **Faisat**
- **Starsys**
- **ECCO**

11.6. Mapa světa a používané technologie mobilní komunikce

Po kliknutí na následující obrázek se otevře nové okno, obsahující mapu světa a přehled základních parametrů technologií pro mobilní komunikaci. U každé země je uvedeno, které z těchto technologií jsou zde používány.

12. Mobilní telefon

Mobilní telefon (hovorově mobil) je elektronické zařízení, které umožňuje uskutečňovat telefonní hovory jako normální telefon, jehož uživatel však není díky použití rádiových vln vázán na místo, kde končí telefonní přípojka.

Podle způsobu připojení do telefonní sítě rozlišujeme:

Celulární (buňkové) telefony – připojují se do celulárních rádiových sítí, které v podobě veřejných pozemních mobilních sítí (Public Land Mobile Network – PLMN) pokrývají hustěji osídlené oblasti většiny zemí; díky roamingovým smlouvám lze většinou komunikovat i ze zahraničí; překážkou však může být nekompatibilita různých systémů mobilních telefonů (GSM proti CDMA2000)

Satelitní telefony – umožňují komunikaci i z odlehlých míst, kde není pokrytí buňkovými sítěmi; protože komunikují s telekomunikační družicí (satelitem) je podmínkou použití volný výhled na oblohu

Bezšňůrové telefony – umožňují volný pohyb po bytě, domě nebo pozemku; komunikují se základnovou stanicí nebo přímo mezi sebou; základnová stanice je vlastněna účastníkem a je připojenou na běžnou pevnou telefonní linku, případně do mobilní sítě

Protože naprostá většina mobilních telefonů používá buňkovou technologii, jako GSM UMTS, CDMA, LTE (nebo starší NMT, v USA AMPS) v mnoha jazycích jsou slova mobilní telefon (anglicky mobile phone) a buňkový telefon (anglicky cellphone) synonyma. To platí i o češtině, kde se název buňkový telefon prakticky nepoužívá, a pro přenosné telefony používající jinou, než buňkovou technologii se používají názvy satelitní telefon a bezdrátový telefon.

Mobilní telefony umožňují nejen komunikaci v rámci mobilní sítě, ale i spojení s pevnou telefonní sítí přímo volbou telefonního čísla na vestavěné klávesnici a poskytují širokou škálu dalších telekomunikačních služeb, jako jsou SMS, MMS, WAP a připojení na Internet. Protože patří k nejrozšířenějším elektronickým zařízením, výrobci je vybavují také dalšími funkcemi dříve dostupnými u Personal Digital Assistant. Mobilní telefony vybavené operačním systémem mohou v mnoha ohledech nahradit osobní počítače.

Mezi největší producenty mobilů patří Nokia, Alcatel, Huawei, Samsung, Apple, Sony, BenQ Mobile (dříve Siemens), Blackberry, HTC, LG Electronics, Motorola. Někteří mobilní operátoři jako T-Mobile si nechávají vyrábět mobilní telefony, které dodávají pod vlastní značkou. Jedinými českými producenty mobilních telefonů jsou firmy Verzo a Jablotron.

Mobily jsou navrženy pro fungování v celulárních sítích a obsahují standardní sadu služeb GSM, která umožňuje telefonům různých typů a v různých zemích vzájemně komunikovat.

Před použitím telefonu je nutné zvolit si mobilního operátora (poskytovatele přenosu) a zařídit si u něj služby. Pro telefony v sítích GSM operátor vydá SIM kartu, která obsahuje unikátní účastnické a autentizační (ověřovací) parametry pro daného zákazníka. Případně operátor vloží zákazníkův identifikátor mikrotelefonu do jeho účastnické databáze, aby mikrotelefon mohl realizovat hovory do sítě. Po vložení SIM karty do telefonu jsou služby přístupné. Mobily nepodporují jen hlasové hovory, ale mohou též posílat a přijímat data a faxy, posílat a přijímat krátké zprávy (nebo „textové zprávy“, víc v článku SMS), přistupovat na WAP a poskytovat kompletní internetový přístup použitím technologií jako GPRS, EDGE, HSDPA a další. Mobily mají samozřejmě hodiny, kalkulátor a umožňují hrát různé hry.

Operátoři podporují službu „roaming“ (cestování). Ta stejnému mobilu umožňuje použití ve více zemích. Mobilní operátoři obou zemí v takovém případě musí mít dohodu o roamingu.

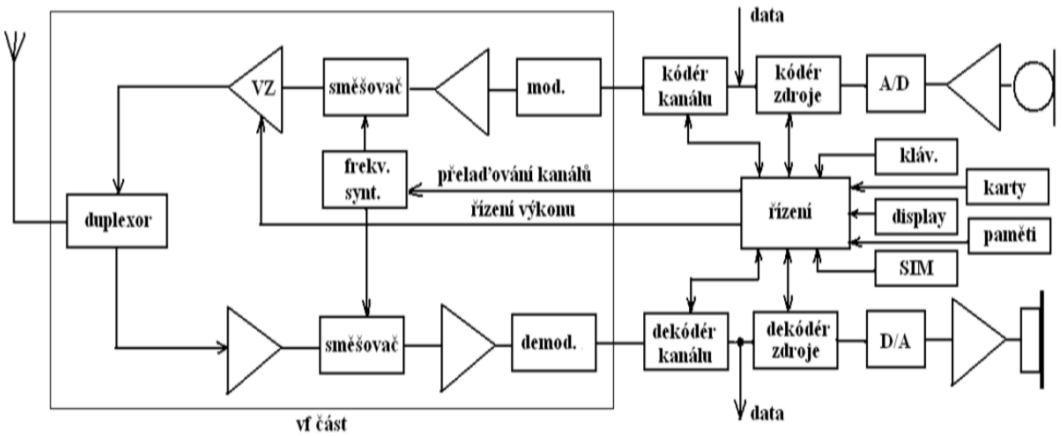
Většina nových mobilů s barevným displejem též umožňuje posílání a příjem multimediálních zpráv, takzvaně MMS. Většinou také mají zabudovaný digitální fotoaparát, což pod-

nítilo určité otázky kolem soukromí. Saúdská Arábie proto na jistou dobu zakázala prodej mobilů s fotoaparátem a povolila jej až v prosinci 2004 (i když země poutníkům na Hadždž umožňovala vzít si mobil s fotoaparátem). Jižní Korea požádala výrobce mobilů, aby zajistili, že při každém snímku fotoaparátu je slyšet charakteristický zvuk.

Přijímače GPS (lokalizační služby) se začínají objevovat integrovány nebo připojené (např. přes Bluetooth) k mobilům. Nejsou určeny jen na pomoc pracovníkům záchranné služby a službám pro odtahování vozidel; GPS v mobilu lze použít i pro osobní navigaci.

Většinu dnešních mobilních telefonů lze dnes díky možnosti přehrávání hudby ve formátech MP3 a WMA, výstupu na sluchátka, kvalitním integrovaným reproduktorům, barevnému displeji, možnosti přehrávat video a podpoře paměťových karet použít jako MP3 nebo MP4 přehrávač.

Mobilní telefony se vyrábějí v různém velikostním provedení, kdy je různým způsobem kladem důraz na tloušťku či velikost telefonu. Nejmenší mobilní telefon na světě, který byl zapsán do Guinessovy knihy rekordů vyrobila izraelská společnost Modu. Tento telefon má na výšku 7,2 cm, na šířku 3,76 cm a je tlustý 7,8 mm.



Vysílání: Mikrofon je obvykle elektretový. Digitalizace analogového hlasového signálu, kódování a modulace jsou popsány výše. Směšovač převede modulovaný signál do příslušného kanálu. Výkonový zesilovač VZ má proměnlivý výkon (max. jednotky W) podle potřeby. Frekvenční syntezátor plnící funkci místního oscilátoru se při příjmu a vysílání neustále přelaďuje.

13. Matice

Matice nad tělesem P je zobrazení $\{1,2,\ldots,n\}\times\{1,2,\ldots,m\}\to P$. Matice se obvykle označuje velkými tiskacími písmeny: A = (…). A teď česky.

13.1. Základní pojmy

Matice je zkrátka jakási tabulka o n sloupcích a m řádcích, přičemž toto značení řádků a sloupců nemusí být vždy stejné, na to pozor. V každé buňce tabulky je poté nějaké číslo nebo jiný výraz. Matice nemusí být pouze čistě číselná, i když se ze začátku dost pravděpodobně s jinými maticemi nesetkáte. Takže jak asi může vypadat matice:

$$A = \begin{pmatrix} 0 & 1 & 5 \\ 8 & 5 & 51 \end{pmatrix}$$

Tato matice má dva řádky a tři sloupce. Prvky matice se značí pomocí indexů a namísto velkého písmene se používá malé písmeno: $a_{11} = 0$ nebo $a_{23} = 51$. První index udává řádek a druhý index sloupec.

13.2. Speciální typy matic

Matice mohou mít různé vlastnosti a některé speciální matice mají dokonce své vlastní názvy.

Čtvercová matice je matice, která má stejný počet řádků jako sloupců. Pokud není matice čtvercová, je obdélníková. Příklad čtvercové matice:

$$A = \begin{pmatrix} 0 & 1 & 5 \\ 8 & 5 & 23 \\ 47 & 154 & 2 \end{pmatrix}$$

Nulová matice je matice, která má na všech pozicích nuly. $a_{ij} = 0$.

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Jednotková matice je čtvercová matice, která má na hlavní diagonále jedničky a všude jinde nuly. Hlavní diagonála je jakoby „úhlopříčka“ zleva doprava. Zkrátka jsou to čísla na souřadnicích, kde se i = j.

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Schodovitá matice je matice, která má nulové řádky na konci (nebo nemá žádné nulové řádky) a každý nenulový řádek má na začátku více nul než předchozí řádek. Toto jsou vše schodovité matice:

$$A_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \pi \\ 0 & 0 & 1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 8 \\ 0 & 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix}$$

Matice transponovaná k matici A je matice A^T, u které platí $a_{ij} = a^T_{ji}$, tj. prvek který byl v i-tém řádku a j-tém sloupci bude v transponované matici na j-tém řádku a i-tém sloupci. Zkrátka zaměníte řádky matice za sloupce.

$$\begin{pmatrix} 0 & 1 & 5 \\ 8 & 5 & 23 \\ 47 & 154 & 2 \end{pmatrix}^T = \begin{pmatrix} 0 & 8 & 47 \\ 1 & 5 & 154 \\ 5 & 23 & 2 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}^T = \begin{pmatrix} 3 & 6 \\ 4 & 7 \\ 5 & 8 \end{pmatrix}$$

Symetrická matice je čtvercová matice A, která se splňuje rovnost $A = A^T$. Prvky symetrické podle diagonály jsou stejné. Můžeme tak napsat, že $a_{ij} = a_{ji}$.

$$A = \begin{pmatrix} 9 & 3 & 4 \\ 3 & 7 & 0 \\ 4 & 0 & 2 \end{pmatrix}$$

Antisymetrická matice je skoro totéž jako symetrická matice, akorát prvky na druhé straně mají opačné znaménko: $A = -A^T$. Kvůli tomu musí být prvky na hlavní diagonále nulové, protože $a = -a = 0$.

$$A = \begin{pmatrix} 0 & -3 & -4 \\ 3 & 0 & 5 \\ 4 & -5 & 0 \end{pmatrix}$$

Diagonální matice je matice, která má nuly všude kromě hlavní diagonály. Přesněji řečeno všude jinde musí být nuly, co je na hlavní diagonále není specifikováno.

$$A_1 = \begin{pmatrix} 9 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

13.3. Základní operace s maticemi

Matice můžeme sčítat, můžeme je násobit nějakým číslem a můžeme násobit i matice navzájem.

Sčítání matic je poměrně intuitivní. Pokud jsou matice stejného typu (= stejný počet sloupců a řádků), výsledná matice bude mít na stejných pozicích součty čísel na odpovídajících pozicích v předchozích maticích. Neboli pokud sčítáme matice $A + B = C$, pak platí $a_{ij} + b_{ij} = c_{ij}$.

$$\begin{pmatrix} 0 & 1 & 5 \\ 8 & 5 & 23 \\ 47 & 154 & 2 \end{pmatrix} + \begin{pmatrix} 5 & 4 & 3 \\ 10 & 20 & 30 \\ 7 & -54 & -12 \end{pmatrix} = \begin{pmatrix} 5 & 5 & 8 \\ 18 & 25 & 53 \\ 54 & 100 & -10 \end{pmatrix}$$

Sčítání matic je zřejmě komutativní a asociativní. A + B = B + A a A + (B + C) = (A + B) + C.

Násobení matic číslem je také intuitivní. Vezmete číslo a vynásobíte s ním každý prvek matice, nic víc. $k \cdot A = k \cdot a_{ij}$.

$$5 \cdot \begin{pmatrix} 0 & 1 & 5 \\ 8 & 5 & 23 \\ 47 & 154 & 2 \end{pmatrix} = \begin{pmatrix} 0 & 5 & 25 \\ 40 & 25 & 115 \\ 235 & 770 & 10 \end{pmatrix}$$

13.3.1. Násobení matic

Násobení matic už je trochu horší záležitost, protože už není intuitivní, jak by člověk čekal. Nestačí pouze vynásobit odpovídající členy. V prvé řadě musí matice splňovat kritérium, že počet sloupců první matice musí být stejný jako počet řádků druhé matice. Zbytek může být libovolný. Nyní už můžeme nadefinovat součin (n je počet sloupců první matice):

$$(A \cdot B)_{ij} = \sum_{p=1}^{n} a_{ip} \cdot b_{pj}$$

To toho víte, co? Teď to zkusím objasnit i těm méně bystrým :-). Vezmete první řádek první matice a první sloupec druhé matice. Nyní vynásobíte první prvek s prvním prvkem a sečtete s násobkem druhého prvku s druhým prvkem a sečtete atd. Tím získáte v nové matici C prvek c_{11}. Úplně nejlepší bude příklad. Vynásobíme tyto dvě matice:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

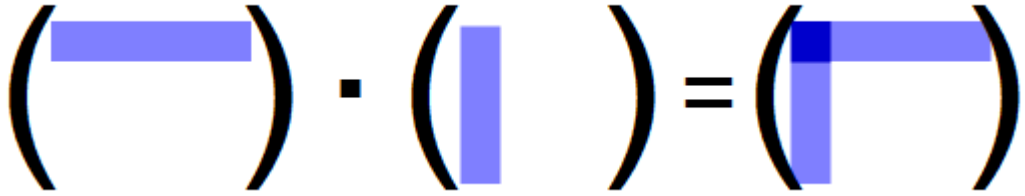
Nyní vybereme první řádek první matice a první sloupeček druhé matice:

$$A = \begin{pmatrix} \boxed{1} & \boxed{2} \\ 3 & 4 \end{pmatrix}, B = \begin{pmatrix} \boxed{5} & 6 \\ \boxed{7} & 8 \end{pmatrix}$$

Pro přehlednost zápisu si nově vzniklou matici označíme C. Pokud chceme získat první prvek této matice, musíme vypočítat toto: $c_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21}$. Upozorňuji, že na prvním místě indexu je řádek, poté sloupec. Po dosazení dostáváme: $1 \cdot 5 + 2 \cdot 7 = 19$. První prvek má hodnotu 19:

$$C = \begin{pmatrix} 19 & ? \\ ? & ? \end{pmatrix}$$

Další prvek, c_{12}, získáme stejným způsobem, pouze vezmeme první řádek a druhý sloupec. Tímto zdlouhavým výpočtem vždy získáme ten prvek, který mají společný. První řádek a první sloupec mají společný prvek na souřadnicích c_{11}, první řádek a druhý sloupec zase c_{12}. Hezky to ukazuje následující obrázek:



Teď už jen rychle donásobím zbytek matice:
$c_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22} = 1 \cdot 6 + 2 \cdot 8 = 22$
$c_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21} = 3 \cdot 5 + 4 \cdot 7 = 43$
$c_{22} = a_{21} \cdot b_{12} + a_{22} \cdot b_{22} = 3 \cdot 6 + 4 \cdot 8 = 50$
Tato čísla už jen zapíšeme do matice:

$$C = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

Správnost výsledku si můžeme ověřit třeba v Excelu nebo v OpenOffisáckém Calcu, které obsahují funkce pro práci s maticemi.

Teď pár všeobecných informací o násobení matic. Předně násobení matic není komutativní. Obecně neplatí, že by $A \cdot B = B \cdot A$, ačkoliv samozřejmě takový případ může nastat. Ale násobení matic je asociativní. Se sčítáním je dokonce distributivní: $A (B + C) = AB + AC$. Pokud násobíme dvě matice $a_{ix} \cdot b_{xn}$, pak výsledná matice bude typu $i \times n$ (bude mít tolik řádků, kolik má první matice řádků a tolik sloupců, kolik má druhá matice sloupců).

13.4. Základní maticové úpravy

Abychom mohli s maticemi efektivně pracovat, musíme si nadefinovat nějaké elementární maticové úpravy. Předně můžeme vynásobit řádek/sloupec matice nějakým číslem různým od nuly. Vynásobení se projeví stejně jako k-násobek matice, ale pouze v onom řádku/sloupci.

Druhá úprava je přičtení k-násobku j-tého řádku k i-tému řádku. Totéž pro sloupce. Zní to trochu strašidelně, ale je to ve skutečnosti jednoduché. Předvedeme si to pro k = 1. Budeme mít tuto matici:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Nyní k prvnímu řádku přičteme druhý řádek. Vezmeme tedy druhý řádek a čísla na ekvivalentních pozicích přičteme k číslům z prvního řádku. S čísly ze druhého řádku se nic nestane, změní se jen první řádek:

$$\begin{pmatrix} 5 & 7 & 9 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Nyní můžeme v úpravách pokračovat. Teď zkusíme přičíst dvojnásobek prvního řádku ke třetímu řádku. Vytáhneme z této nově vzniklé matice první řádek, vynásobíme ho dvěma, čímž získáme řádek (10, 14, 18) a tato čísla přičteme ke třetímu řádek. Opět – první řádek se nezmění, změní se pouze třetí řádek:

$$\begin{pmatrix} 5 & 7 & 9 \\ 4 & 5 & 6 \\ 17 & 22 & 27 \end{pmatrix}$$

Teď přičteme zase ke třetímu řádku součet prvního a druhého řádku. V podstatě to není nic nového, protože když ke třetímu řádku přičteme nejprve druhý řádek a poté první řádek, musíme dojít ke stejnému výsledku. Součet prvního a druhého řádku bude roven: (5, 7, 9) + (4, 5, 6) = (9, 12, 15). Součet tohoto řádku s třetím řádkem pak bude roven: (9, 12, 15) + (17, 22, 27) = (26, 34, 42).

$$\begin{pmatrix} 5 & 7 & 9 \\ 4 & 5 & 6 \\ 26 & 34 & 42 \end{pmatrix}$$

A ještě jedna sloupcová úprava (ty se obvykle nepoužívají tak často, protože nejsou tolik přehledné). Přičteme první sloupec ke druhému sloupci:

$$\begin{pmatrix} 5 & 12 & 9 \\ 4 & 9 & 6 \\ 26 & 60 & 42 \end{pmatrix}$$

A úplně naposledy vynásobíme druhý řádek dvojkou:

$$\begin{pmatrix} 5 & 12 & 9 \\ 8 & 18 & 12 \\ 26 & 60 & 42 \end{pmatrix}$$

13.4.1. Lineární závislost

Teď si ještě vysvětlíme, co jsou to závislé řádky/sloupce. Řádek je lineárně závislý, pokud tento řádek lze vyjádřit jako lineární kombinací ostatních řádků matice. Zkrátka pokud budete schopni různě sčítat řádky tak, aby vám nakonec vyšel hledaný řádek, je tento řádek lineárně závislý. Příklad:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 6 & 9 & 12 \end{pmatrix}$$

Vezmete-li dvojnásobek prvního řádku a druhý řádek, dostanete řádek třetí. Pokud od třetího řádku odečtete tuto kombinaci, dostanete nulový řádek (řádek obsahující samé nuly). Tento řádek je závislý. Úpravy tedy provedeme takto (vynásobíme první řádek dvojkou, přičteme ke druhému řádku, odečteme od třetího):

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 6 & 9 & 12 \end{pmatrix} \sim \begin{pmatrix} 2 & 4 & 6 \\ 4 & 5 & 6 \\ 6 & 9 & 12 \end{pmatrix} \sim \begin{pmatrix} 2 & 4 & 6 \\ 6 & 9 & 12 \\ 6 & 9 & 12 \end{pmatrix} \sim \begin{pmatrix} 2 & 4 & 6 \\ 6 & 9 & 12 \\ 0 & 0 & 0 \end{pmatrix}$$

Pokud máme matici A, která má n řádků a A_i označuje i-tý řádek, pak řekneme, že matice neobsahuje lineárně závislý řádek, pokud:

$$\alpha_1 A_1 + \alpha_2 A_2 + \ldots + \alpha_n A_n = 0$$

právě když $\alpha_1, \alpha_2, \ldots, \alpha_n = 0$. Nula na pravé straně rovnice představuje nulový řádek. Tj. pokud jediné řešení této rovnice je nulové řešení. Pokud nalezneme jiné řešení, pak matice obsahuje lineárně závislý řádek. Obdobně pro sloupec. Pro předchozí matici by platilo:

$$2(1\ 2\ 3) + 1(4\ 5\ 6) - 1(6\ 9\ 12) = = (6\ 9\ 12) - (6\ 9\ 12) = (0\ 0\ 0)$$

Tedy hodnoty alfa jsou $\alpha_1 = 2$, $\alpha_2 = 1$, $\alpha_3 = -1$.

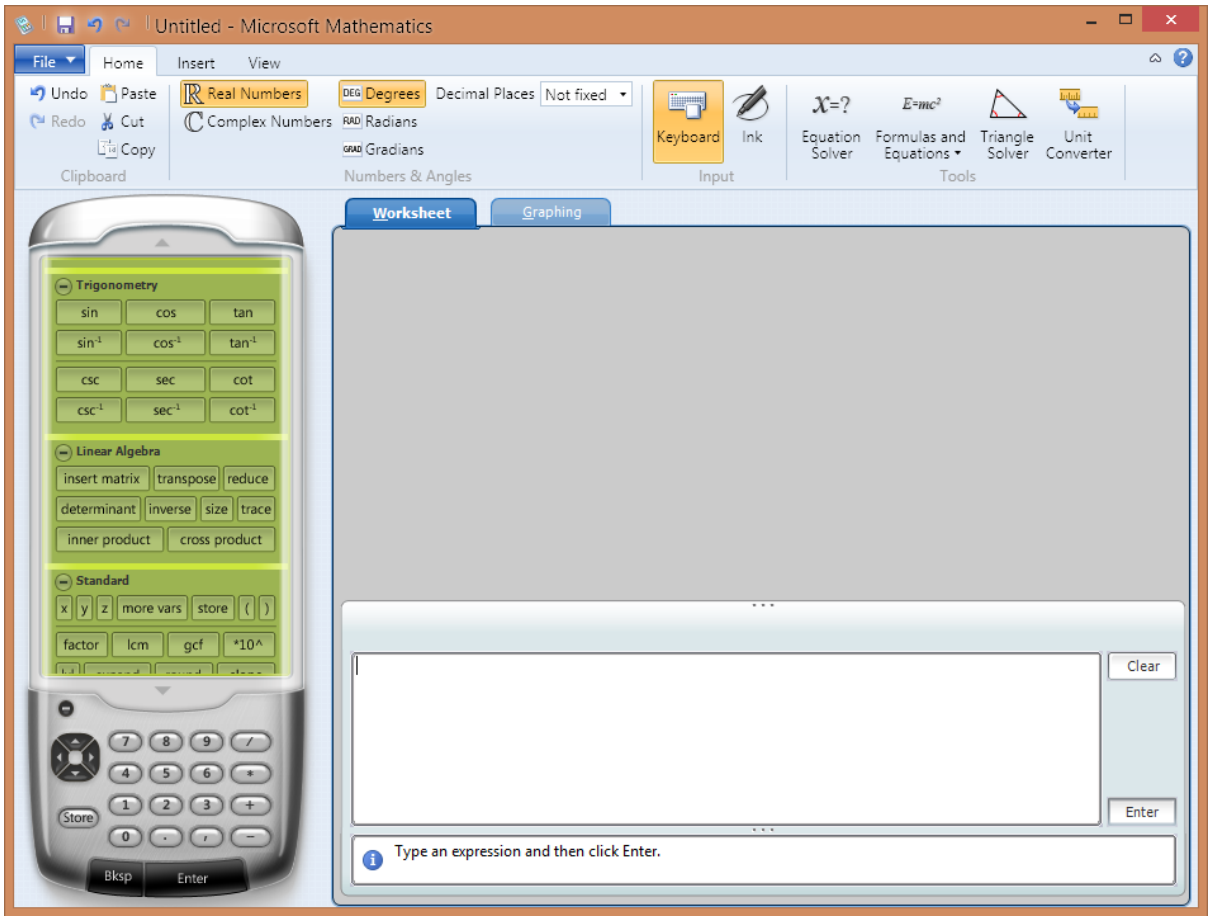
Počet nezávislých řádků nebo sloupců udává hodnost matice.

13.5. Regulární a singulární matice

Matice se nazývá regulární, jestliže má maximální hodnost (tj. pokud se v ní nevyskytuje žádný lineárně závislý řádek) a jestliže je to matice čtvercová. Čtvercová matice se nazývá singulární, jestliže není regulární (tj. jestliže matice obsahuje alespoň jeden lineárně závislý řádek). Tyhle dva pojmy jsou docela důležité, respektive často na nich stojí nějaké definice. Spousta věcí je definována, jen pokud je matice regulární.

13.6. Programy pro práci s maticemi

Práce s maticemi není nic náročného, ale krom MS Excel je možné užít MatLab, či MS Mathematics, který je bezplatný. Doporučuji použít poslední jmenovaný, který je k dostání i v 64 bitové edici a umožňuje pracovat s maticemi n*m a provádět s nimi všechny operace a to i za předpokladu, že matice obsahují proměnné. Menu programu můžeme najít na následujícím obrázku.



Pro práci s maticemi musíme rozkliknout sekci linear algebra, kde pro vložení matice nalezneme položku matrix, pak již stačí jen zadat rozměr matice. Ovládání programu je intuitivní a postupně generované mezikroky a mezivýsledky závisí na tom, jak velkou část výpočtu

zadáme po stisknutí tlačítka enter, kterým se ukončuje zadávání. Tento všestraný nástroj je velmi jednoduchý a účinný i v jiných opracích, nežli jsou matice a výhodou je, že pokud z něj něco zkopírujeme do MS Word, či jiného programu sady Office, tak je to vloženo jako položka editoru rovnic (kopírováním z tohoto programu dostaneme do programů ze sady Office i matice větší, nežli umožňuje editor rovnic). Jinak ještě co se týká matic, matici je možné zadat pomocí nabídky insert, a po vložení do Excelu není tato matic kompatibilní pro výpočet. Česká mutace není dostupná, tudíž se musíme spokojit s anglickou mutací, nicméně sám nejsem angličtinář, anglicky neumím vůbec nic, nicméně práce s tímto programem mi nečiní žádný problém, protože je opravdu intuitivní.

Práci, obzvláště složitější výpočty, je možné si ukládat přímo ve formátu tohoto SW (.gcw), ale je podporován i přímý export do Wordu, pomocí něhož je možné škálu formátů pro uložení a sdílené práce rozšířit o všechny podporované (viz kapitola o Wordu).

13.7. Matice v PC grafice

Napsání této nějaké nastiňující kapitoly o maticích mělo jeden jediný důvod a to ten, že v počítačové grafice je obraz jen maticí bodů (u rastrové) a maticí polynomů (u vektorové). Matice je nejsnazší možnost, jak popsat nějaké těleso a pracovat s ním v oblasti PC grafiky. Vytvoření obrazu na obrazovce znamená zpracovat minimálně 3 matice, přičemž jen ta poslední se opravdu fyzicky zobrazí. Pro často užívaná tělesa (písmena, číslice, základní geometrické tvary a křivky) máme v grafické kartě již předpřipravé maticové podoby. Předpřipravené maticové podoby se upravují pomocí matic a konstant vlastností, které má mít konkrétní zobrazené těleso a všechna zobrazovaná tělesa (jak vektorová, tak rasterová) musí být následně umístěna do matice bodů zobrazovací jednotky.

Už jen z nastínění principu je jasné, že úplnou matematizací PC grafiky se zobírat nemůžeme, je to totiž předmětem vysokoškolského studia a jde o samostatný studijní obor buď na katedře matematiky, nebo na katedře informatiky. My si v této kapitole ukážeme, jak vypadají jednoduché matice pro základní operace s obrazci a tělesy, aby byla jen nějaká malá představa, protože je mi naprosto jasné, že po studiu předešlých kapitolek nemůžete najednou být mistři v maticových výpočtech a ani já jím nejsem, a to s maticemi počítám několik let. Předchozí kapitolky měly jen ukázat, že matice existují, jak vypadají a jak se s nimi dělají operace, které s normálními čísli děláme někdy od druhé třídy základní školy.

Nyní tedy přejdeme k ukázkám jednotlivých matic základních úprav a poté si ukážeme nějakou konkrétní křivku, kterou upravíme jednou z těchto úprav, abychom viděli, jak to vypadá v praxi. Jen je nutné si uvědomit jeden důležitý fakti, že výpočet matice je zdlouhavý, ale protože jej potřebujeme rychle, tak se provádí se šíří slova nejčastěji 128 bit, tedy operuje se se 128 bity matice v jednom okamžiku, všechny matice jsou samozřejme pro výpočet převedeny do sledu binárních čísel. Výpočet pomocí matice je efektivní a rychlejší nežli výpočet pomocí klasických postupů. Až si matice osvojíte, tak si to zkuste někdy se soustavou rovnic, která bude mít 3 rovnice o 3 neznámých. Ale dost už povídání a pojďme rovnou na to.

Translaci, neboli posunutí ve 2D, uděláme pomocí následující matice: $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{pmatrix}$, kde Δx a Δy jsou velikosti x a y složky vektoru, kterým by se dalo popsat posunutí. Pro translaci ve 3D se pak užívá takováto matice: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x\Delta & y\Delta & z\Delta & 1 \end{pmatrix}$.

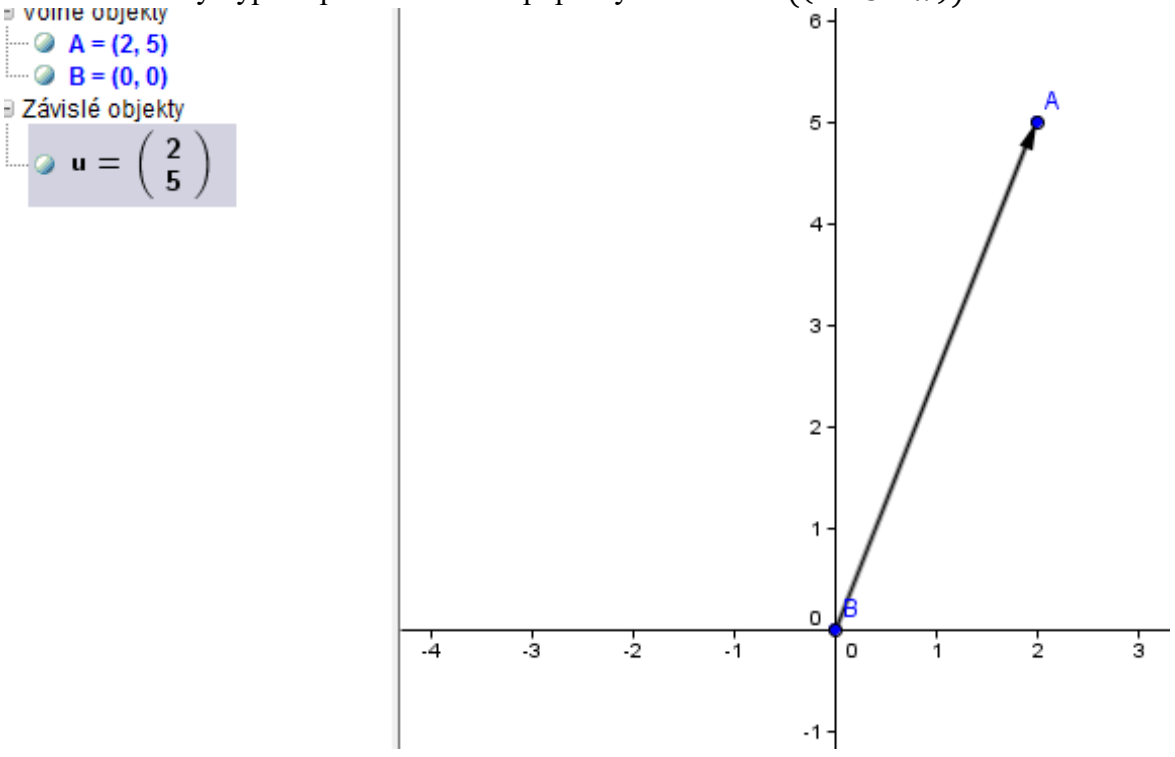
Rotaci, tedy otočení o daný úhel, není o nic těžší vymyslet. Opět nám stačí nějaké základní znalosti analytické geometrie a matic. Pro 2D grafiku by matice rotace podle souřadné

osy x vypadala následovně: $\begin{pmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$ a její implementace pro 3D grafiku: $\begin{pmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

Poslední maticí, které slouží k základním úpravám v PC grafice a ukážeme si ji, je matice škálování, tedy zvětšení či zmenšení. To, zda bude matice zvětšovat, či zmenšovat záleží na znaménku koeficientu k. zvláštností této matice je, že pokud nedodržíme, že $k_x = k_y$ (=k_z), tak dojde k úpravě tělesa v jednotlivých osách o jiné hodnoty, čímž dojde k jeho tvarové deformaci. Tuto matici jsem si nechal na závěr, protože je nejlehčí. Takže tedy 2D: $\begin{pmatrix} kx & 0 & 0 \\ 0 & ky & 0 \\ 0 & 0 & 1 \end{pmatrix}$ *a* samozřejmě i 3D varianta: $\begin{pmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$.

Pokud si vezmeme obecně nějakou 2D polynomiální křivku, která je v základu popsaná obecnou maticí $\begin{pmatrix} xn & yn \\ \dots & \dots \\ x1 & y1 \\ x0 & y0 \end{pmatrix}$, lze zapsat jako matici: (x,y). Tuto matici ro naše účely rozšíříme o pomocnou souřadnici, takže tedy dostaneme: (x,y,1) => (x,y,w). Když budeme chtít tuto křivku škálovat, tak dostaneme: $\begin{pmatrix} x & y & w \end{pmatrix} * \left(\begin{pmatrix} kx & 0 & 0 \\ 0 & ky & 0 \\ 0 & 0 & 1 \end{pmatrix} \right)$, pokud si zvolíme k = 3, tak dostaneme: $\begin{pmatrix} 3x & 0 & 0 \\ 0 & 3y & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Pokud do tohoto obecného příkladu dosadíme čísla, tak to dopadne následovně po všech úpravách, o kterých byla v této kapitole řeč.

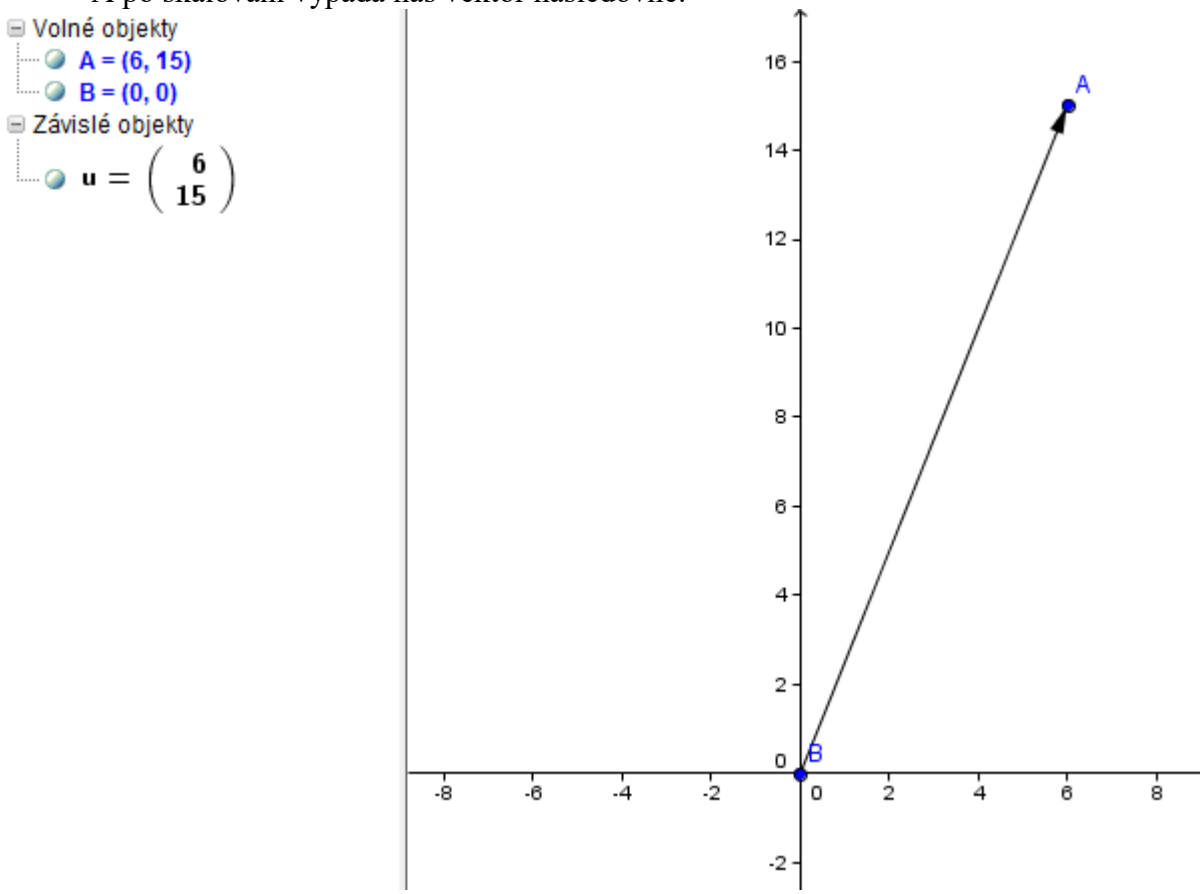
Graficky vypadá původní vektor popsaný naší maticí $\begin{pmatrix} 2 & 5 & w \end{pmatrix}$:



Nyní Provedeme škálování následujícím způsobem:

$$\begin{pmatrix}2 & 5 & w\end{pmatrix} * \left(\begin{pmatrix}3 & 0 & 0\\ 0 & 3 & 0\\ 0 & 0 & 1\end{pmatrix}\right) = \begin{pmatrix}6 & 0 & 0\\ 0 & 15 & 0\\ 0 & 0 & w\end{pmatrix} => \begin{pmatrix}6 & 15 & w\end{pmatrix}.$$

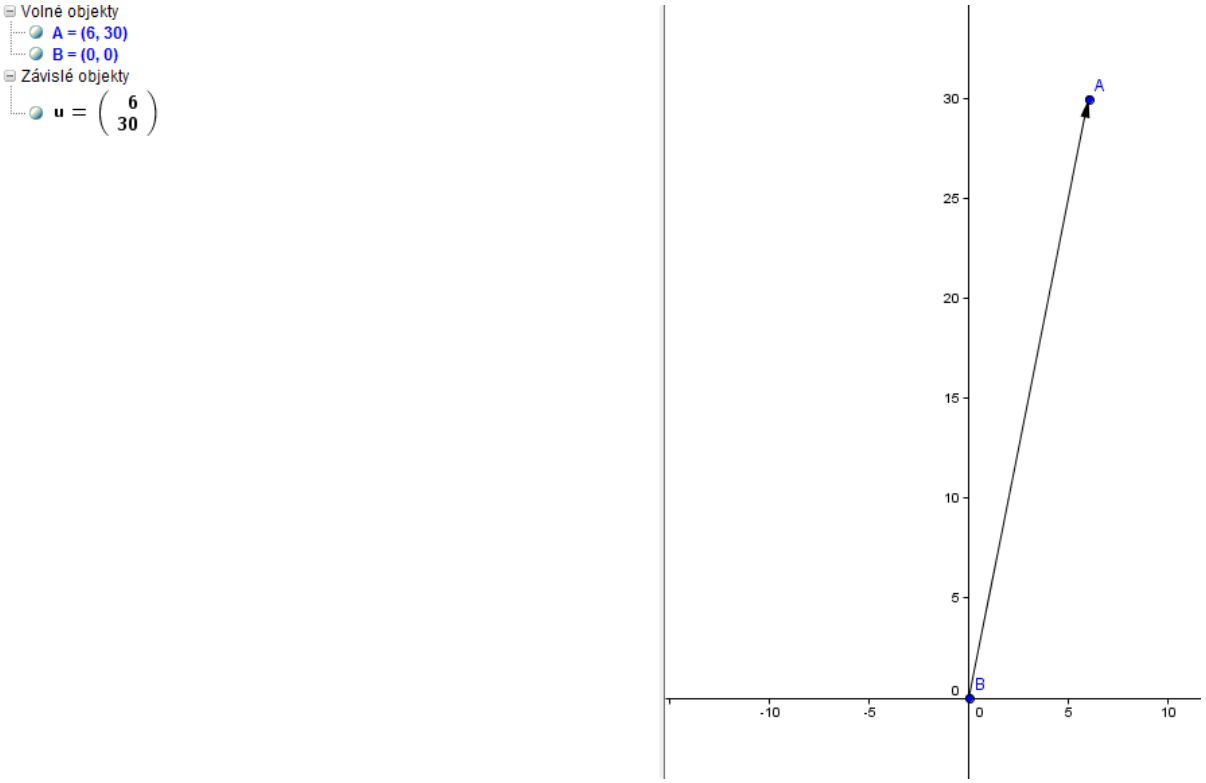
A po škálování vypadá náš vektor následovně:



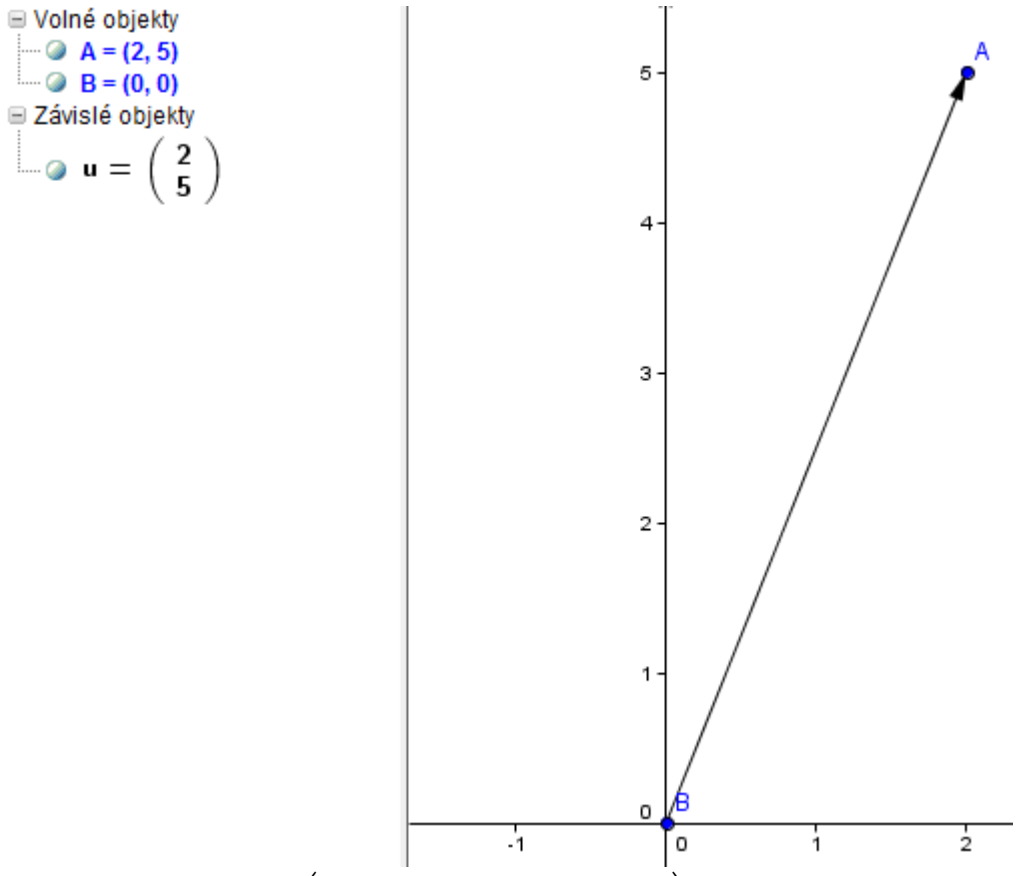
Nyní si ukažme příklad toho, jak by vypadalo, když k ≠ k, tento příklad si uděláme rovnou číselně, aby to bylo všechno dobře vidět.

$$\begin{pmatrix}2 & 5 & w\end{pmatrix}\left(\begin{pmatrix}3 & 0 & 0\\ 0 & 6 & 0\\ 0 & 0 & w\end{pmatrix}\right) = \begin{pmatrix}6 & 0 & 0\\ 0 & 30 & 0\\ 0 & 0 & w\end{pmatrix} => \begin{pmatrix}6 & 30 & w\end{pmatrix}$$

Opět si výsledek zakreslíme, abychom viděli, jak to vypadá i my, kdož mají raději pravítka.

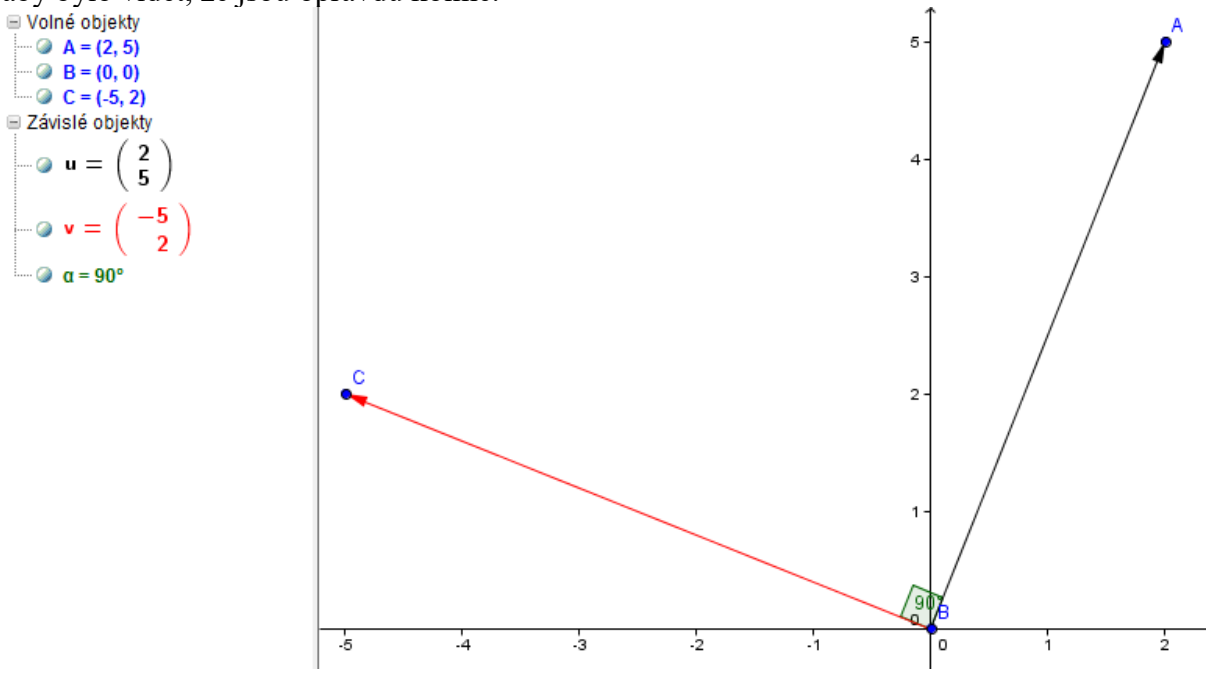


Abych vám ukázal, že matice jsem snad odvodil správně všechny, tak si poďme ukzat ještě rotaci a to opět na původním vektoru, který jsme užili v předchozím případě. Máme tedy: $(2 \quad 5 \quad w) * \begin{pmatrix} \cos(\varphi) & \sin(\varphi) & 0 \\ -\sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$, kde si úhel φ zvolíme 90°, aby to bylo z obrázků na první pohled vidět. Graficky si připomeneme základní vektor.



$$((2 \quad 5 \quad w))\begin{pmatrix} \cos(90) & \sin(90) & 0 \\ -\sin(90) & \cos(90) & 0 \\ 0 & 0 & 1 \end{pmatrix} = (-5 \quad 2 \quad w)$$

A nyní tedy o 90° otočený červeně značený vektor. Původní modrý tam zůstal úmyslně, aby bylo vidět, že jsou opravdu kolmé.



13.8. Závěrem

Tato knížka není učebnicí matematiky ani lineární algebry, abychom zde nalezli kvalitní výklad matic, který vám objasní všechna kouzla i úskalý tohototo matematického zobrazení. Řekli jsme si stručně něco málo o základech práce s maticemi, o tom jak matice vypadá, kde se používá v počítači a jak vypadají vybrané jednoduché matice užívané v počítačové grafice. Na praktických příkladeh jsme si ověřili, správnost odvození techti matic. V praxi jsou v počítačích matice dalece složitější a to ne jen díky tomu, že obsahují binární čísla, ale hlavně že jsou mnohonásobně větší a provádějí se s nimi dalece složitější operace, protože je potřeba zohlednit velké množství faktorů při tvorbě obrazu. Pokud by někoho tato problematika zajímala hlouběji, tak mohu doporučit přednášky a materiály pana Josefa Pelikána, který se počítačovou grafiou a její matematikou zaobírá na matematickofyzikální fakultě university karlovy v Praze. Jeho webové stránky jsou: http://cgg.ms.mff.cuni.cz/~pepca/ .

14. Základ počítačové grafiky

Malý úvod obstaral již konec minulé kapitoly. Jen pro pořádek, počítačová grafika je vše, co se týká zpracování grafické informace pomocí počítače. Za grafickou informaci považujeme množiny bodů, obrazce a křivky, tělesa, prostě vše, co můžeme vidět. Dnes již nelze bez grafiky provozovat počítače vůbec, pokud vám nějaký IT technik, řekn, že grafiku na jeho serveru budeme hledat marně, tak tím myslí, že jeho grafika se omezila na vykreslování textu neboli posloupnosti znaků, které jsou předdefinované. Prodký rozvoj počítačové grafiky nastal v 70. letech, kdy přišly na scénu CAD-CAM programy a počítačová podpora návrhů a výroby. Dnes PC grafika zasahuje do velké spousty oblastí našeho každodenního života, jako jsou například: *zpracování* textu, obrázků, fotografií, videa, *tvorba* animací, filmových triků, navrhování čehokoliv (CAD, Inventor, SolidWorks atd.), grafika na internetu, modelování virtuální reality, simulace reálných dějů atd.

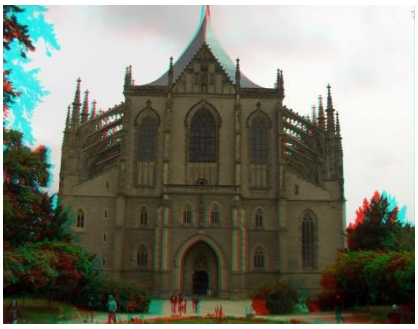
Rozlišujeme dva základní druhy grafiky: vektorovou, která objekt popisuje pomocí polynomů a rasterovou, která objekt popisuje jako množinu bodů. Jak vektorová, tak rasterová grafika existují ve 2D i ve 3D, kde tedy převažuje uložení informace ve vektorové podobě.

14.1. Rasterová grafika

Základem je mřížka (rastr) obrazových bodů – pixelů, a její velikost (šířka × výška). Při vstupu / výstupu z / na periferní zařízení (skener / tiskárna) je důležité rozlišení v dpi (kolik bodíků na palec; 1 palec je asi 2,54 cm). Jak již zaznělo v minulé větě, tak v rasterové odobě nám snímají všechna vstupní zařízení a zobrazuí vschna výstupní zařízení. S rasterovou grafikou se můžeme setkat i v televizorech, mobilních telefonech, kamerách a fotoaparátech. Příkladem tohoto typu grafiky je následující obrázek, nocméně je jich plná tato knoha.



nebo 3D příklad:



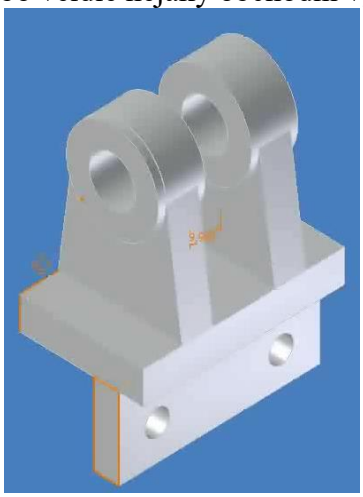
Rasterová grafika se v pořítačích běžně užívá pro ukládání videa, obrázků, zobrazují se v tomto typu obrázky umístěné na internetu a tento typ je jejiný, který je podporován v sadě Office a rogramu malování, ač se to na první pohled nemusí zdát.

Výhodou je přirozené pořízení a zobrazení na rastrových zařízeních (skener, digitální fotoaparát, monitor, tiskárna), lepší možnosti práce s barvou, přirozrná perspektiva v zobrazení a celková věrnost předloze.

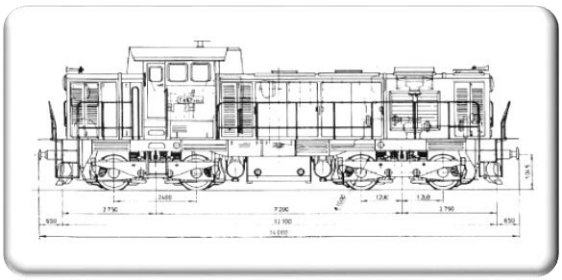
Za nevýhody lze považovat často velký nárůst objemu dat, ztráta grafické informace při transformacích, problémový převod na vektorový popis (tzv. trasování), „rozsýpání“ obrazu při přílišném zvětšení (je vidět jednotlivé body, což se týká opravdu zvětší v řádech desetitisíců, dříve nás omezí raster zobrazovací jednotky, podu jde o kvalitní digitální fotografii).

14.2. Vektorová grafika

Základním prvkem jsou objekty (úsečka, křivka, obdélník), které lze matematicky popsat (polynomy n. řádu) a těmto objektům se přiřazují různé atributy – tloušťka a barva čáry, vzhled výplně apod. Tento typ grafiky je rozšířený hlavně v průmyslu a počítačových hrách. Veške modelování výrobků a řízení strojů probíhá právě pomocí vektorové grafiky. Naprosto typickým příkladem může být nějaká takováto úchytka, kterou jsem kreslil před lety v Inventoru, nebo veldle nějaký obchodní výkres lokomotivy.

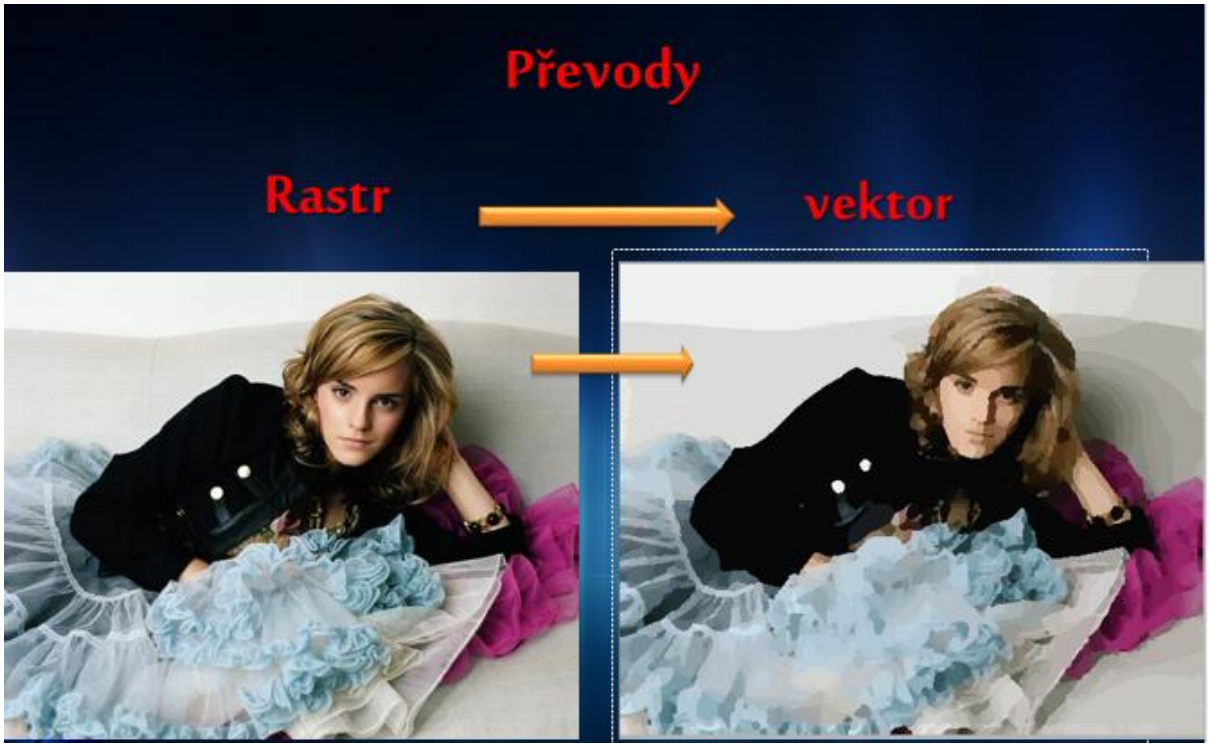


za 2D třeba výkres:



Výhodou je nezávislost na výstupním zařízení, bezproblémové transformace, snadný převod do rastru (prostě se spočítají body křivky popsané olynomem), možnost kombinace s bitmapou, neomezená manipulace s rozměry, malé prostorové nároky na paměť pro trvalé uložení.

Naopak nevýhodou je horší možnosti práce s barvou a nutnost převodu do rasterové podoby, aby bylo možné zobrazení. Další nepříznivou záležitostí je vysoká cena kvalitního SW pro tvorbu a zpracování, která i více jak 10 krát převyšuje cenu nejlepších rasterových editorů.

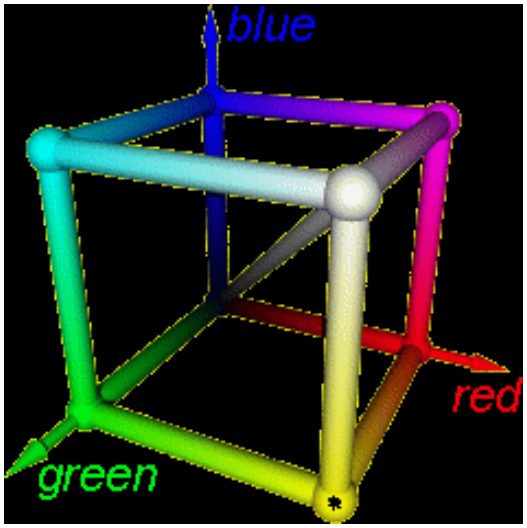


14.3. Základní barevné systémy (modely)

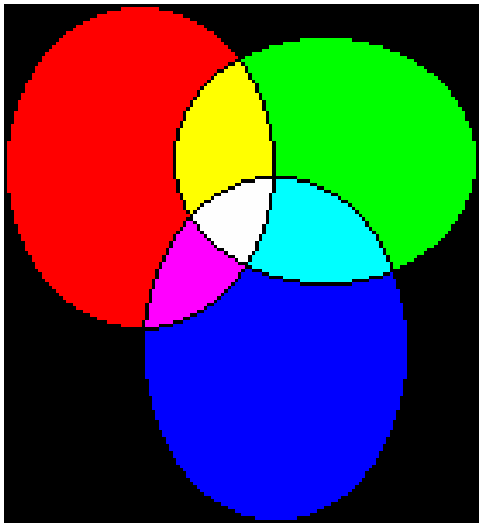
14.3.1. RGB

Barvy se skládají ze tří základních složek: červené (Red), zelené (Green) a modré (Blue). Jednotlivé složky vyzařují světlo své barvy a tím se také řídí jejich míchání. Vzniká tak trojrozměrný prostor barev reprezentovaný jednotkovou krychlí s vrcholy: (0,0,0) ...černá, (1,1,1) ...bílá, (1,0,0) ...červená, (0,1,0) ...zelená, (0,0,1) ...modrá, (1,1,0) ...žlutá, (1,0,1) ...fialová, (0,1,1) ...tyrkysová. Na diagonále mezi černou a bílou jsou tzv. odstíny šedi (stejné zastoupení všech složek).

Systém se používá při zobrazování na monitoru, fotografování a natáčení. Systém RGB je nejrozšířenějším systémem barev. S rozvojem počítačové grafiky došlo k rozšíření na RGBA – složka A (Alfa-kanál), která určuje průhlednost vzhledem k pozadí (0 - neprůhledný, 1 - zcela průhledný). Toto rozšíření se prorazilo více na světlo s motivy Aero od Windows Vista.

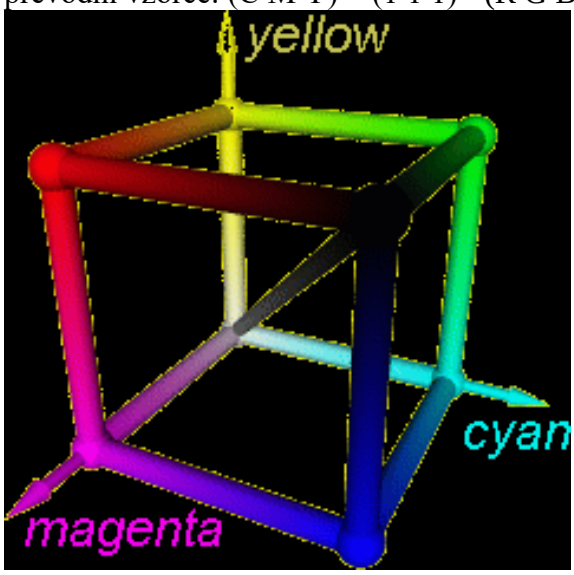


nebo:

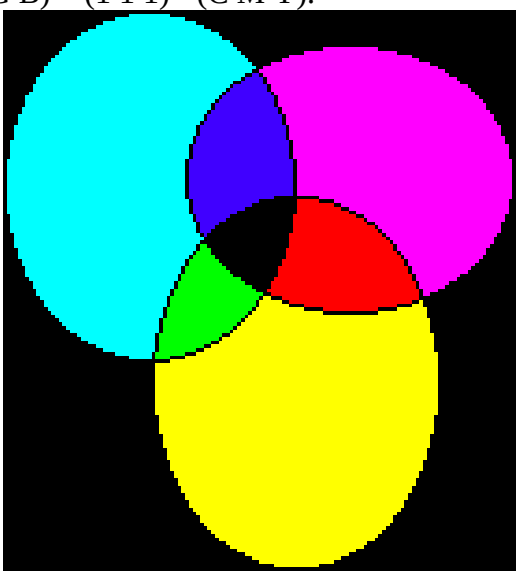


14.3.2. CMY (resp. CMYK)

Jde o doplňkový k systému RGB, který vychází z principu míchání malířských barev (světlo je pohlcováno). Opět má tři základní složky (pigmenty): tyrkys (Cyan, azurová), purpur (Magenta, fialová) a žluť (Yellow) a podobnou krychli barev. Tohoto sytému se užívá při barevném tisku. Smíchání všech tří pigmentů dá černou pouze teoreticky (prakticky vznikne tmavě hnědá), proto bývá přidávána kvalitní čerň (blacK) a systém se pak označuje CMYK
převodní vzorce: (C M Y) = (1 1 1) - (R G B), (R G B) = (1 1 1) - (C M Y).

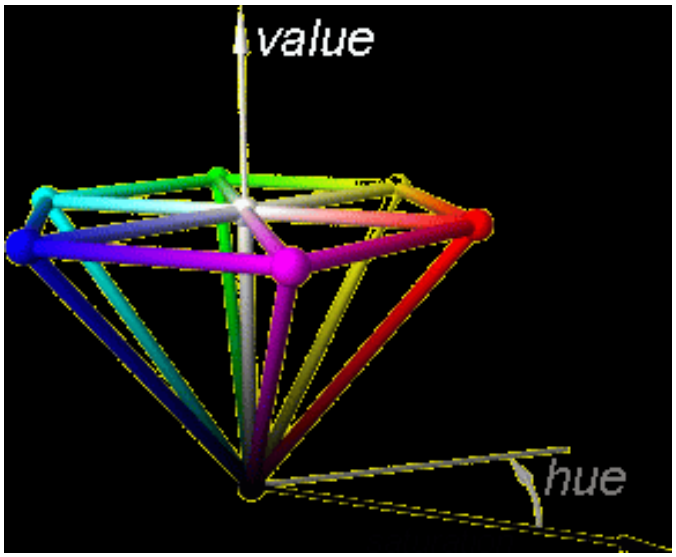


nebo:



14.3.3. HSV

Barva je v tomto systému určena třemi parametry: barevný tón (Hue, převládající barva), sytost (Saturation, příměs jiných barev) a jasová hodnota (Value, množství bílého světla). Systém je modelován pravidelným šestibokým jehlanem a není příliš rozšířen, nicméně jde o poslední ze základních možností tvorby barvy. Užívá se v jemném míchání barev v grafických programech.



14.4. Barevná hloubka

Udává počet zobrazitelných barev na daném zařízení a vztahuje se nejčastěji k základnímu modelu RGB. Standardem je dnes 24 - bitová grafika, tj. pro každý pixel je barevná informace uložena ve 24 bitech, tedy na každou složku R, G a B tak připadá 8 bitů. 1 bit může nabývat pouze 2 hodnoty - 0 a 1; do 8 bitů se tak vejde 28=256 hodnot – od 0 do 255, tedy pro všechny tři složky se tak dostane 256×256×256=224=16 777 216 možností, tj. asi 16,7 mil. zobrazitelných barev (tzv. plné barvy, angl. True Color). Podobně se dříve používala 16 - bitová grafika, tj. 216=65 536 možných barev (tzv. High Color).

14.5. Grafické formáty

Grafické formáty definují způsob popisu a uložení grafických dat. Těchto formátů existuje mnoho druhů a typů, které jsou specifické pro různé aplikace a platformy (operační systémy). Teprve časem se některé z nich staly všeobecně uznávanými. Větší rozměry, rozlišení, počet barev předlohy → větší objem dat → nutnost jejich komprese. Kompresní (komprimační) algoritmy máme bezztrátové – nedochází ke ztrátě dat, nebo ztrátové - záměrná ztráta dat (vyšší úspora místa). Kompresní poměr je původní objem dat ku komprimovanému, tedy většinou ve tvaru Něco:1 např.: 20:1.

14.5.1. JPG

Tento formát byl vyvinut skupinou skupina Joint Photographic Experts Group. Jde o ztrátovou kompresy obrazu, která se zároveň stala nepsaným standartem pro web, fotoaparáty, mobilní telefony a zpracování obzázků v PC. Obraz se komprimuje po blocích (dlaždicích) 8x8 pixelů a vlastní komprese má následující postup.

1. Převedení barev R, G, B na Y, R, B (bez ztráty kvality);
2. Podvzorkování barvonosných složek z Y:R:B 4:2:2 na 4:2:0. Je to téměř zanedbatelná ztráta, protože oko je mnohem citlivější na jasovou složku.
3. DCT (Discrete Cosine Transform) – diskrétní kosinová transformace; provádí se v blocích 8 x 8 hodnot, jde o bezeztrátový převod hodnot Y, R, B na frekvenci a úroveň, vede k získání velkého množství nulových hodnot.
4. Kvantování (zaokrouhlování) výsledků DCT; přináší zdaleka největší ztrátovost.
5. Kódování: různé algoritmy, např. Huffmanův. Princip je velmi jednoduchý: znakům vyskytujícím se s větší četností se přiřadí kód s malým počtem bitů a naopak. Tak se sníží množství dat.

14.5.2. BMP

Celým jménem Microsoft Windows Bitmap. Bitová mapa každému pixelu přiřazuje barvu pomocí RGB složek (pixely jsou seřazeny po řádcích odshora a zleva). Bez ohledu na obsah obrázku je velikost souboru vždy stejná (výška × šířka × počet bitů na pixel + hlavička v niž je uvedena velikost obrázku, způsob komprese a další údaje). Většinou není obraz nijak komprimován, případně lze použít metodu RLE (ovšem jen pro omezenou 8-bitovou paletu barev).

14.5.3. GIF

Jde o Graphics Interchange Format od firmy CompuServe. Obrázky používající paletu max. 256 barev a kompresi metodou LZW. Tento fármát umožňuje uložit více obrázků (i s různými paletami) do jednoho souboru › tak lze vytvářet tzv. animované gify (podporuje i cyklickou animaci). Mimo jiné lze nastavit prokládání řádků (interlaced) - jednotlivé řádky obrazu nejsou ukládány za sebou, ale následujícím způsobem:

1. každý osmý řádek počínaje nultým
2. každý čtvrtý řádek počínaje čtvrtým
3. každý čtvrtý řádek počínaje druhým
4. každý druhý řádek počínaje prvním

Pro jednu barvu z palety je možno nastavit průhlednost (tzv. alfa-kanál).

14.5.4. PNG

PNG neboli Portable Network Graphics je v v jistém směru nástupce GIFu. Jde o formát především pro přenos kvalitních obrázků sítí (velá kvalita obrazu a malá velikost (viz program výstřižky integrovaný ve Windows od verze Win7). Využívá bezztrátové komprese metodou LZW a dvourozměrné prokládání v sedmi krocích – pro řádky i sloupce. Umožňuje pouze jediný obrázek v souboru, nelze tedy užít pro animace na rozdíl od GIFu. Podporovaná barevná hloubka až 32 bitů (3 x 8 bitů na RGB složky, 8 bitů na alfa-kanál průhlednosti.

14.5.5. RAW

Jde o formát používaný v profesionálních fotoaparátech a prémiových telefonech zaměřených na fotografování jako je Nokia Lumia 1020. Tento formát neukrývá obraz přímo, jde o tzv. digitální negativ, který je možné prohlížet jen profesionálními editory fotografií, programy přiloženými k fotoaparátům, které mají tento formát jako výchozí a nově lze na Windows 10 tento formát prohlížet jako běžnou fotografii. Tento formát ukrývá veškeré informace přímo z čipu, tedy informace o míře osvitu jednotlivých bodů čipu všech čipů a užitých barevných filtrech, informace o korekci vad zobrazení použitého objektivu a nastavené hodnotě citlivosti. Tyto informace jsou všechny bezztrátově zkomprimovány do jednoho souboru představujícího obrázek ve formátu RAW. Výsledný obrázek je vytvořen v až v počítači, což umožňuje použít aktualizované opravy vad zobrazení případně manuálně provést tyto opravy, aniž by byly tyto aktualizace nahrány do fotoaparátu.

Tento formát má velkou řadu zajímavých vlastností, ale protože s ním nepracujeme a pracovat nebudeme, tak se o něm již více rozepisovat nebudu. Ale dalo by se říct, že je to takové Ferrari mezi formáty digitálních fotografií.

14.5.6. Vektorové formáty

Zde si uvedeme jen pár příkladů: formát DXF (CAD-CAM SW), dwg (AutoCad), ipt (Inventor), nebo Flash (ten ovšem umožňuje i práci se zvukem, hypertextem atd.), POV-Ray, písma a další. Ve vektorových formátech není tolik rozdílů v kompresy a podobně, ale jsou v nich rozdíli na úrpvni metadat, teré uchovávají potřebné matice polynomů.

14.6. Editory

Editorů je celá řada a vycházejí v hodně krádkých intervalech nové verze. Kromě her jde snad o nejrychleji se rozvíjející SW.

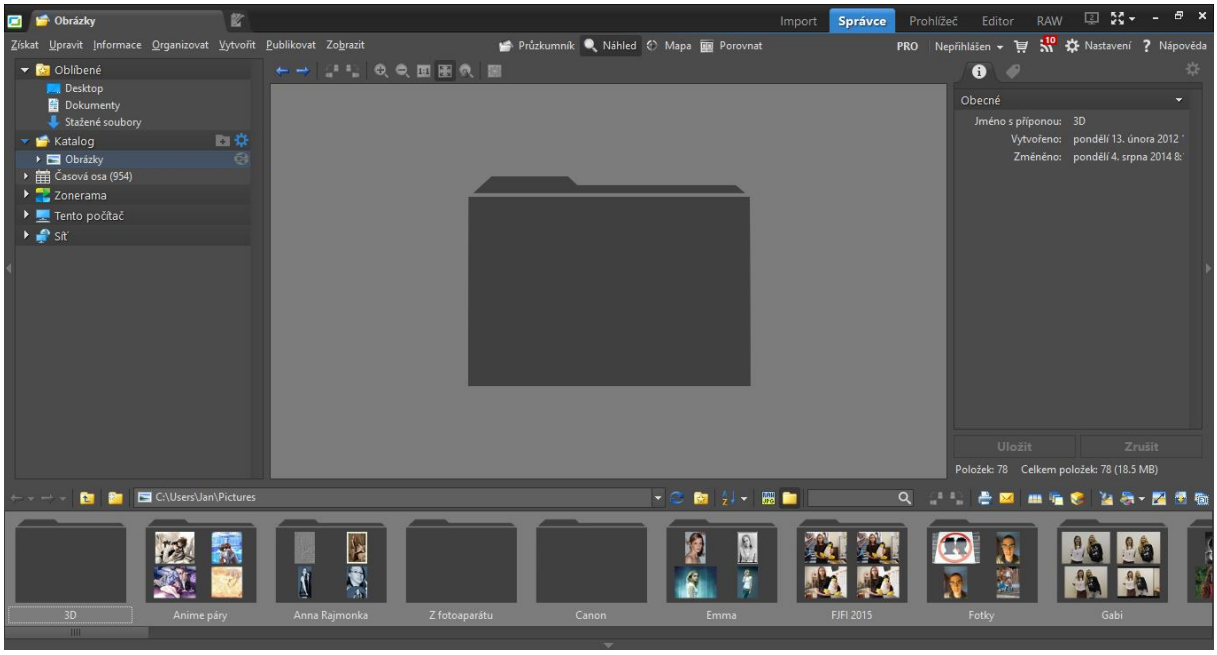
Za rasterové mohu jmenovat Zoner Photostudio, Adobe Photoshop a gimp. Tato trojice je nejrozšířenější v naší republice, nicméně nesmím zapomenout na jednoduché a universální malování integrovanédo Windows a dost oblíbené mezi studenty hlavně technických oborů.

Za vektorvé je možné jmenovat progamy společnosti Autodesk (Inventor, Auto CAD, Maya, 3ds max, atd.), kterou jsou dnes světovou elitou a následuje je npařikla SolidWorks, který je vhodný především pro odlitky. Patří sem ale třeba programy jako je Corel drowe, a dalšíprogramy pro tvorbu animovaného fimlu, technické dokumentace a simulací.

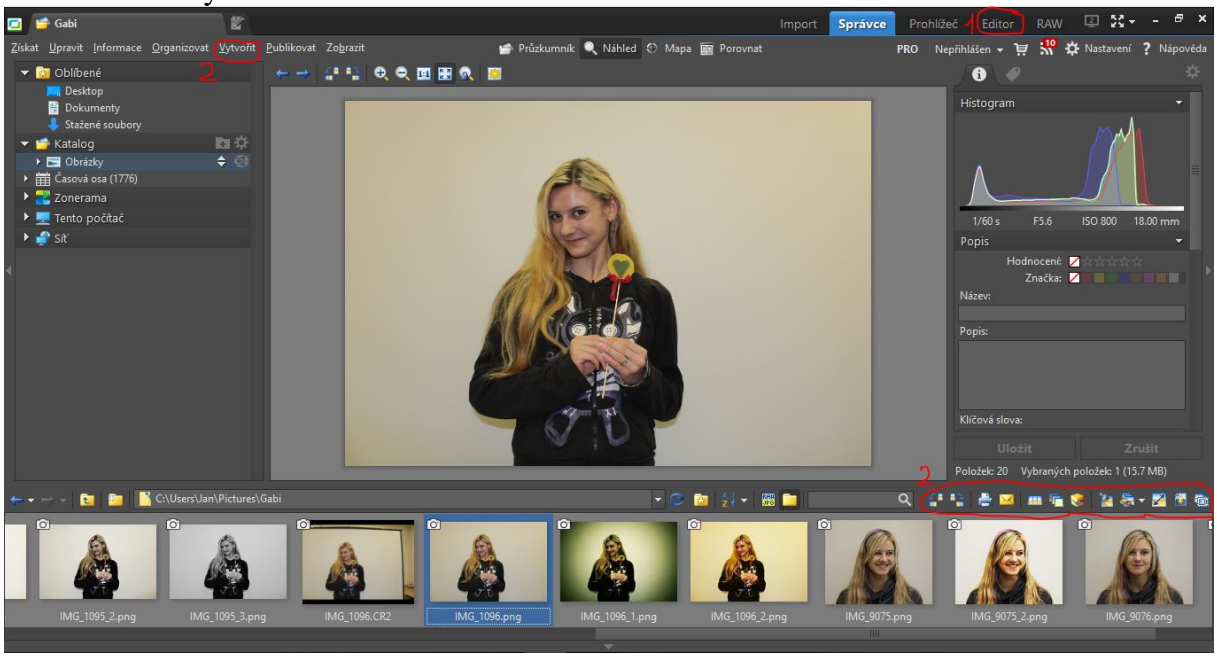
14.6.1. Zoner Photostudio

V této části si uděláme lehkou rekapitulaci toho, jak pracovat se Zonerem, co se kde nachází a jak dosáhnout kterého efektu či úpravy. Nejedná se o podrobnou knihu specializovanou na Zoner (tu mohu doporučit: PECINOVSKÝ, Josef. *333 tipů a triků pro Zoner Photo Studio*. Brno: Computer Press, 2013. ISBN 978-80-251-3572-3.), ale o popis základních funkcí a orientace v programu.

Základní obrazovka programu, pokud jej spouštíme přímo ze zástupce, nikoliv otevřením konkrétního snímku:

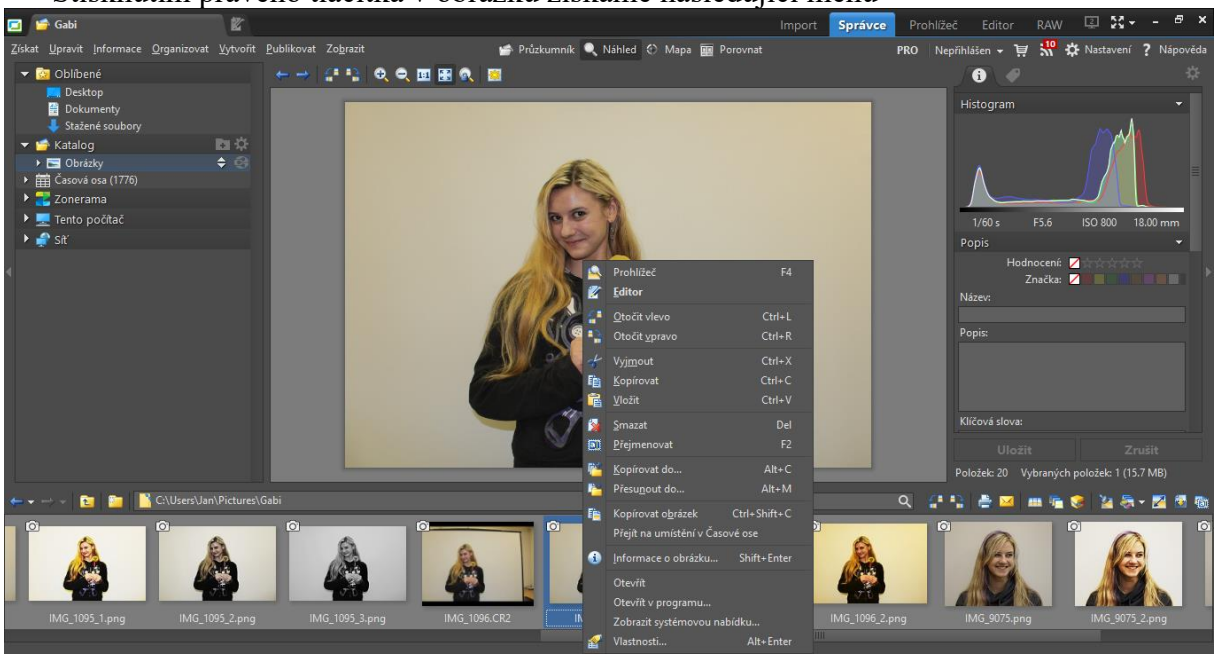


Na této obrazovce se nám ve spodním pásu zobrazí všechny složky, které jsou obsaženy ve výchozí knihovně systém, tedy ve složce Obrázky. Pro nás je důležité otevřít si konkrétní složku s obrázky a zvolit konkrétní obrázek.



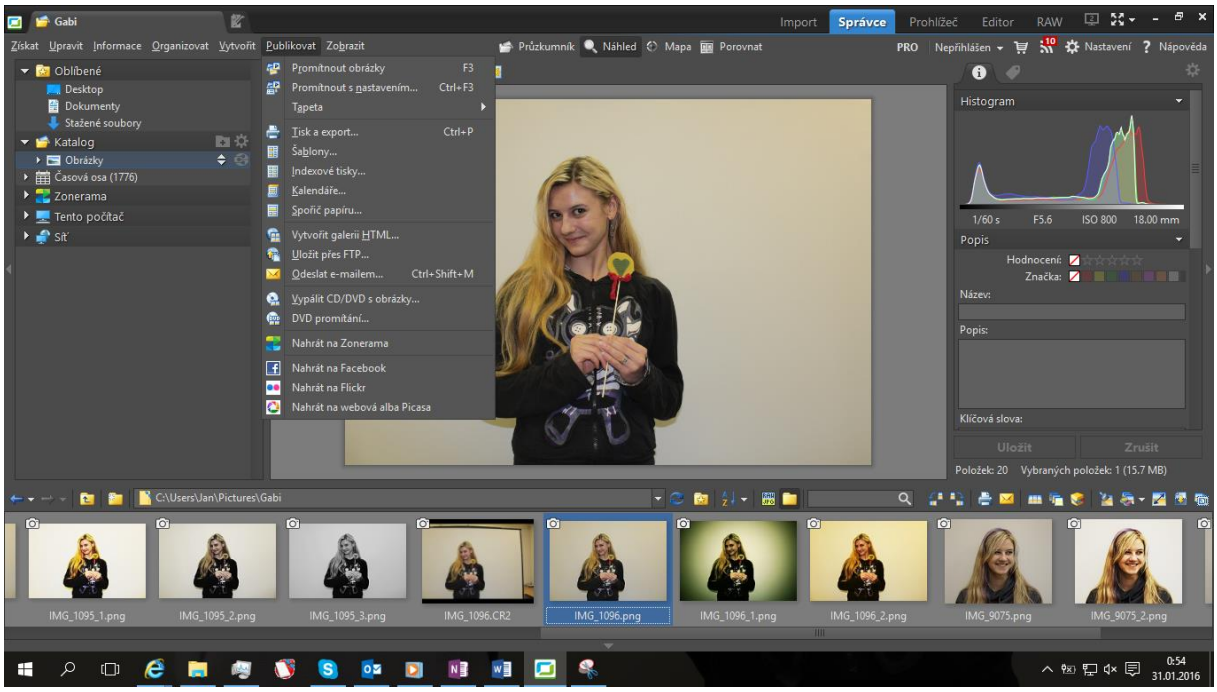
1. Je přepnutí do editoru pro většinu úprav
2. Jsou základní rychlé úpravy, jako je otočení, převedení do jiného grafického formátu, tisk, sestavení panoramatu, sestavení 3D snímku i HDR prolínání expozic, pokud možnost vytvořit, tak můžeme obrázek (či výběr více obrázků) uložit do pdf, v dolní nabídce je navíc změna velikosti obrázku.

Stisknutím pravého tlačítka v obrázku získáme následující menu



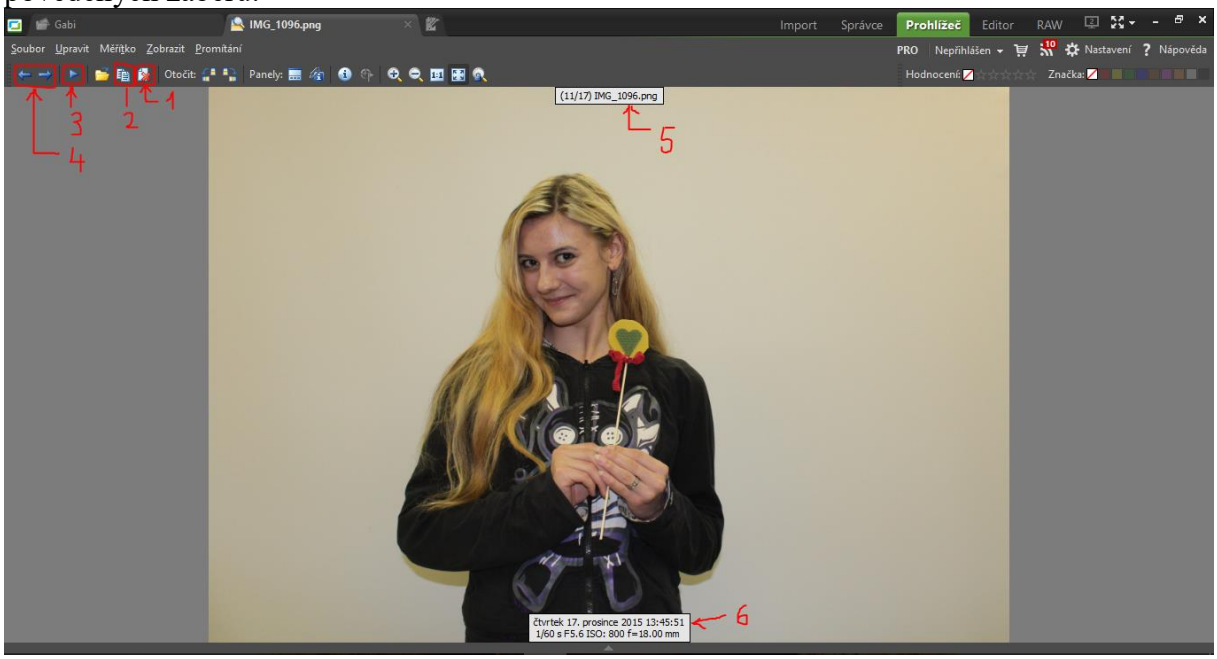
Nabídku pravého tlačítka využijeme především pro vytvoření duplikátu snímku, přesunu snímku do jiné složky, zobrazení podrobnějších informací o snímku, jeho smazání (zde je možné provést i pomocí klávesy Delete) a operacích se schránkou jako je kopírovat, vyjmout a vložit.

V aktuálním okně je možné provádět i skupinové úpravy více fotografií, což se hodí zejména pro změnu formátu souboru, či změnu rozměrů na jednotnou velikost třeba za účelem přípravy tisku. Další šikovnou nabídku nalezneme v levé horní části jako předposlední, jde o nabídku Publikovat.



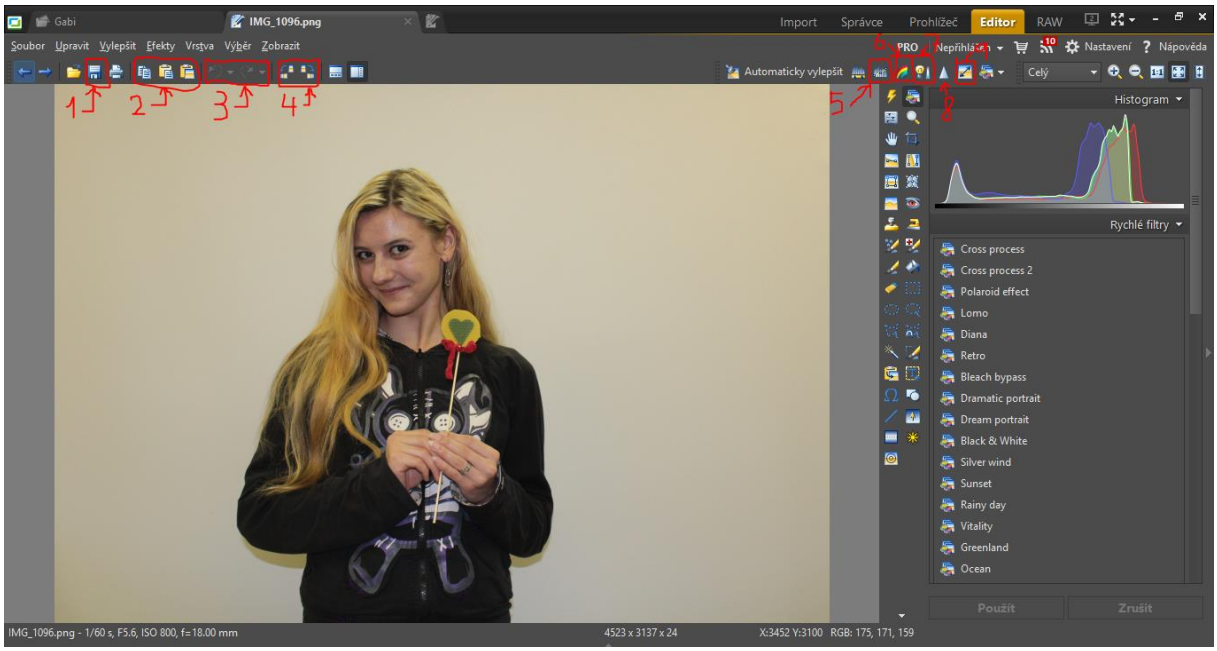
S touto nabídkou je vhodné pracovat vždy při vybrání většího množství fotografií ze složky, případně všech fotografií uložených ve složce. K podrobné práci s touto nabídkou se vrátíme později.

Nabídka prohlížeč je vhodná pro rychlé prohlédnutí, roztřízení a smazání opravdu nepovedených záběrů.



1. Je příkaz pro smazání aktuálně zobrazeného snímku
2. Provede zkopírování snímku, tedy uloží obrázek do schránky
3. Spustí prezentaci fotografií z dané složky, která běží ve smyčce do jejího ukončení
4. Přejde na předchozí (doleva) nebo následující (doprava) snímek
5. Zobrazení kolikátý snímek z kolika v dané složce to je a jeho název
6. Informace o snímku jako je datum a čas, doba expozice, clonové číslo, citlivost a ohnisková vzdálenost

Nabídka editor:

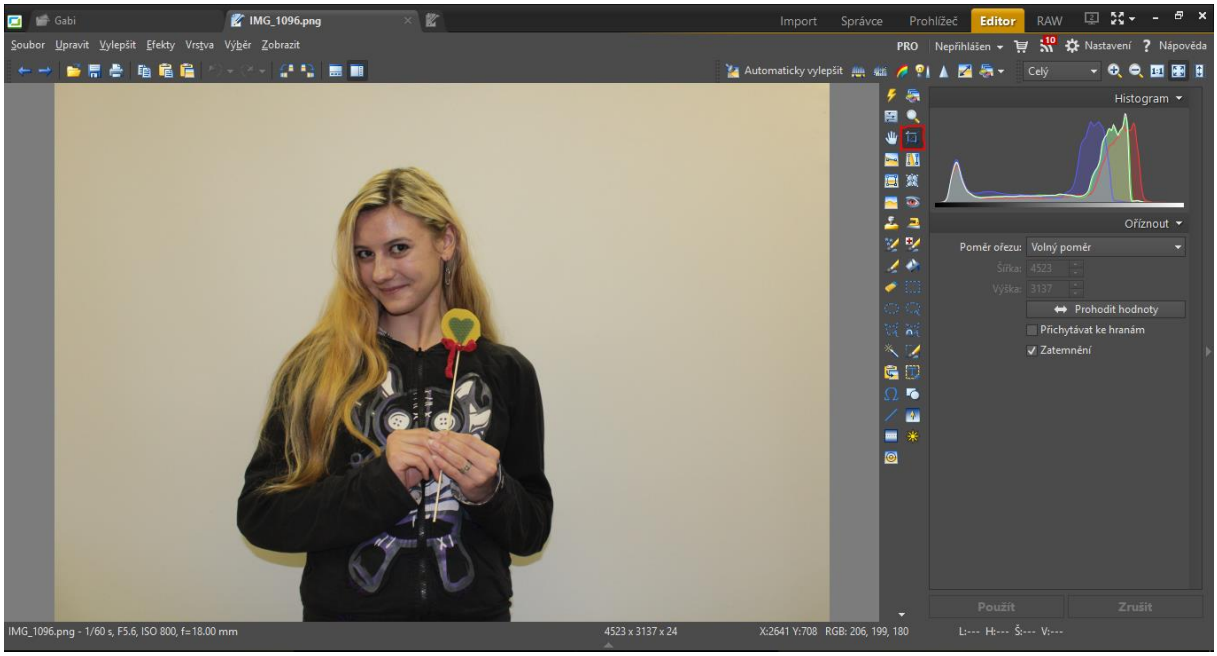


1. Uložení všech změn snímku
2. Práce se schránkou (kopírovat a vložit)

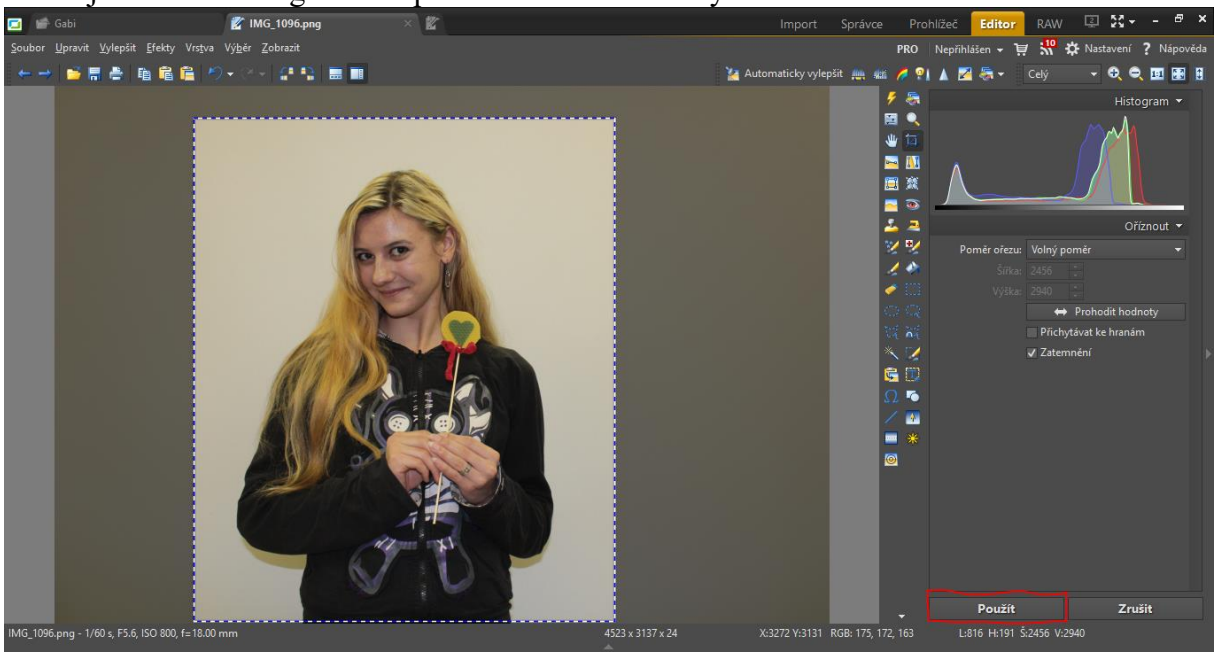
3. Vrácení provedené změny, nebo znovuvyvolání vrácené změny
4. Otočení fotografie doprava nebo doleva, vždy o 90°
5. Úprava jasu, kontrastu a gamy, spolu s jejich kombinacemi pomocí vyvažovací přímky a histogramu
6. Úprava barev, jasu, kontrastu, osvětlení a gamy
7. Úprava teploty barev a osvětlení
8. Doostřování
9. Změna rozměrů

Tolik k základní orientaci v programu a nyní se pojďme věnovat konkrétním možnostem. Začneme u úprav fotografií, protože zde je možností nejvíce, takže si ukážeme jen ty nejzákladnější.

14.6.1.1. Ořezávání



Nástroj k oříznutí fotografie si spustíme kliknutím na vyznačenou část.

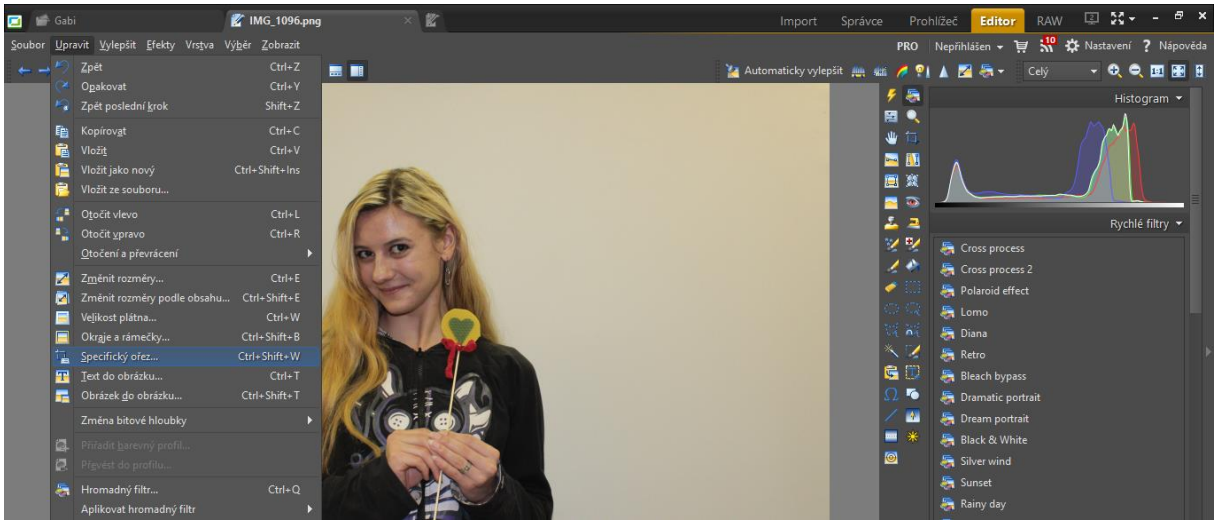


Označíme tu část fotografie, kterou chceme zachovat po oříznutí. Po prvotním označení je možné ještě najet myší na hrany výběrového obrazce a tažením myší (kliknutím levého tlačítka myš a pohybem myši při stisknutém levém tlačítku) změnit polohu jednotlivých hran. Klinutím pravého tlačítka myši dovnitř výběrového obrazce provedeme oříznutí, nebo můžeme v pravé dolní části programu na tlačítko Použít.

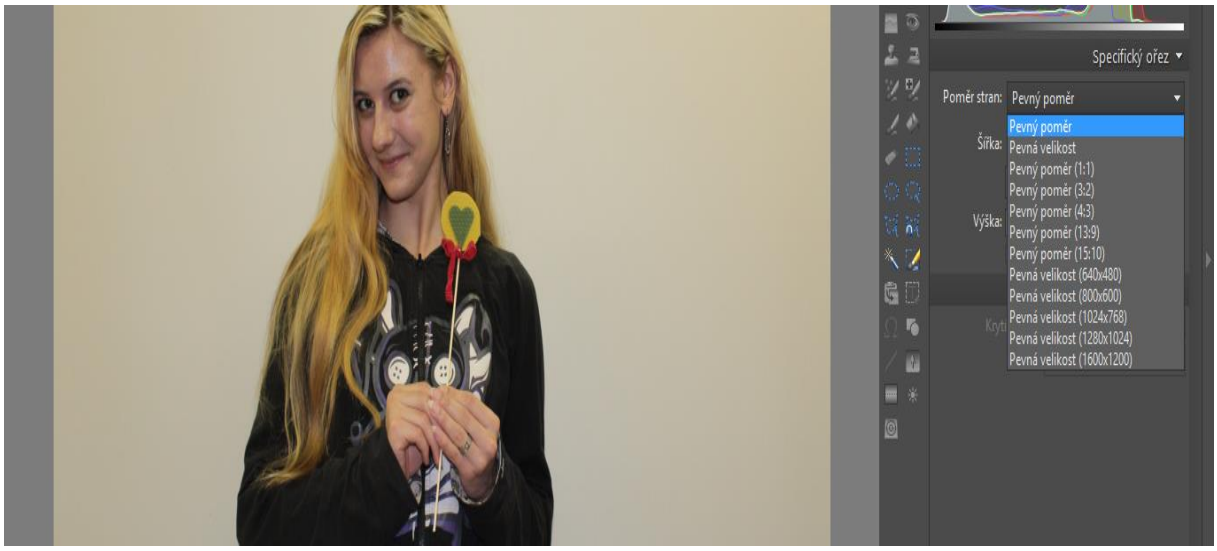
Pokud se nám výběr nepovedl, tak jej zrušíme kliknutím na tlačítko zrušit, nebo kliknutím levým tlačítkem mino výběr.

14.6.1.1.1. Specifický ořez

Jde o speciální typ ořezu, který umožňuje oříznout fotografii v daném poměru stran, na přesně dané rozlišení apod.



Což vede k otevření následujícího menu.



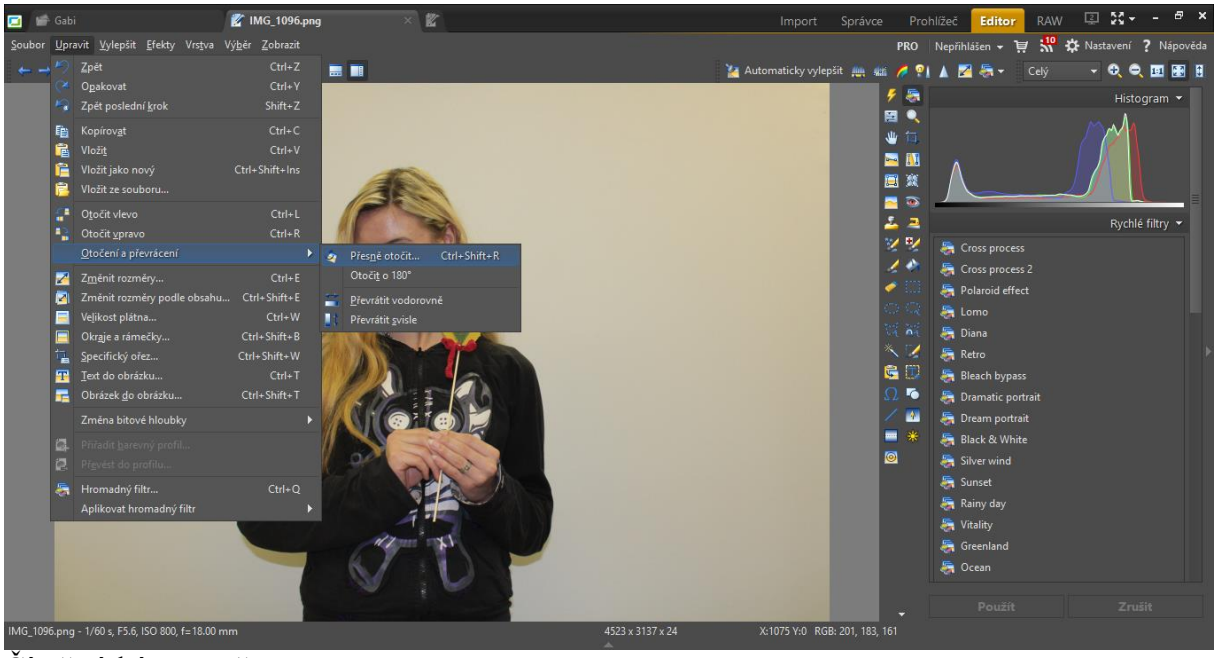
Po vybrání režimu jej dále nakonfigurujeme dle pokynů.

14.6.1.2. Otočení obrázku

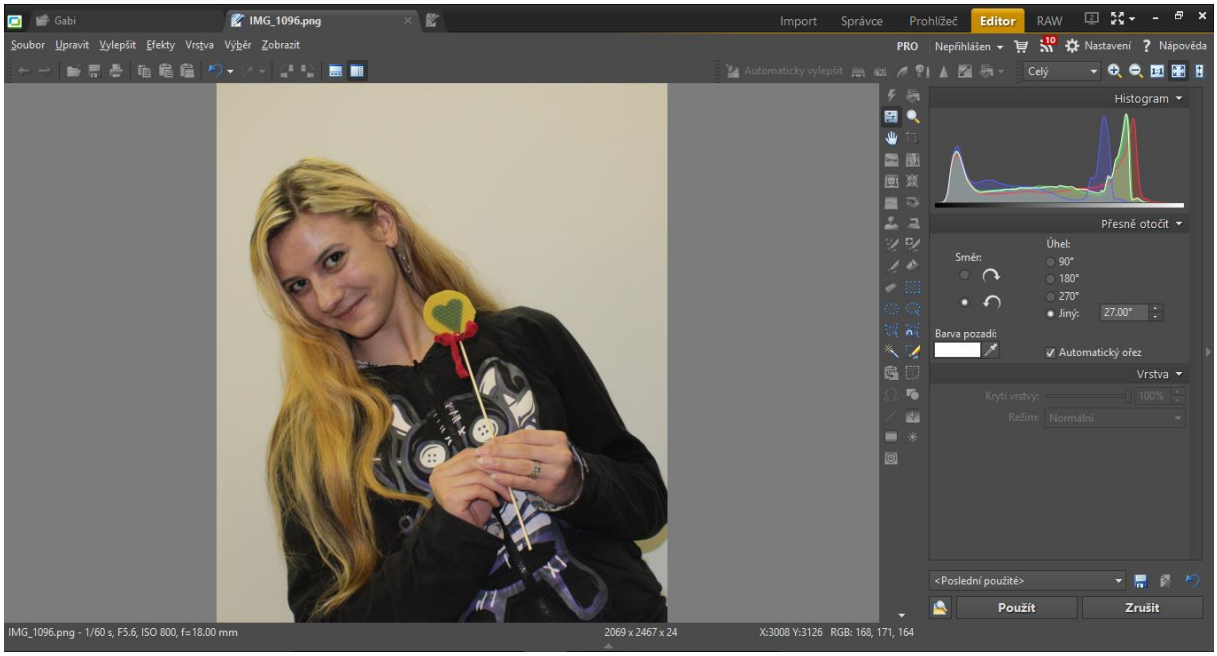
Pro rychlé otočení obrázku o 90, 180, či 270 stupňů použijeme předdefinované ovládání.



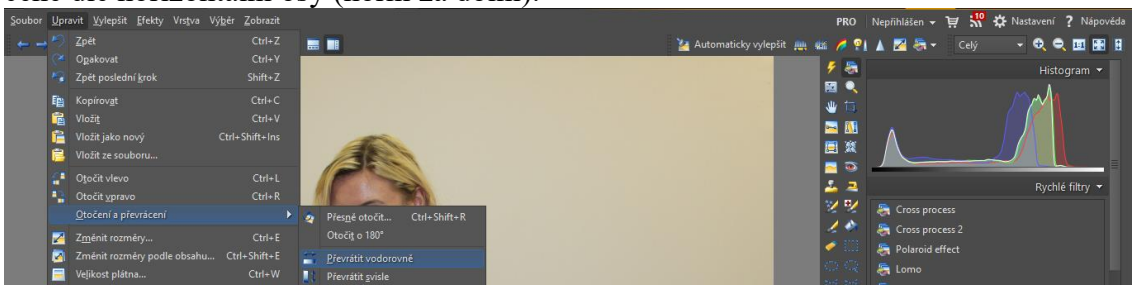
Další předefinovanou možností je možnost otočení o předefinovaný úhel, což najdeme dle obrázku.



Čímž získáme možnost

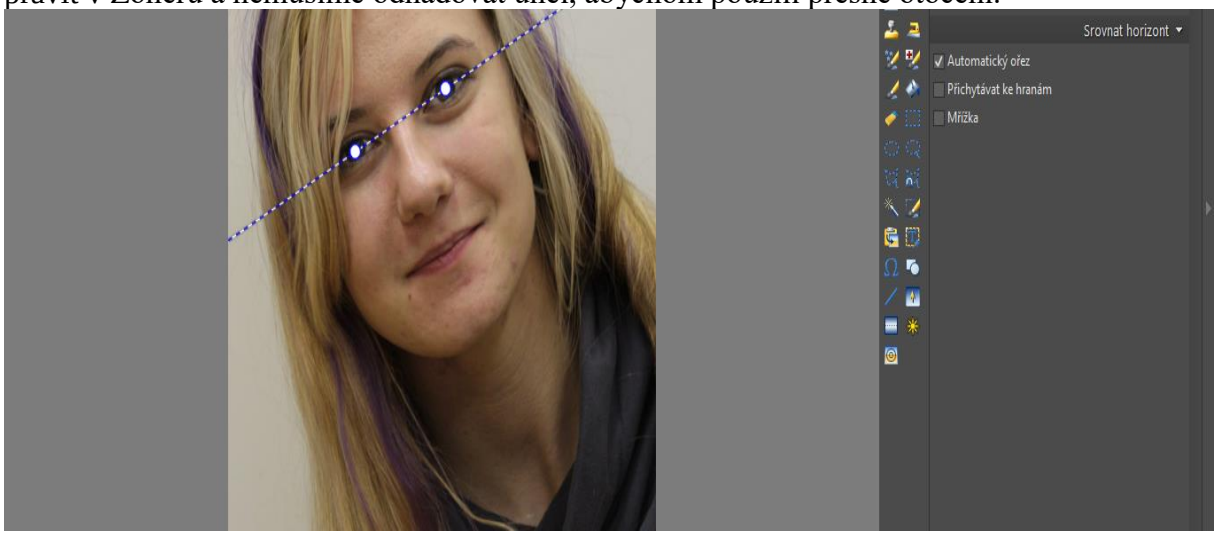


Další možností je převrácení dle svislé osy (záměna stan levá za pravou), nebo převrácené dle horizontální osy (horní za dolní).



16.6.1.3. Srovnání horizontu

Pokud se nám nepovede při fotografování udržet vodorovně fotoaparát, tak i to lze napravit v Zoneru a nemusíme odhadovat úhel, abychom použili přesné otočení.

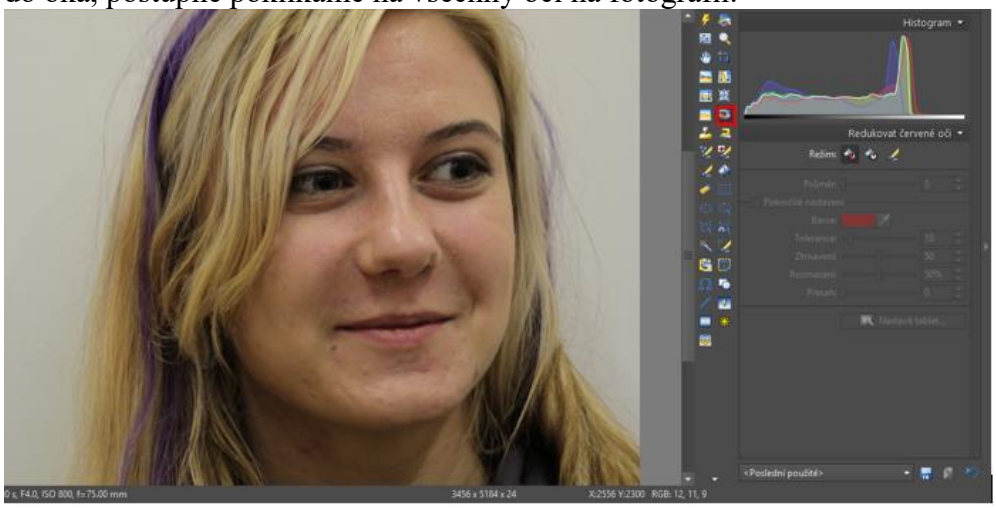


Pomocí zvýrazněné nabídky si aktivujeme „umělý horizont“, který uchopíme za jeden z bodů vyznačených kolečkem a srovnáme si jej podle něčeho, co má být vodorovně (viz obrázek, kdy do roviny srovnáváme oči). Pokud máme horizont srovnaný, stejně jako máme na obrázku, tak jen klikneme na tlačítko Použít a horizont se přetočí do vodorovné pozice spolu s fotografií, kterou fixuje v dané dvojici bodů. Takto můžeme otočit fotografii o libovolný úhel a nemusíme přitom znát a zadávat jeho číselnou hodnotu.

Fotografie se automaticky ořízne tak, aby nevznikly někde nevyplněné fleky a zůstal čtvercový nebo obdélníkový tvar fotografie.

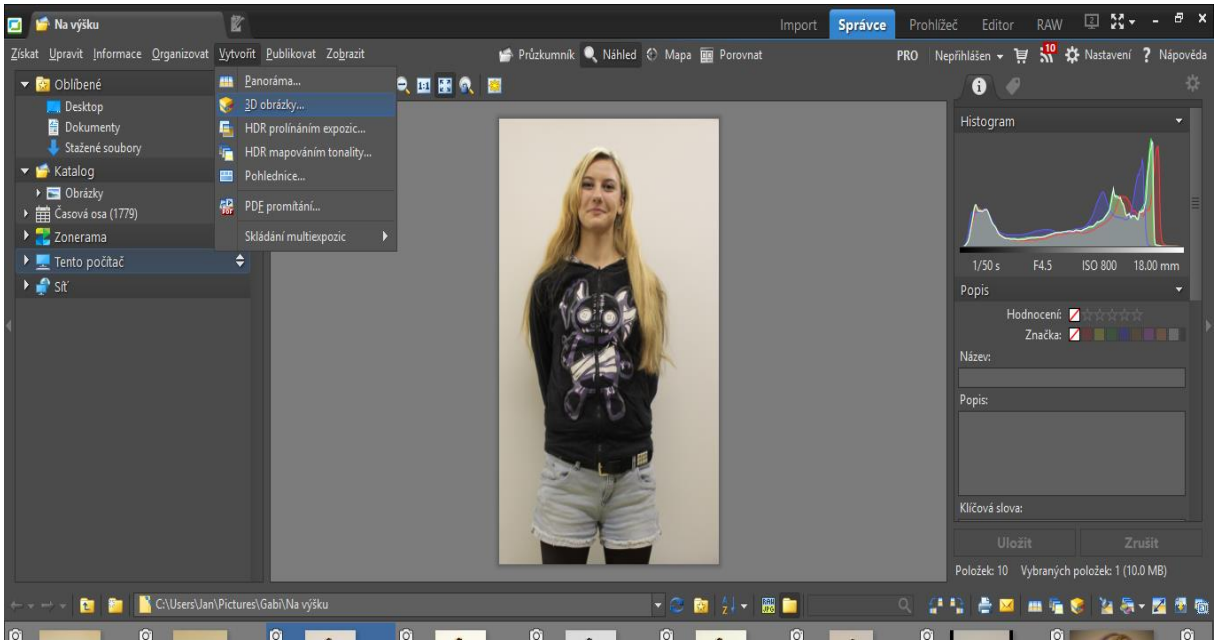
14.6.1.4. Korekce červených očí

Dnes již není příliš potřeba, protože většina fotoaparátů tuto provádí automaticky, pokud je v nich tato funkce aktivní. V Zoneru to funguje jen velmi primitivně, fotku si přiblížíme pomocí lupy, následně si zvolíme nástroj Redukovat červené oči a následně klikneme přesně do oka, postupně poklikáme na všechny oči na fotografii.

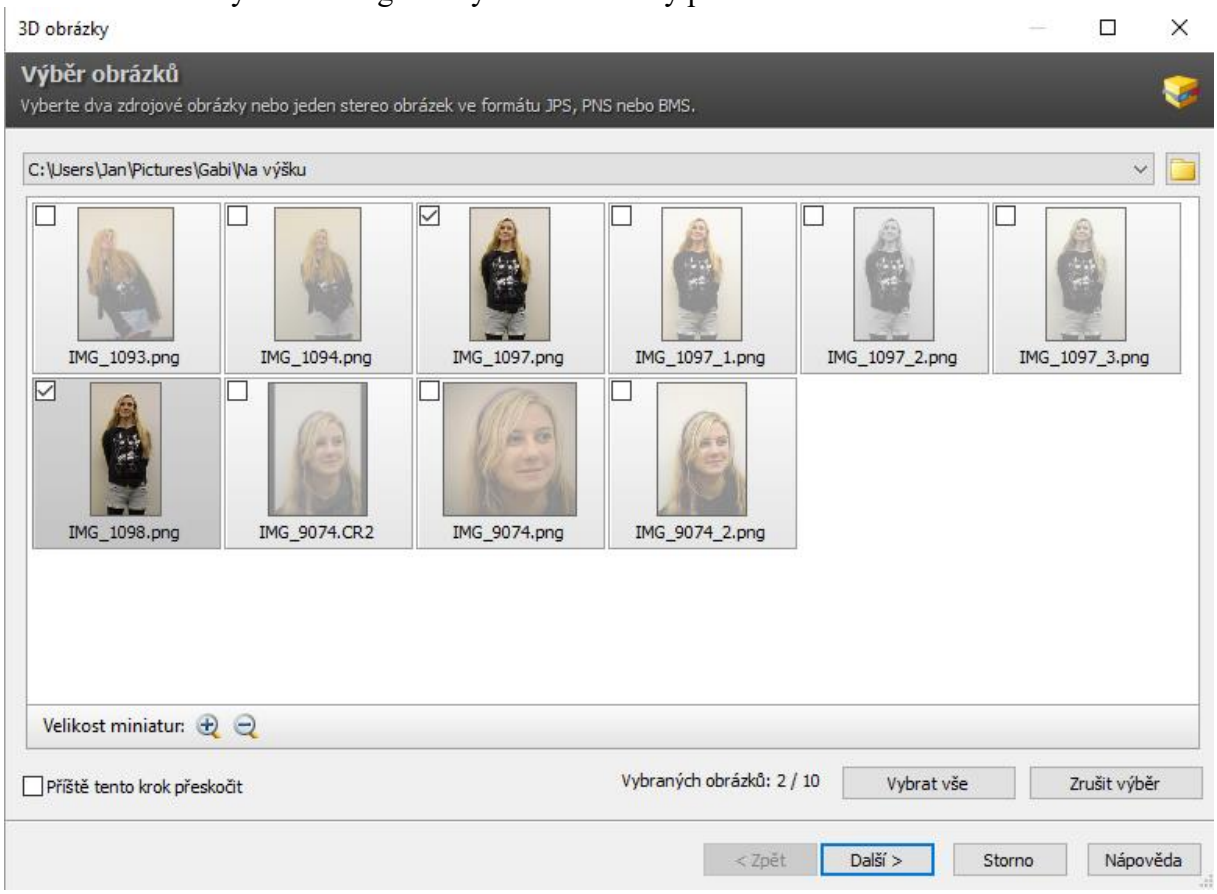


14.6.1.5. Skládání 3D fotografií

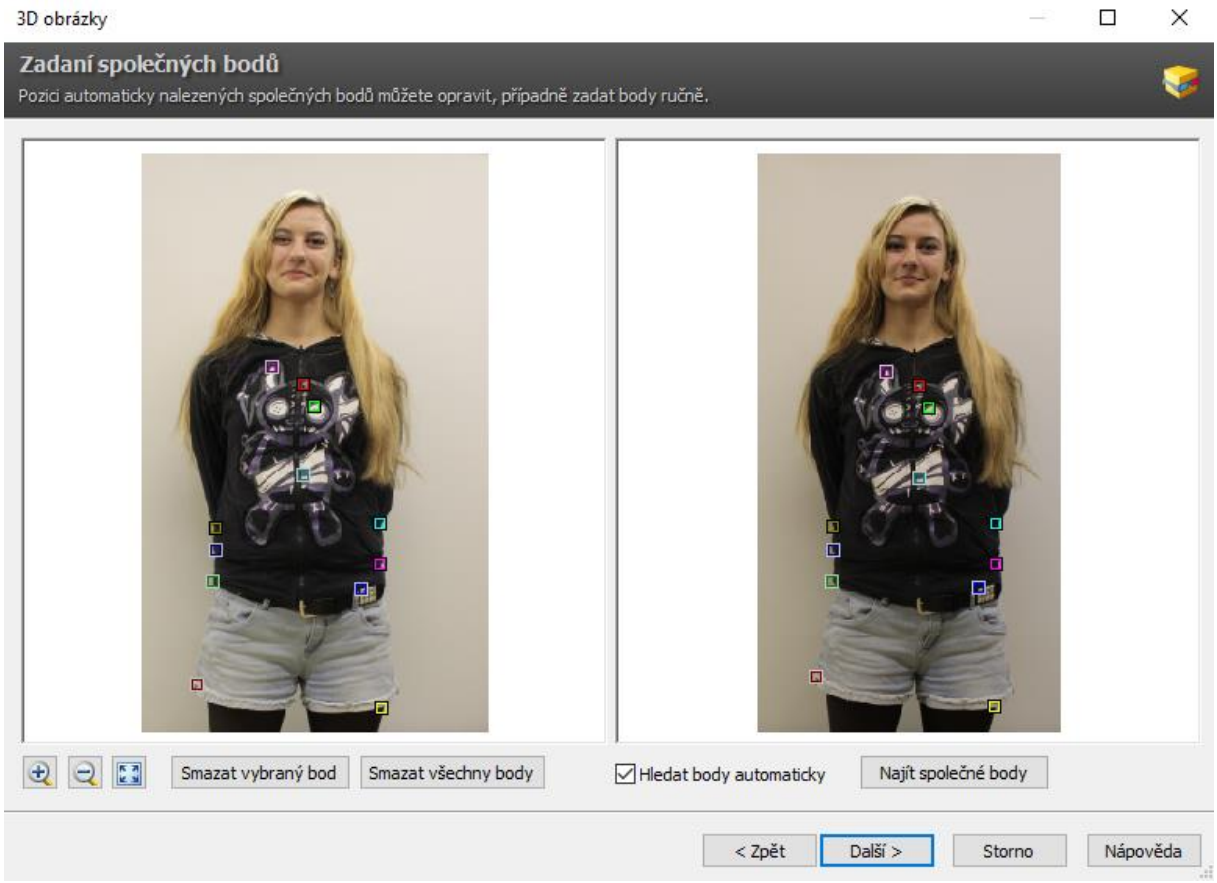
Toto využijeme pro skládání 3D snímků, které jsme si připravili dle postupu, který jsme si popsali v kapitole Fotografujeme 3D snímky. Takto pořízené snímky přesuneme do jedné složky. Následně otevřeme Zoner na záložce správce.



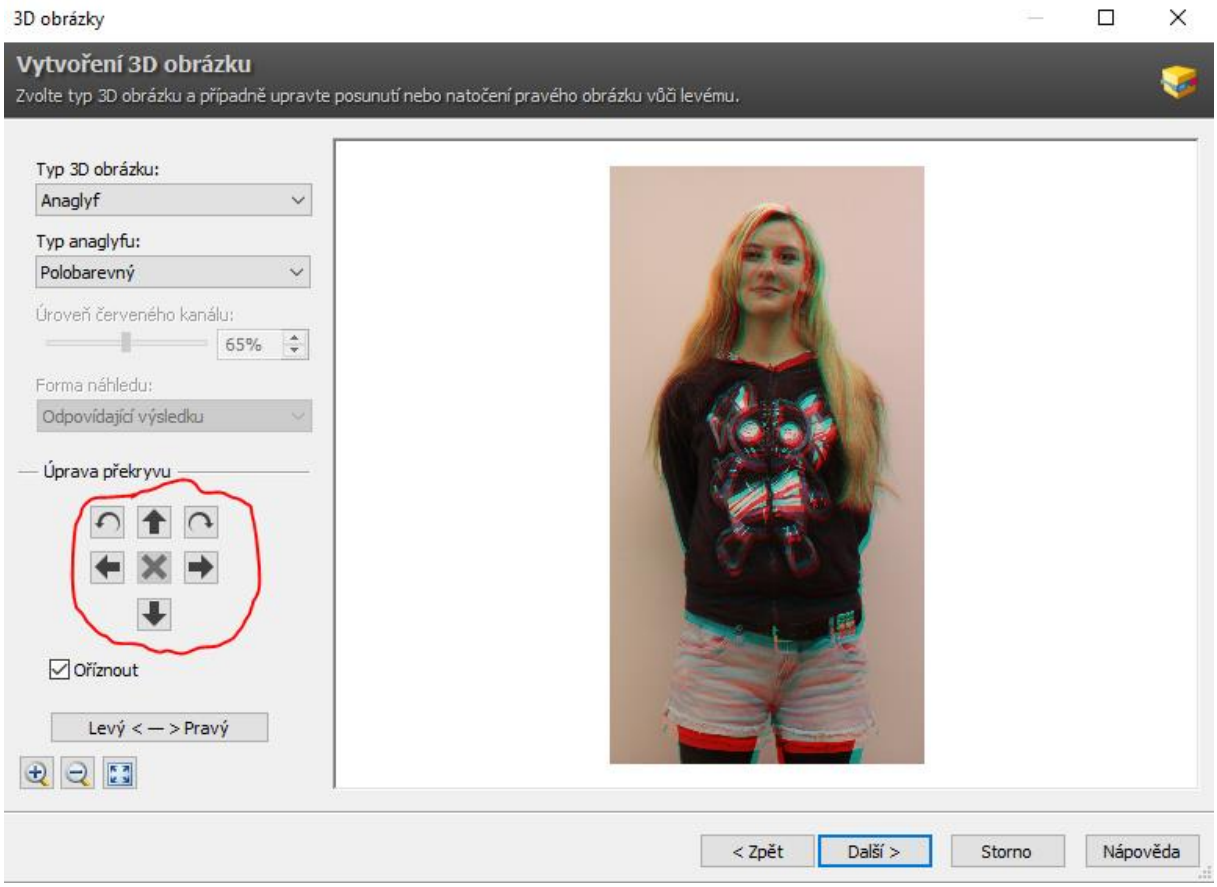
Následně k naší vybrané fotografii vybereme i druhý pohled.



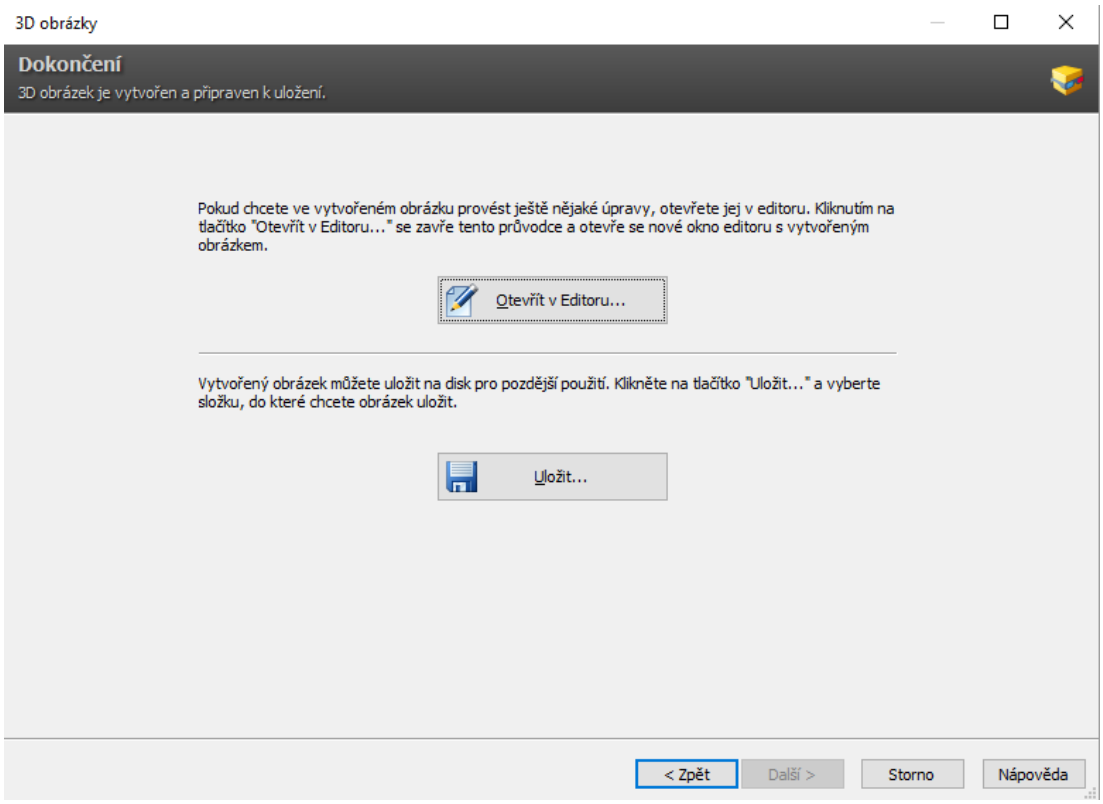
Vybereme další a zkontrolujeme, zda systém rozpoznal společné body na fotografii a dáme Další. V případě, že to umíme a chceme, přidáme či ubereme společné body, ale není to nutné. Všechno potvrdíme tlačítkem Další.



Nyní nastává čas na to, abychom použili 3D brýle. Nyní již budeme pomocí v kruhu vyznačených šipek postupně posouvat obrazy tak dlouho, doku se nám nesloží v ostrý 3D obraz, tedy obraz mající hloubku.

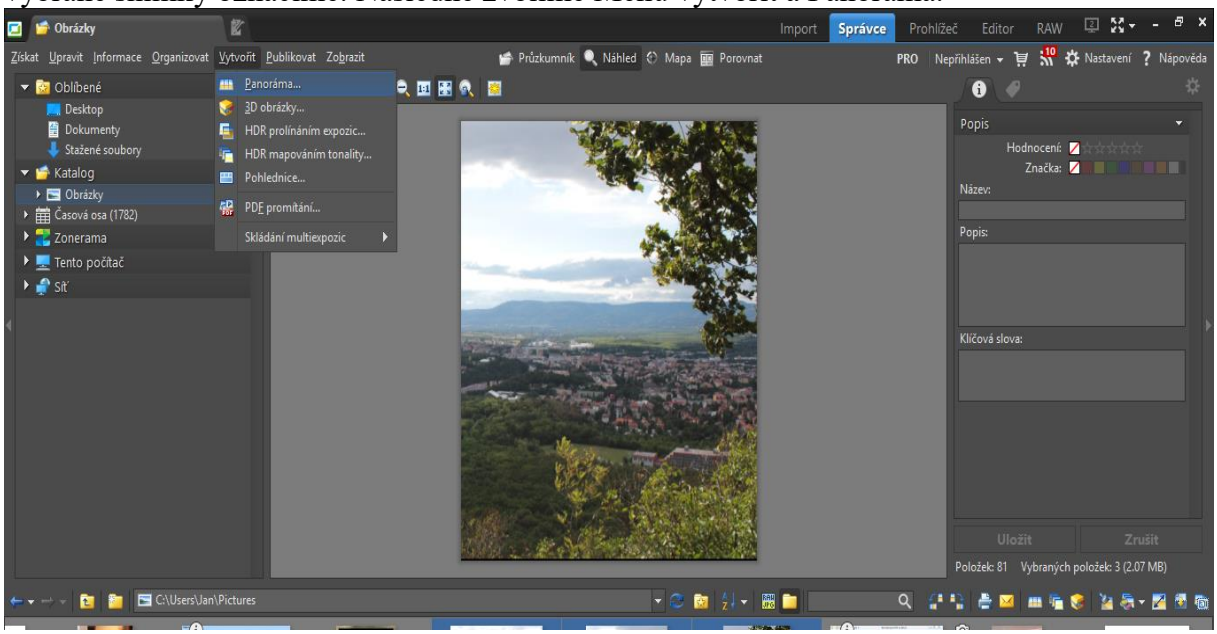


Pokud máme obraz hotový, tak tlačítkem Další přejdeme k volbě uložení či dalšího zpracování vytvořeného obrazu.

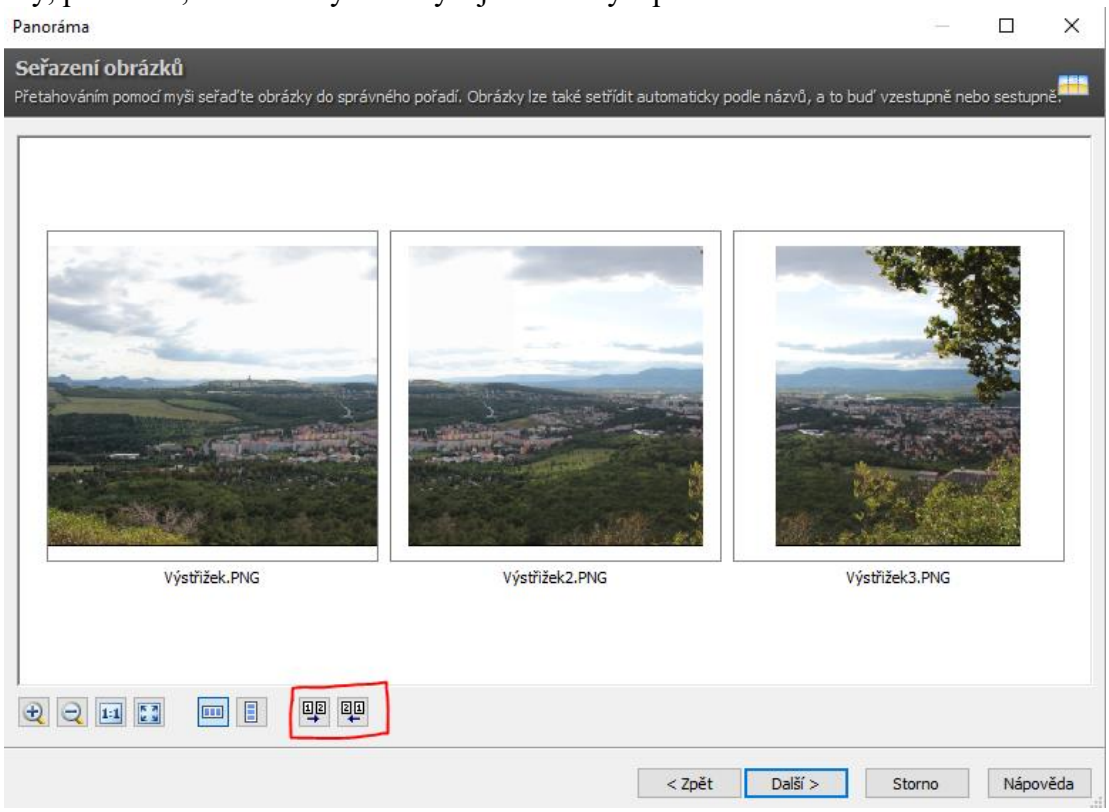


14.6.1.6. Skládání panoramatických snímků

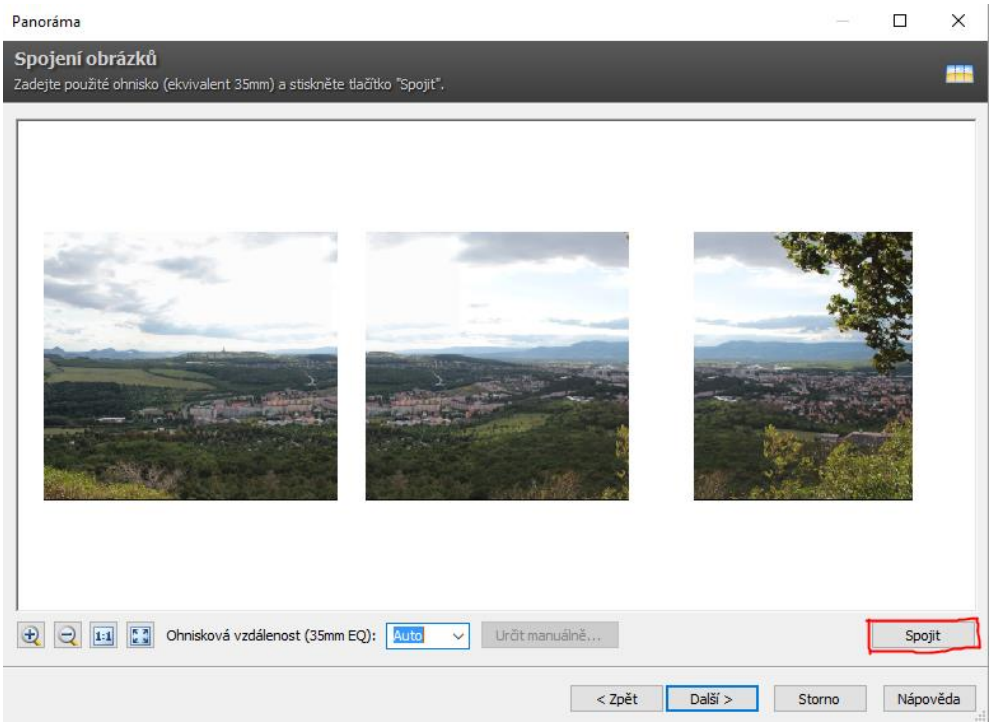
Vytvořené snímky dle postupu v kapitole Fotografujeme panorama opět přesuneme do jedné složky. Následně otevřeme Zoner a pomocí záložky Správce najdeme patřičné fotky a vybrané snímky označíme. Následně zvolíme Menu vytvořit a Panorama.



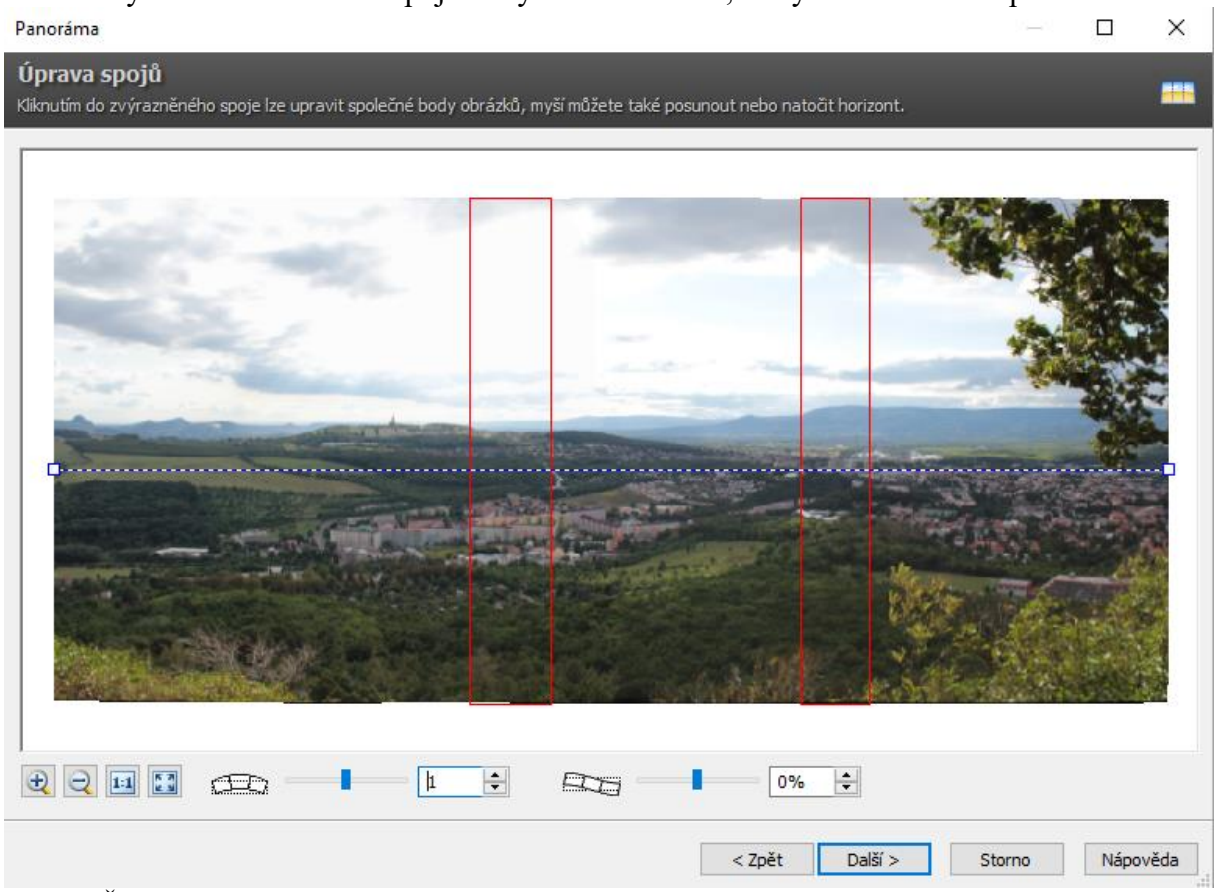
V otevřeném kontextovém okně klikneme na další, pokud máme vybrané všechny snímky, pokud ne, můžeme vybrat zbývající snímky a poté dát Další.



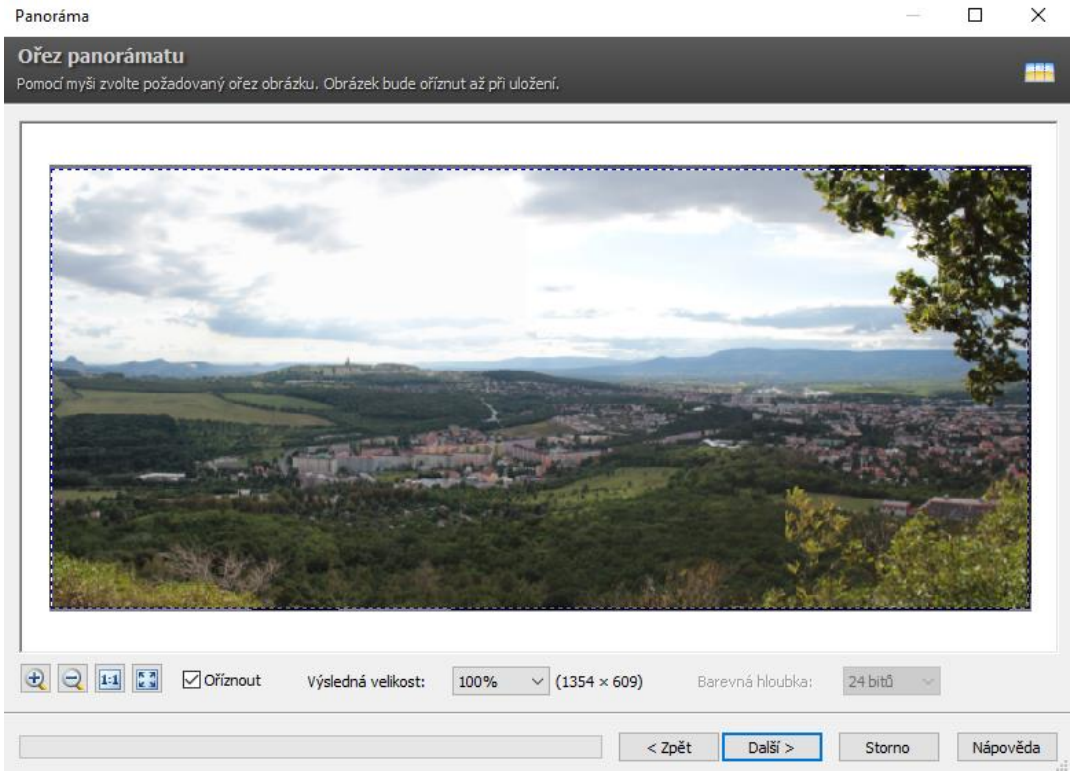
Pomocí vyznačené dvojice tlačítek je možné otočit pořadí snímků pro případ, že bychom fotografovali zprava doleva. Pokud jsou všechny zobrazené obrázky v pořádku a záběr je celý, tak můžeme zvolit Další.



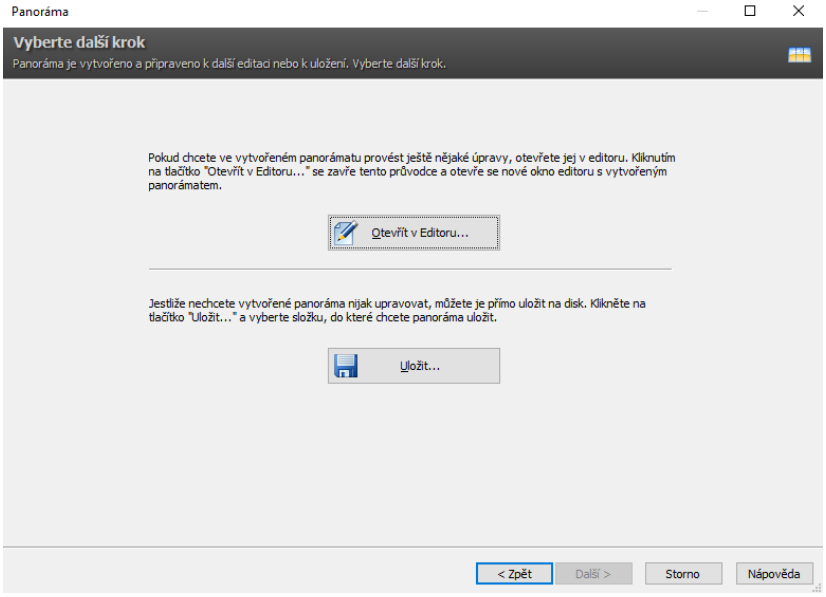
Pomocí vyznačeného tlačítka Spojit si vytvoříme náhled, který můžeme dále upravovat.



Červené obdélníky zvýrazňují překrývání fotografií a vodorovná čára značí horizont. Dole pak můžeme najít úpravu zkosení, většinou je dobré nechat výchozí volbu, pokud jsou přechody ve fotografii i v měřítku 1:1 plynulé, tedy krom červených obdélníků nic na přechody mezi snímky neupozorňuje. Pokud je všechno v pořádku, tak můžeme dát tlačítko Další.



Zde můžeme upravit ořez, který je ve výchozím nastavení vybrán tak, aby byla zakryta „prázdná“ místa, která mohou vniknout při horní nebo dolní hraně. Pokud chceme výřez zmenšit, uchycením hrany a při stisknutém levém tlačítku ji posuneme. Po vyladění výřezu klikneme opět na tlačítko Další.



V tomto bodě již zvolíme, zda chceme fotografii uložit, nebo otevřít v editoru pro další zpracování.

Další postupy a návody udávám ve skriptech digitální fotografie, které jsem sepsal pro cetrum Arpida a budu též publikována online.

15. Počítačová sazba

Desktop publishing (zkráceně DTP), neboli pčítačová sazba je tvorba tištěného dokumentu za pomoci počítače. K tomu, se, aby mohl dokument vzniknout, je zapotřebí kromě počítače a obsluhy, obvykle DTP operátor nebo grafik, také sázecího počítačového programu (např.: MS Publisher). Termín DTP se rozšířil v 80. letech 20. století, kdy společnost Aldus uvedla na trh sázecí program PageMaker. Tento program časem doplnily profesionální QuarkXpress, Adobe InDesign, zdarma poskytovaný TeX (čti TEK) a jeho pozdější komerční klony a v posledních pár letech také Scribus.

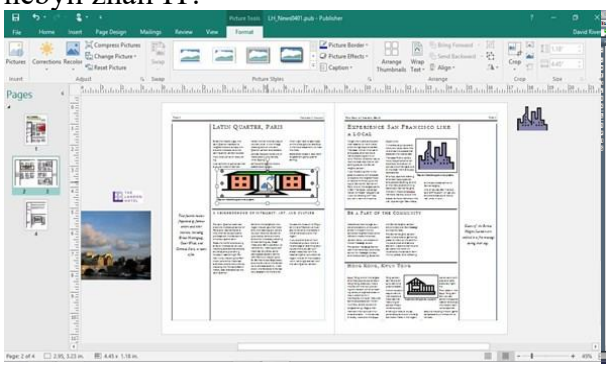
Sazba vzpočívá ve vkládání textů a obrázků na stránku a jejich úpravách. Většina sázecích programů pracuje v režimu WYSIWYG a nabízí celou řadu funkcí, od importu mnoha formátů až po pokročilou přípravu pro tisk.

V dnešní době se zvyšuje počet pracovišť běžných firem využívajících DTP a také se zvyšuje se uživatelský komfort a přístupnost nabízených DTP prostředků. Vzájemně se prolínají aplikační možnosti klasických textových editorů a DTP prostředků (DPT program lze nahradit poslední verzí MS Word, či PowerPoint).

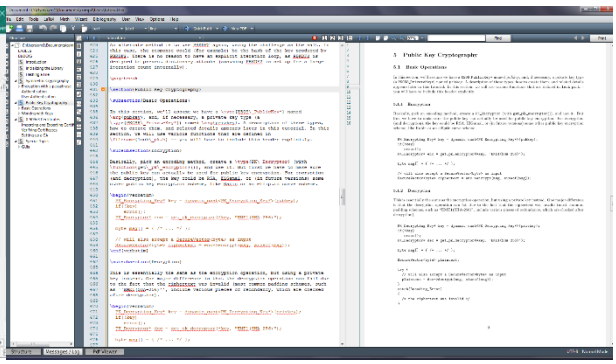
Největší procento užovatelů tvoříuživatelé, kteří disponují výtvarným vzděláním, v problematice dokumentového designu jsou bez hlubší znalosti a dovednosti v informačních technologiích. Další velkou skupinou jsou uživatelé, kteří jsou znalí informačních technologií, kteří nemají výtvarné cítění, nicméně v poslední době se vzájemně sbližují obě třídy z hlediska schopností a dovedností. Už o něco menší, nicméně stále srovnatelnou skupinou uživatelů jsou firmy, které se profesionálně zabývají tvorbou podkladů pro polygrafický průmysl a pracoviště běžných firem, které využívají dle aktuální potřeby DTP prostředky. Nejmenší skupou jsou jednotliví uživatelé, kteří využívají občasně DTP pro prezentování vlastních aktivit.

Hlavním článkem DTP sestavy je počítač, ke kterému jsou připojena odpovídající vstupně výstupní zařízení – např.: skener, tiskárna, tablet (náhrada myši, nikoli tablet, jak si jej představí většina z nás), digitální fotoaparát, kamera a další.

Velkou zásluhu na rychlém rozvoji DTP má společnost Apple, která udělala postskriptovací jazyky, které usnadnily převod a také dostala v začátcích počítače mezi američany, kteří nebyli znalí IT.



MS Publisher 2016



Latex Editor pro Windows

16. Webové stránky, základní seznámení

Webové stránky jsou jedna z mnoha tisíc různých služeb globální sítě jménem internet. Ač se to na první pohled nemusí zdát, tak webové stránky tvoří jen cca 20% přenesených dat, zbytek služeb je často domácímu uživateli skryt. V roce 1989 ve švýcarském středisku CERN v Ženevě definoval Tim Berners-Lee hypertextový systém. O rok později napsal Tim Berners-Lee první program pro tvorbu hypertextových stránek a pro systém navrhl název World Wide Web. Až v roce 1992 je ustanoveno W3C Consorcium, které vydává standarty pro psaní webových stránek, aby stránky byly celosvětově kompatibilní bez ohledu na platformu. Formát html značí soubor hypertextového jazyka a jde o formát, který je plně kompatibilní pro všechny platformy.

WWW (World Wide Web) je celosvětová pavučinová síť, která je součástí internetu. Webová stránka (stránka WWW) je dokument, který je uložen na webovém serveru ve svých základních elementech a který načítá a zobrazuje prohlížeč.

Webový server je počítač umístěný v síti Internet, který obsahuje data, ze kterých jsou sestaveny webové stránky.

Prohlížeč (browser), jde o program běžící na počítači uživatele, který je schopen načíst z webového serveru data tvořící webovou stránku, tato data do oné stránky sestavit a zobrazit na obrazovce počítače.

HTML (Hypertext Markup Language) je jazyk, který je použitý k popisu webových stránek.

HTML nebo XHTML soubor je textový soubor obsahující kód jazyka HTML (XHTML). Tento soubor obsahuje návod pro sestavení webové stránky, je čten, interpretován a ve výsledné podobě zobrazován prohlížečem v podobě webové stránky.

XML (Extensible Markup Language) je jazykem pro záznam strukturovaných dat, který má přísnější pravidla pro psaní kódu, umožňuje tvorbu vlastních tagů podle potřeby, zcela odděluje obsah od formátovacích příkazů.

XHTML definuje verzi jazyka HTML kompatibilní s XML, tedy mění definici HTML tak, že striktně využívá syntaxi jazyka XML.

Syntax je „větná skladba“ daného jazyka. V mluvených jazycích definuje vztahy mezi slovy ve větě a popisuje, jak správně utvořit větu. Syntaxe v informatice definuje pravidla pro zápis příkazu v nějakém programovacím nebo formálním jazyce. Popisuje význam použitelných znaků, význam klíčových slov (slova, která mají v daném jazyce pevně stanovený význam).

16.1. Základní rady a principy

Při návrhu stránek šetřit velikostí dat. Je vhodné zmenšit obsah tak, aby se načítal co nejrychleji, ale byl kvalitní, týká se to hlavně obrázků, užité grafiky a webových aplikací, které by měly být napsány tak, aby měly co nejmenší a nejjednodušší strukturu.

Umísťování odkazů seřídí pravidly. Tvořme text tak, jako kdyby v něm žádné odkazy nebyly. Odkazy jsou čitelné už tím, že mají obvykle modrou barvu a jsou podtržené. Příklad nesprávného použití: Na ulici Lipové č. 25 naleznete Muzeum orientálního umění, umění, jehož domovská stránka je zde. Příklad správného použití: Na ulici Lipové č. 25 se nalézá Muzeum orientálního umění, které vytvořil známý orientalista Jaroslav Řipský. Hlavně odkazy neplýtvat, vím že ti, kdo znají moje stránky by mohli říct, že to sedí, ale tam jsou odkazy na umístění externích souborů, což je vhodné, pokud máme málo místa na webovém serveru.

Manipulace s obrázky je další častý problém spousty webových stránek, pomineme-li tedy flash sekvence. Používejme co nejmenší velikost obrázků a doporučuji formát jpg (v Zoneru je přímo možnost uložit jako jpg pro web), či png. Obrázky by hlavně měly splňovat

svůj účel a měly by mít popis. Odkazy na grafiku na jiných serverech www prodlužují načítání, ale mohu doporučit, pokud někdo chce zveřejňovat kvalitní fotografie ve formátu raw (případně bmp) udělat jedno, dát náhled o malých rozměrech v jpg a na externí server nahrát originál (i desítky MB) a do náhledu vložit odkaz pro stažení.

Stránky by měly být optimalizovány pro přehledné čtení a mít jednoduchou grafiku. U dlouhých dokumentů přetékajících přes několik stránek hrozí, že čtenář ztratí přehled nebo orientaci. Udělejme na začátku takového dokumentu obsah (nadpisy jednotlivých částí dokumentu) a vytvořte provázání pomocí odkazů, nebo rozdělme takový dokument na více samostatných stránek. Základní navigační rozcestníky by u jakýkoliv stránek, které se nevejdou na jednu obrazovku, měly být na začátku i na konci dokumentu. Neumísťujme na vaše stránky tlačítka typu Zpět, Předchozí Další … Pokud používáte grafické odkazy (tlačítka), nezapomínejte na alternativní text pro uživatele, kteří mají vypnutou grafiku (<!--[if !supportLists]--><!--[endif]-->).

16.2. Co je tedy html?

Textový jazyk pro popis stránky. HTML stránka je obyčejný textový soubor obohacený o tagy a atributy, které určují význam textu a vzhledu celé stránky. Popis stránky napsaný v tomto jazyce není třeba překládat do strojového (binárního) kódu. V textové podobě je předložen prohlížeči a ten stránku sestaví. Podstatné dvě složky jazyka jsou:

- Hypertext (odkazy na externí elementy)
- Markup (značky pro formátování dokumentu – tagy)

Pokud se na stránce vyskytují binární data (obrázky, animace…) je na ně odkázáno. Tato data nejsou umístěna do souboru se základním popisem. Jazyk má své příkazy (tagy). Příkazy jsou spolu se svými parametry uzavírány do špičatých závorek.

16.3. Syntaxe

Jazyk HTML má textový formát a přímo v tomto formátu je používán. Jazyk je vystavěn především z tagů a odkazů. Tag je chráněné slovo jazyka uzavřené do špičatých závorek.

Všechny příkazy jazyka jsou tagy, vše, co je mimo špičaté závorky je vlastní obsah stránky

Tagy se rozdělují do dvou skupin:

Párové tagy – vyskytují se ve dvojicích a uzavírají mezi sebe vlastní obsah. Jeden tag zahajuje a druhý ukončuje danou konstrukci. Oba tagy mají shodné jméno, ukončující tag obsahuje na začátku lomítko. Na příklad: <B>tento text bude tučný </B>

Nepárové tagy – jsou tvořeny pouze jedním slovem uzavřeným do závorek. Příklad: <IMG src=”obrazek.gif“/>

Existuji atributy (vlastnosti), pomocí kterých můžeme ovlivnit chování příkazu (tagu). Pokud chceme pro tag nastavit více atributů, tak oddělují se mezerou. Některé atributy jsou stejné pro více tagů, např. align, size, color... Atributy spolu s některými tagy mohou být umístěny v externím souboru CSS (tzv. kaskádové styly, ale těmi se zaobírat nebudeme).

16.4. Současné www, tedy standart xhtml 5

Zavádí přísnější pravidla pro psaní kódu:

- Tagy a atributy musí být psány malými písmeny.
- Všechny elementy musí mít uzavírací značku.
- Hodnoty atributů musí být v uvozovkách.
- Uvnitř prvku <html> musí být obsažena hlavička <head>, která musí obsahovat prvek <title>.

- Elementy se nesmí překrývat (Příklad překrytí: zdůrazněný odstavec)
- V komentářích musí být pomlčky <!- - komentář - - >
- Nepřípustná je minimalizace atributů. Atribut musí mít vždy přiřazenu hodnotu.
- V tagu <html> musí být atributu xmlns přiřazena hodnota standardu: <html xmlns = "http://www.w3.org/1999/xhtml" xml:lang="cs"> …..</html>
- Nesmí se používat nestandardní elementy v kódu.

16.5. Kódování pro český jazyk

Pro většinu jazyků, ve kterých je text napsán, je potřeba nastavit správné kódování, aby se zobrazovaly správně všechny znaky užité v psaném projevu. Určitě to zná každý z nás, když se mu na webových stránkách na místo písmen s diakritikou zobrazují čtverečky či jiné kolikrát dost divné znaky. Základem je, aby prohlížeč měl nastaveno stejné kódování (stejné nastavení mapy znaků) jako webová stránka. Pokud chceme na webových stránkách kvalitně kódovat český jazyk, tak máme 3 možnosti:

iso-8859-2, což je ISO norma (Latin 2) pro východo a stredo evropské jazyky na standardu UNIXu (Linuxu).

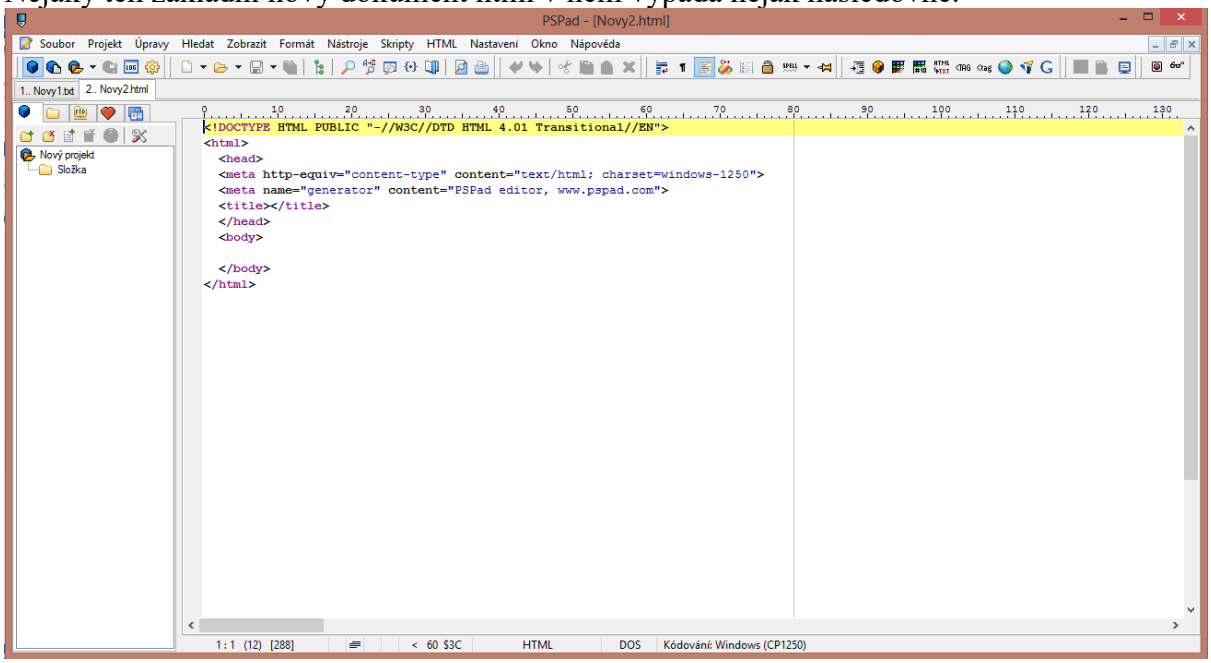
windows-1250 je nejběžnější kódování češtiny. Bylo vytvořeno pro platformu Windows, respektive Internet Explorer (prvotní nastavení starších IE).

UTF-8 je univerzální kódování Unicode, které je shodné pro všechny jazyky.

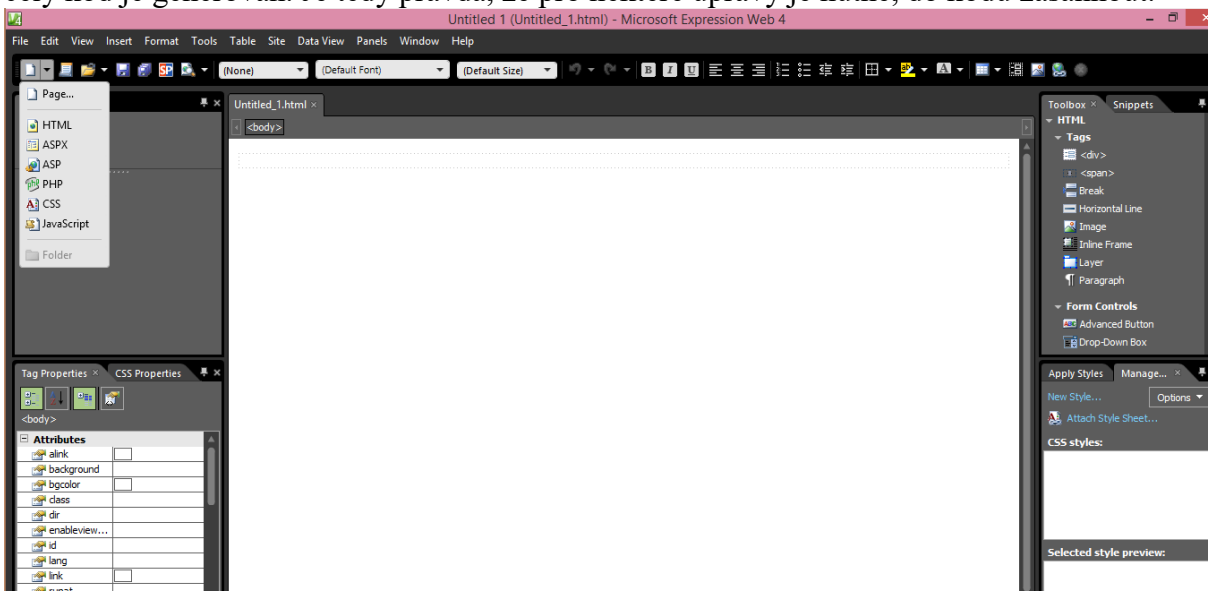
16.6. V čem psát?

Záleží, co potřebujeme, pokud se naučíme kvalitně jazyk a bude nám stačit jednoduchá stránka, tak můžeme použít poznámkový blok, kdy u výsledného souboru s kódem jen změníme příponu z txt na html.

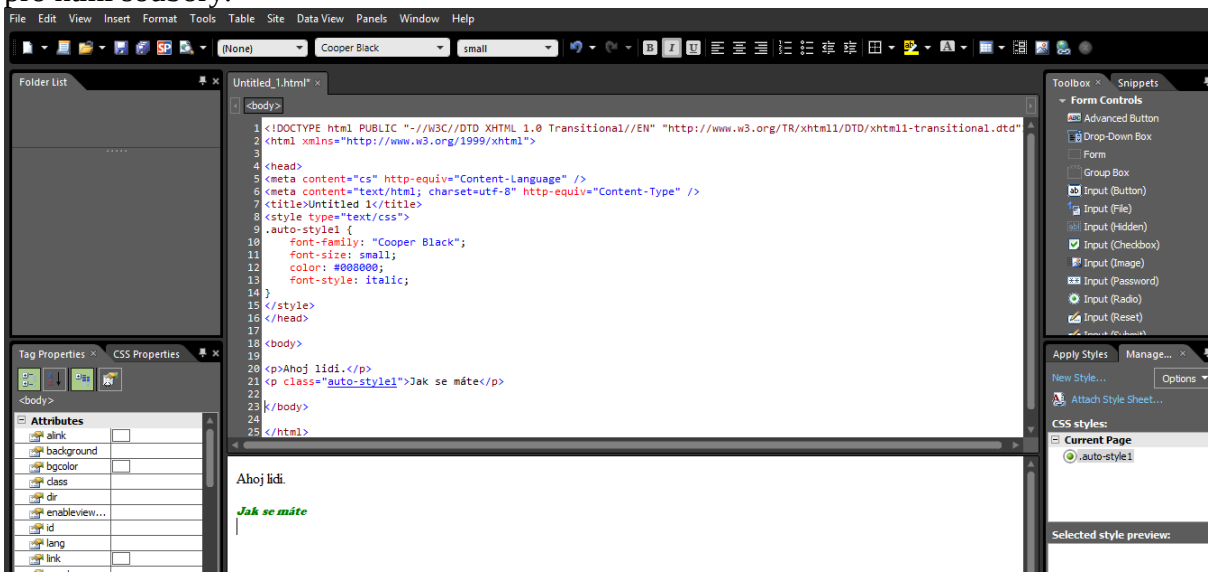
Pokud chceme psát složitější web, nebo si nejsme tak kamenně jistí v příkazech, tak můžeme použít editor typu PS Pad, který je vybaven nápovědou (po zadání ctrl + mezerník zobrazí seznam tagů) a pomocnými tabulkami pro vyplňování atributů. Tento free editor bych doporučil všem začátečníkům a všechny ukázky, které zde uvidíme byly vytvořeny v něm. Nějaký ten základní nový dokument html v něm vypadá nějak následovně.



Pro složité konstrukce nebo „mazáky“ doporučuji program MS Expresion Web, který má velké možnosti ne jen v html. Jeho velkou nevýhodou pro lidi jako jsem já je to, že není k dostání český jazykový balíček, tedy program je nutno obsluhovat v angličtině. Program podporuje jak přímou práci s tagy a atributy, nebo vytváření stránky ve formě grafiky a textu, kdy celý kód je generován. Je tedy pravda, že pro některé úpravy je nutné, do kódu zasáhnout.

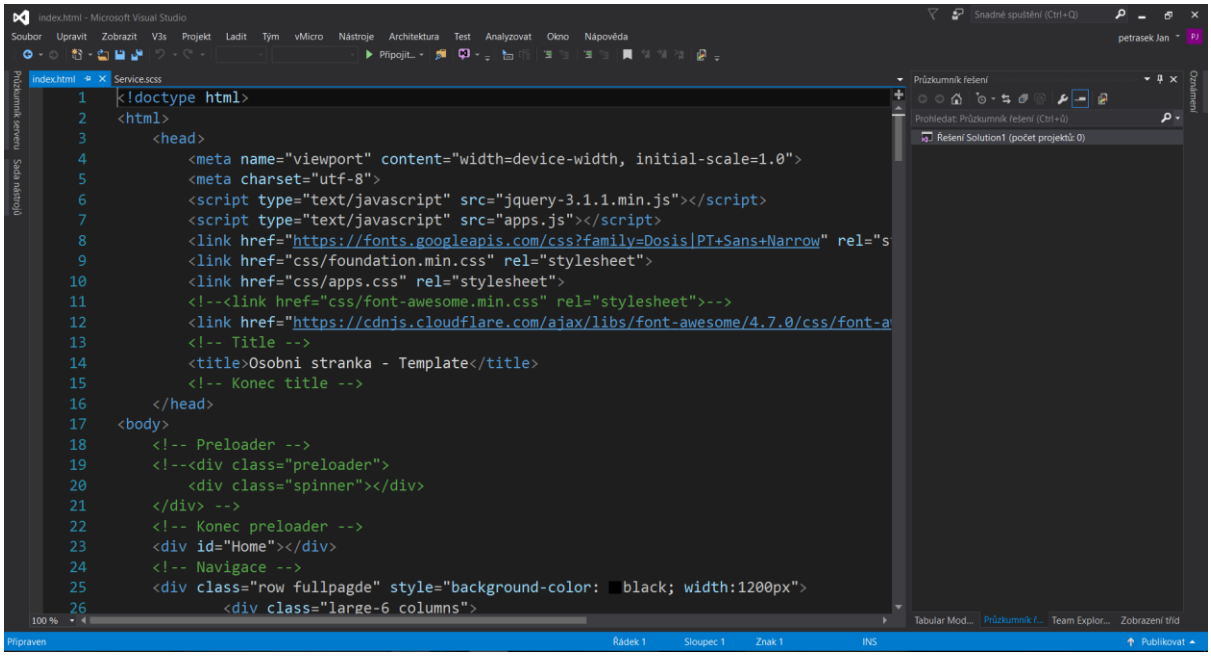


A takhle nějak vypadá i kód, který je automaticky generován ve výchozím nastavení pro html soubory.



Jak je vidět na ukázce, tak tento program je skutečně určen spíše pro pokročilé uživatele, ač musím přiznat, že dokáže hodně ulehčit práci. Sám se s tímto editorem teprve učím, takže se nebudeme nikterak více pouštět do jeho zkoumání, ale pro ty zkušenější mohu doporučit.

V dnešní době lze kvalitně psát ve Visual Studiu verze 2015 a 2017, kdy je kvalitní podpora pro všechny jazyky, které jsou potřba pro tvorbu webu. Pro php doporučuji použít i doplněk a velkou výhodou je podpora scss – preprocesorovaných kaskádových stylů.



16.7. Základní elementy hlavičky

- Doctype – Nastavuje verzi jazyka
- Přechodový (transitional) režim – Umožňuje starší příkazy, porušování striktních pravidel
- Rámcový režim – Obrazovka dělená do buněk
- Přísný režim – Netoleruje staré tagy a další chyby

Příklad hlavičky najdeme na následujícím obrázku, kde je vidět, jak se používají jednotlivé elementy.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=windows-1250">
  <meta name="generator" content="PSPad editor, www.pspad.com">
  <title></title>
  </head>
```

16.8. Elementy

16.8.1. Řádkové elementy

Jsou to elementy, které se nachází uvnitř textu, nedochází po nich k zalomení. Obvykle plní funkci zvýraznění nějaké části textu. Umožňují samostatně formátovat úseky textu.

Patří sem např. <a>, , , , , <sup>, <sub> a <q>. Řádkové elementy mohou obsahovat text a jiné řádkové elementy, nemohou ale obsahovat blokové elementy. Co že znamená který tag? Pojďme se na ně podívat.

- ** - kurzíva**
- ** - kurzíva**
- ** - tučný**
- ** - učný**
- ** - logický úsek textu**
- ** - index dolní index**
- ** - horní**

- **<q> </q> - citace (uvozovky), (atribut cite + url zdroje)**
- **<a> - odkaz**
- **
 - odřádkování**

Jak vypadá užití takového řádkového elementu v praxi si ukážeme na dalším obrázku. Dejme tomu, že chceme vytvořit text, který vypadá nějak následovně.

Emma si šla koupit *nové* nové šaty na pondělní vystoupení.

Formátování je pravda nesmyslné, ale je zde ukázka toho, jak se užívají řádkové tagy. Jen malá poznámka, většina řádkový tagů je párová. Kód pro naší nesmyslnou větu bude tedy v jazyce html vypadat následovně.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=windows-1250">
  <meta name="generator" content="PSPad editor, www.pspad.com">
  <title></title>
  </head>
  <body>
  <b>Emma</b> si šla koupit <i>nové</i> nové šaty na
   <a href="www.bing.com">pondělní</a> vystoupení.

  </body>
```

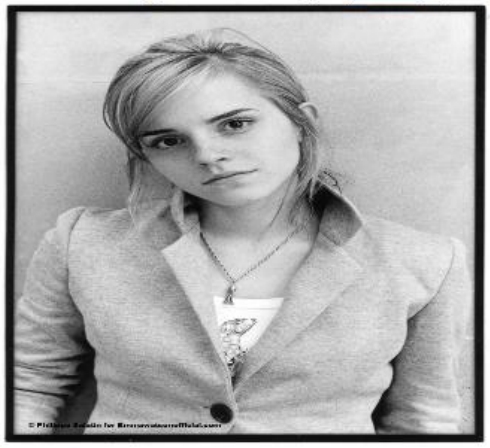
Poďme si rozebrat jediný možná trošku komplikovaný tag, kterým je <a>, tedy odkaz. Povinný je atribut <href> nebo <name>. Odkaz může směřovat na: jiný web, jiný soubor na lokálním počítači, místo na té samé stránce. Odkazem běžně bývá text, či obrázek. Pokud odkaz směřuje na jinou stránku, tak bude vypadat nějak takto: vyhledávač. Na začátku odstavce jsou zmíněny dva atributy a nyní po prvním příkladu nstal čas si prozradit, který atribut se kdy užívá. Atribut <heref> = ¨URL¨ se používá, pokud se jedná o odkaz na jinou webovou stránku a lze jej použít v menu pro odkázání se na jinou html „stránku" našeho webu i když to není zrovna učebnicově správně. Atribut <target> = ¨název¨ se užívá pro odkázání se na místo na stránce či soubor na lokálním zařízení. Nicméně oba tagy pracují rovnocenně, takže stačí si pamatovat jeden jediný. Pokud se odkazuji na jinou stránku (na lokále): hodnota href je relativní cesta k souboru (<a href"stranky/pevni_stranka". Pokud se odkazuji na nějaké místo v aktuální stránce, tak se vloží se záložka zde je záložka směřuje na odkaz: nadpis.

16.8.2. Blokové Elementy

Jsou to elementy, které tvoří nějaký blok. Po takovém elementu je text dokumentu zalomen, odřádkován. Tyto elementy mohou obsahovat řádkové elementy i další blokové elementy. Mezi ně patří například odstavce (<p>) nebo nadpisy (<h1> - <h6>), tabulky (<table>) , element <body>, , seznamy (<ul>, <li>), <pre>, <blockquote>. <h1> až <h6> Jsou nadpisy, které mají podobnou strukturu jako jednotlivé úrovně nadpisů třeba ve Wordu. <hr></> tvoří oddělovací čáru. Tzv. entity – spedciální příkazy pro vložrní znaků, které nejsou na klávesnici (..... vloží mezeru, Α ..... vloží řecké písmeno alfa apod.).

Poďme si opět skusi nějaký opravdu nesmyslný příklad. Dejme tomu, že chceme vytvořit takovouto stránku:

Emma si šla koupit *nové* nové šaty na pondělní vystoupení.



Lorem
ipsum dolor sit amet consectetuer eros suscipit tempor netus at.
Augue volutpat elit pede consectetuer Aliquam nibh justo pretium
Quis nibh Aenean Donec tellus elit nec mauris nibh feugiat nec.
Mauris tortor elit porttitor nunc neque semper facilisis orci eros or
Tempus justo Morbi tempor venenatis ridiculus mollis.

Co zde vidíme, jde vlastně o několik odstavců, větu z prvního příkladu a obrázek. Tak se poďme podívat, jak by mohl vypadat zdrojový kód.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=windows-1250">
  <meta name="generator" content="PSPad editor, www.pspad.com">
  <title></title>
  </head>
  <body>
      <br><b>Emma</b> si šla koupit <i>nové</i> nové šaty na
      <a href="www.bing.com">pondělní</a> vystoupení. </br>
      <a href="http://www.emmuszka.estranky.cz/"><img src="C:/Users/Jan/Pictures/Emma Wa
      <hr align="left" color="#0000FF"><p><em><b>Lorem</b></em><br></>
      ipsum dolor sit amet consectetuer eros suscipit tempor netus at.
      <br>Augue volutpat elit pede consectetuer Aliquam nibh justo pretium Nullam nibh.
      <br>Quis nibh Aenean Donec tellus elit nec mauris nibh feugiat nec.
      <br>Mauris tortor elit porttitor nunc neque semper facilisis orci eros orci.
      <br> Tempus justo Morbi tempor venenatis ridiculus mollis.</p>


  </body>
</html>
```

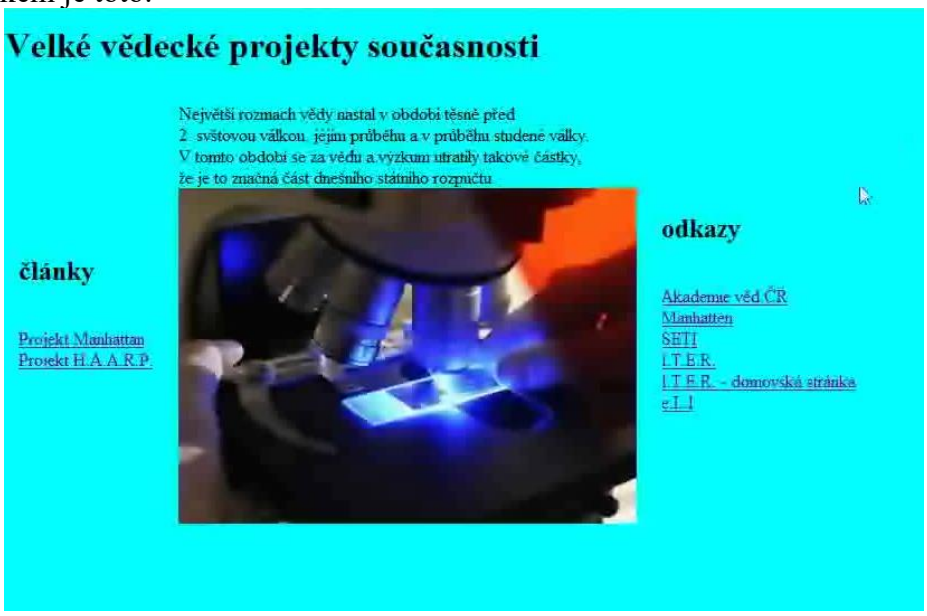
Co, že znamenají jednotlivé tagy?

-   - odstavec (font-family, color, text-align (zarovnání), margin-left (šířka okraje bloku), background-color (hodnota je barva pozadí)
-   ucelený oddíl – směrování pomocí skriptu, nadřazen odstavci
- <img> </img> obrázek – povinný atribut src (cesta k obrázku)
- <Pading> </> šířka vnitřního okraje
- <margin> </> vnější okraj stránky
- <border> </> ohraničení
- Text-aligen – zarovnání textu
- Odrážkový seznam:

- Číslovaný seznam:
- Položka seznamu:
- Tabulka: <table></table>

Tabulka je vlemi dobrý a mocný nástoj, protože umožňuje vytvořit jednoduchý web. Jako takový malý příklad můžu ukzat jeden jednoduchý kód, který netrval vytvořit ani 10 minut.

Výsledkem je toto:



Když to vidíme, tak je nám asi jasné, že jde o tabulku a není to nic složitého. V Expresion Webu by to byla urážka na 2 minuty. Kód je následující:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  <meta http-equiv="content-type" content="text/html; charset=windows-1250">
  <meta name="generator" content="PSPad editor, www.pspad.com">
  <title></title>
  </head>
  <body bgcolor="aqua">
     <table cellpadding="10">
       <tr valign="middle"><h1>Velké vědecké projekty současnosti</h1></tr>
       <tr><td><h2>články</h2>
             <br /><a href="Manhatten.html">Projekt Manhattan</a>
             <br /><a href="H.A.A.R.P.html">Projekt H.A.A.R.P.</a>
           <td>Největší rozmach vědy nastal v období těsně před
            <br />2. světovou válkou, jejím průběhu a v průběhu studené války.
            <br />V tomto období se za vědu a výzkum utratily takové částky,
            <br />že je to značná část dnešního státního rozpočtu.
            <br /><img src="12952163.jpg" alt="12952163.jpg, 15kB" title="12952163" border="0" height="300" width="410"></td>
           <td><h2>odkazy</h2>
            <br /><a href="http://www.cas.cz">Akademie věd ČR</a>
            <br /><a href="http://tn.nova.cz/magazin/historie/kalendarium/projekt-manhattan-znamenal-vstup-lidstva-do-jaderne-ery.html">Manhatt
            <br /><a href="http://www.seti.org/">SETI</a>
            <br /><a href="http://czechforiter.cz/cz">I.T.E.R.</a>
            <br /><a href="http://www.iter.org">I.T.E.R. - domovská stránka</a>
            <br /><a href="http://www.eli-beams.eu/cs/evropsky-projekt/">e.L.I</a></td></tr>
         </table>
  </body>
</html>
```

16.9. Rady pro usnadnění práce

1. Všechny soubory spojené se stránkami si přesunout již před tvorbou do jednoho adresáře a podadresářů
2. Domovskou stánku nazvat index.html
3. Pro vytvoření rozložení užít tabulku (pro jednoduchý web)
4. Pro každý článek užít novou stránku, kterou odkazem spojíme s index.html (nezapomenout na zpětnou vazbu)
5. Cesty k souborům musí být relativní
6. Užít interní stylopis, nebo CSS
7. Pokud potřebujeme nápovědu, tak užijme weby: **http://www.stranektvorba.cz/** a **http://www.jakpsatweb.cz**

17. Programování

Konečně je to tady, po více jak 400 stranách je tu kapitola, která je úvodem do problematiky, na kterou většina z vás, tedy hlavně kluků, čekala. Nevím proč, ale programování je rozhodně jednou z nejpřitažlivějších oblastí informačních technologií, ač spolu se sítěmi patří k nejtěžším problematikám. Chtěl bych jen na úvod říci, že v této a další kapitole si ukážeme základní principy a vysvětlíme si některé ohledy na tvorbu programu. O programování existují statisíce publikací, které čítají stovky a často i více jak tisíc stran, proto není možné od tohoto přehledu čekat, že z vás udělá programátory. Tato kniha vám má pomoci pochopit princip a dát základy od dvojice jazyků v takové podobě, aby bylo možné tyto jazyky oba, nebo některý z nich dále rozvíjet studiem na patřičné škole či za pomocí patřičné literatury.

Je tvůrčí činnost, která vytváří nástroj pro řešení skupiny podobných úloh (problémů). Programování se snaží napodobit realitu okolního světa za pomocí několika různých pohledů. První pohled klade důraz na postup řešení, tedy algoritmus a je nazýván strukturované programování. Druhý prohled se zaměřuje prvotně na data, kdy jej nazýváme objektově orientované programování (OOP).

Může se provádět na několika úrovních, od strojového kódu (programování procesorů), přes nižší jazyky (Assembler), které programují mikrokontrolery a automaty až po vyšší jazyky (Java a C), což je takzvané objektově orientované a strukturované programování, které slouží k psaní aplikací pro OS a částí OS.

Program je algoritmus přepsaný do programovacího jazyka. Programovací jazyk udává pohled, který musíme mít při tvorbě programu, strukturované programování zastupuje jazyk C, OOP zastupuje jazyk Java. Existuje ale jeden jazyk, který umožňuje psát programy v obou pohledech, a ještě užívat další možnosti, tímto jazykem je C++. Běžně se užívají, krom již jmenovaných, jazyky Basic, C# a Python. Na tomto místě můžu prozradit, že se dále budeme zaobírat základy jazyka Java a C++.

Tato kapitola vyžaduje jistou podobu samostudia, tedy to, že se budete pokoušet sami vyřešit nastíněný problém a přepíšete si ukázkové kódy, se kterými bude následně aktivně pracovat a zkoušet je upravovat, aby došlo k jejich pochopení.

17.1. Definice problému

Tuto kapitolu bych začal jednou podstatnou informací a to tou, že většinou je programátor sám tvůrcem zadání. Programátor, nebo skupina programátorů, dostane od zákazníka požadavek na výstup programu a většinou i možnosti vstupu do programu. Na programátorovi samotném je, aby si definoval problém a rozebral jej.

Nežli začnu psát program, tak musím sám definovat problém, který daný program bude řešit, dále tento problém musím zobecnit, pokud jsem to nezvládl rovnou. Musím stanovit požadované vstupy, popřípadě musím vymyslet, jak upravit vstupy, které mám k dispozici od zákazníka na ty, které potřebuji. Následně musím, pokud mi to nedefinuje zákazník, definovat, jak bude vypadat výstupu, zda to bude soubor, když tak jakého formátu, číslo, matice, text, či graf. Dále musíme vymyslet, jak program bude reagovat na chybu, chybný vstup (jiný formát), vnitřní chyba (nedefinovaný stav), nepřiřazení prostředků od OS, přerušení činnosti (restart OS způsobený výpadkem napájení) a spousty dalších možných chybových stavů.

Dalším krokem je nastínění problému. Tento krok je v praxi bližším rozborem problémů a přípravou všech podkladů pro psaní programu. Hlavně u výpočetních aplikací je potřeba asistence matematika, nebo alespoň nějaké tabulky. Nevýhodou této práce je, že programátor nemusí sám aktivně zvládat všechny oblasti problému. Většina problémů je příliš rozsáhlých a složitých na to, aby byly naprogramovány jako jeden blok, většinou je nutné problém rozdělit na více dílčích jednoduchých problémů, které se snadno napíší a z těchto bloků sestavit

program. Pokud píši OOP, tak musím navrhnout komunikaci mezi objekty. Pokud máme všechny poklady připraveny, tak musíme zapsat algoritmus.

17.2. Algoritmus

Pod tímto podivným slovem se ve skutečnosti ukrývá obecné řešení problému. Algoritmy se vyskytují všude kolem nás, takovým typickým příkladem jsou matematické a fyzikální vzorce. Jeho přepis do programovacího jazyka tvoří program. Algoritmus jako takový je nezávislý na programovacím jazyku.

Definice: systém jednoznačných pravidel umožňující řešit problém z nějaké skupiny podobných problémů v konečném počtu kroků

Algoritmus řeší skupinu podobných problémů a nesmí dojít k jeho zacyklení i když je možné, že algoritmus bude obsahovat cyklické části. Speciálním případem algoritmu je algoritmus lineární, který neobsahuje žádné cykly.

17.2.1. Vlastnosti algoritmu

- **Determinovanost**
  Neumožňuje jiný výklad (jednoznačnost, přesnost)
- **Hromadnost**
  Kroky jsou řešením pro skupinu úloh
  Neobsahují konkrétní vstupní hodnoty (Vstupní data se mění)
- **Rezultativnost**
  Činnost je konečná
  Musí dát vždy nějaké výsledky
  Nesmí dojít k zacyklení
- **Efektivnost**
  Spotřeba času/paměti
  Finanční náročnost spojená s vývojem/provozem

17.2.2. Zápis algoritmu

- **Mluvenou řečí**
  Přirozená člověku, ale nejtěžší na vystižení
- **Grafické vyjádření**
  Vývojové diagramy, strukturogramy, jazyk uml
- **Pseudojazyky**
  Kombinace běžné mluvy a programovacího jazyka (mluvené slovo a příkazy programy jazyka)
- **Podprogram či programem**
  Je zvlášť řešený úsek problému, hrozí jeho opakování
  Voláme jej jménem do celkového programu

17.2.3. Vývojové diagramy

Jsou jedním z typů grafů, které slouží k popsání programu. Pokud programujeme klasicky, nikoli paralelně asynchronně, tak jsou i nejvhodnějším nástrojem pro popis algoritmu, který předchází samotnému přepisu do programovacího jazyka a je možné na nich vyzkoušet i funkčnost algoritmu.

Pro popis se užívají normované značky, do nichž se píší názvy proměnných, hodnoty konstant a operace s nimi. Značky se spojují tzv. orientovanými čarami, neboli šipkami, ve směru postupu provádění operací. Diagram je zakreslován vždy zora dolů a je možné zde graficky provést i rozložení na podproblémy a dané dílčí problémy rozkreslit samostatně.

- Zpracování – jednoduché příkazy přiřazení, či vpis 
- Rozhodování – podmínky 
- Předdefinovaná činnost – volání podprogramu 
- Vstup/výstup – čtení (klávesnice)/tisk (monitor)
- Spojka – vpisuje se číslo (návaznost) 
- Složená závorka, poznámka/vysvětlivka {}
- Mezní značka (začátek a konec) 

17.3. Výraz

Většina z nás zná matematický výraz. V programování je výraz něco podobného, jako v matematice rovnice, ale zdůrazňuji, že jen podobného! Výraz přiřazuje proměnné (konkrétnímu místu v paměti) nějakou hodnotu, která může být zadána buď přímo číselně, nebo vznikne vyhodnocením složitějšího výrazu. Výraz je vyhodnocován dle priorit operátorů, tedy podobně, jako když provádíme výpočet v matematice, násobení má přednost před sčítáním a podobně. Obdobně jako v matematice jsou přednosti upravovány závorkami.

Příkladem může být třeba výraz: c16 = **c3+c7*(c1-c4) . S**ymbol, který v matematice znamená rovnost, tak v programování čteme jako přiřaď. Rovnost se ve většině programovacích jazyků zapisuje symbolem == .

17.3.1. Podmínka

Je logický výraz (má hodnotu pravda/nepravda – true/false) a slouží k rozdělení kódu do větví či k opakování části kódu po nějakou danou dobu. Užívají se logické operátory (and, or, not, nand, xor, xnor) a relační operátory (<>=). Podmínka jako taková ovšem nestačí, odmínka se vždy kombinuje s příkazem podmínečné činnosti (větvení kódu), tedy příkazem if, nebo s příkazy tvořícími cyklus, tedy příkazy while a do while, či Repeat until.

Příklady:

A>100&B>100&C>100

negace: A< =100 OR B < =100 OR C< = 100

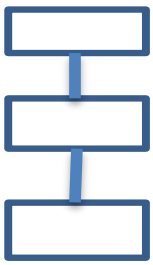
Cmod2=0 ==> C je sudé

(x<>0) OR (y<>0) ==> Bod neleží v počátku os x/y

17.4. Příkazové struktury

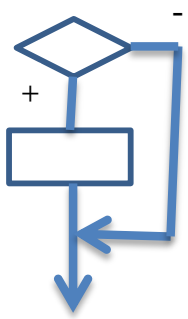
Jsou předepsané konstrukce (schémata) pro větvení a cykly, kterým odpovídají konkrétní příkazy daného jazyka. Tyto struktury jsou nezávislé na jazyku a jsou klíčem k pochopení funkce příkazů a také vývojové diagramu.

Nejjednodušším zástupcem je posloupnost, tedy přímí sled příkazů bez větvení a cyklů.



Dalším zástupcem je neúplné větvení, které se reprezentuje příkazem if, či v pascalu if then.

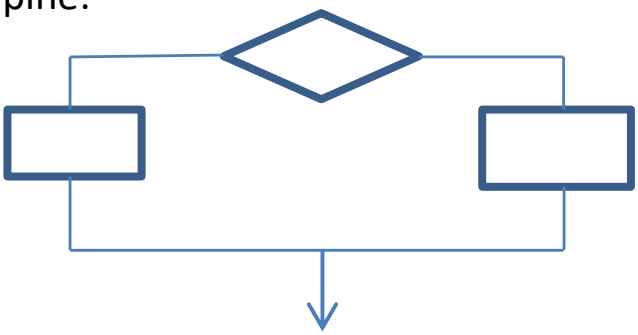
neúplné:



If-then
Platí +
Neplatí -

Naproti tomu větvení úplné dělí kód do dvojice větví. Bývá reprezentováno příkazy if else, či if then else.

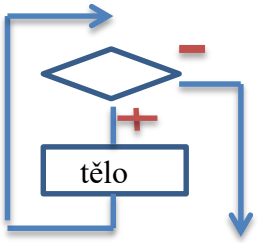
-úplné:



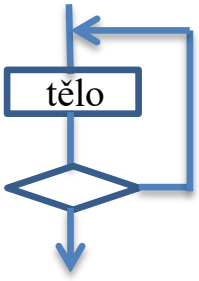
If-then-else

17.4.1. Cykly

První skupinou jsou cykly s testem na začátku, kdy se testuje podmínka a pokud je splněna, tak se přejde k příkazům těla cyklu, pokud není, tak je cyklus vynechán. Pokud podmínka přestane platit, tak je cyklus ukončen. Takovéto cykly jsou reprezentovány příkazem while.



Druhou skupinou jsou cykly s testem na konci, jejichž tělo se vždy provede minimálně jednou. Podmínka zde slouží k ukončení cyklu. Takovéto cykly jsou reprezentovány příkazy do while či repeat until.



Třetí skupinou jsou cykly s předem známým počtem opakování, které jsou reprezentované příkazem for. Takovéto cykly obsahují počítadlo a jsou speciálním případem cyklů s testem na konci.

17.5. Sestavení programu

Sestavení programu již vyžaduje jako první z kroků jeho tvorby znalost programovacího jazyka. V tomto kroku dojde k přepsání algoritmu a definic vstupů a výstupů do programovacího jazyka. Po přepisu pak již vývojové prostředí provede překlad do binární podoby spustitelné v počítači.

17.6. Ověření správnosti

Ověření správnosti probíhá ve dvou blocích. První blok provede již překladač při vytváření spustitelného souboru. V tomto bloku stroj ohlásí případné chyby v příkazech, které mohli vzniknout při přepisu algoritmu do programovacího jazyka.

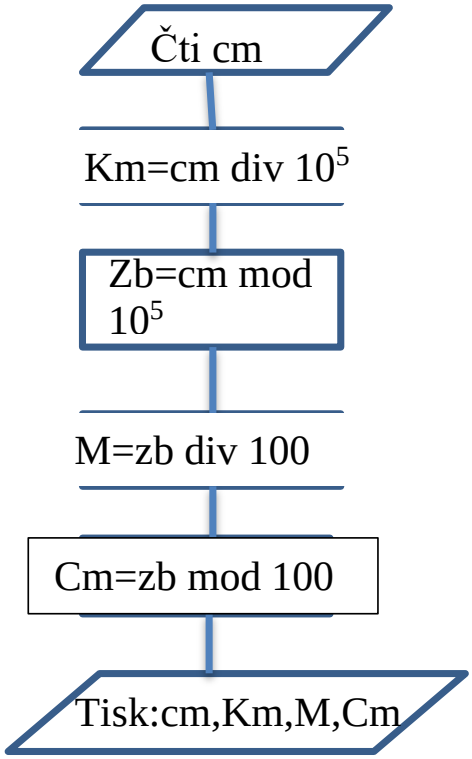
Druhý blok provádí programátor, nebo specialista, který se označuje jako tester buď ručně, nebo automatizovaně pomocí jiného prográmku. Cílem tohoto bloku je ověřit správnost výsledků, najít neošetřené chybové stavy a odladit aplikaci do podoby, ve které bude nejlépe fungovat. I pro tyto účely jsou dostupné pomocné programy.

17.7. Příklad návrhu programu

Definice problému

- Přepočet cm na km, m, cm
- Vstup celé kladné číslo (cm)
- Vytiskni výsledky a zadanou hodnotu

Vývojový diagram



Kód v jazyce java:

```
Pro jednoduchost bez čtení.
public class prevod2
{
   public static void main(String[] args)
   {
     int x,km,m,cm,zb;
     x=129628;
     km=x/100000;
     zb=x%100000;
     m=zb/100;
     cm=zb%100;
     System.out.print("zadaná hodnota v cm:"+x+"\n km:"+km+"\n metrů:"+m+"centi-
metrů:"+cm);
   }
}
```

17.8. Vývojová prostředí

Vývojové prostředí je program, který umožňuje tvorbu programu od napsání kódu až po tvorbu spustitelného souboru. Vývojové prostředí obsahuje nějakou podobu textového editoru, překladač, někdy preprocesor či interpret, to již záleží na využívaném jazyku.

Na tomto místě si představíme jedeno universální prstředí, které je k dispozici jak v bezplatné verzi Expres, tak v placených verzích professional a ultimate. Ve verzi 2013 jsou všechny edice dostupné i v českém jazyce. Tímto mocným nástrojem je MS Visual studio. Dále si představíme opravdu jen velmi krátce Java editor, kterým je aplikace BlueJ. Základním vojovým prostředím pro Javu je Netbeas od společnosti Oracle, ale toto prostředí je poměrně složité a je celé v anglickém jazyce, takže se budeme držet u aplikace BlueJ, která má i českou jazykovou mutaci, pokud budeme pracovat v jazyce Java.

17.8.1. MS Visual Studio 2013

Jak již bylo řečeno, tak tento silný nástroj můžeme mít zdarma, pro stažení stačí užít adresu: http://go.microsoft.com/?linkid=9832280&clcid=0x405 a pro získání českého jazykového balíčku pak adresu: http://go.microsoft.com/?linkid=9832287 .

Požadavky na systém:

- ✓ Windows 7 SP1 (x86 a x64)
- ✓ Windows 8 (x86 a x64)
- ✓ Windows 8.1 (x86 a x64)
- ✓ Windows Server 2008 R2 SP1 (x64)
- ✓ Windows Server 2012 (x64)
- ✓ Windows Server 2012 R2 (x64)

Požadavky na hardware:

- ➢ Procesor pracující na frekvenci 1,6 GHz či rychlejší
- ➢ 1 GB paměti RAM (1,5 GB v případě používání ve virtuálním počítači)
- ➢ 5 GB volného místa na pevném disku
- ➢ Pevný disk s rychlostí 5400 otáček/min.
- ➢ Grafická karta podporující rozhraní DirectX 9 a rozlišení zobrazení 1024 × 768 nebo vyšší

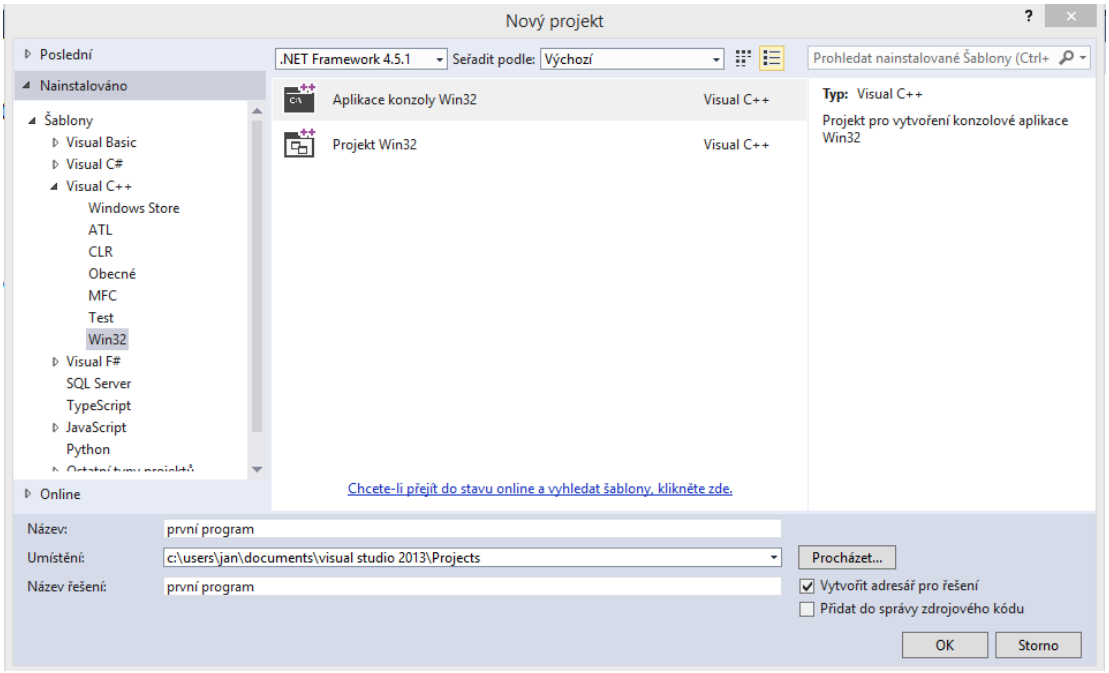
Další požadavky:

- Tato verze Visual Studia funguje nejlépe s Internet Explorerem 10 nebo novějším. Pokud v počítači nemáte Internet Explorer 10 nainstalovaný, nemusí některé funkce fungovat podle očekávání.
- Ve Windows 8.1 a Windows Server 2012 R2 se vyžaduje aktualizace KB2883200 (dostupná taky přes Windows Update).

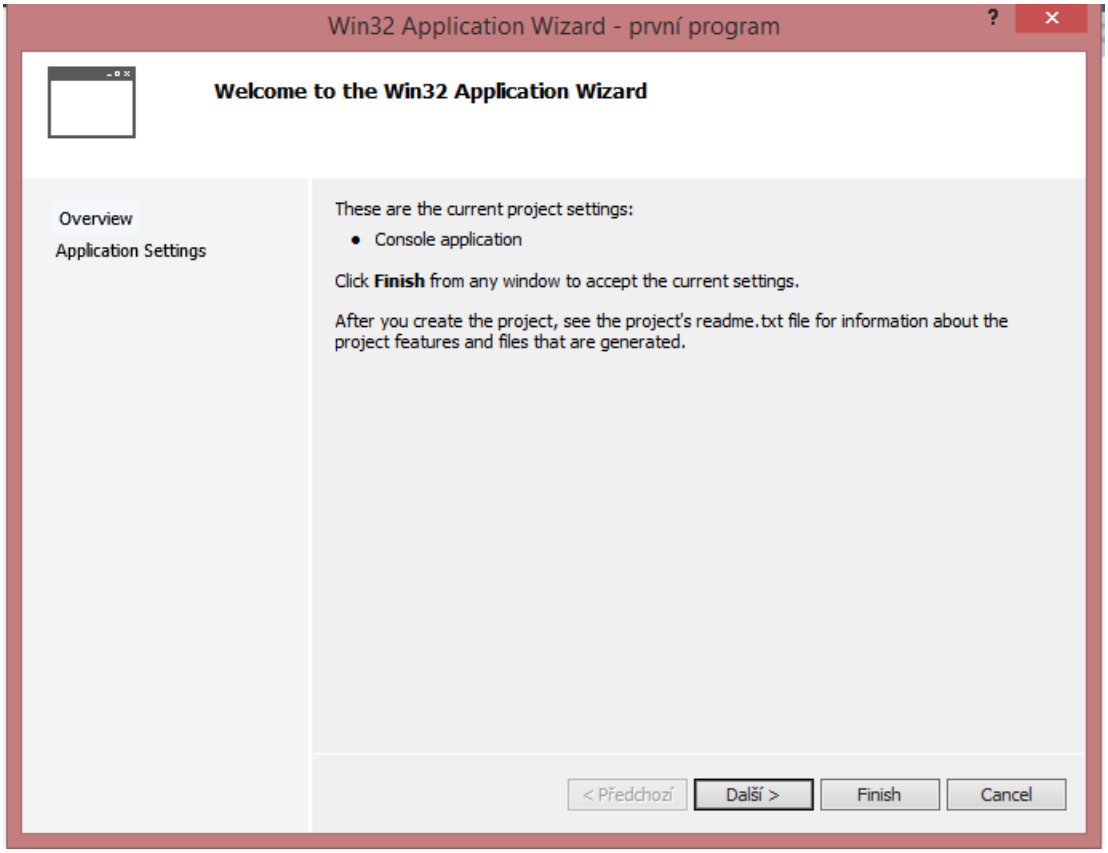
Visual Studio Express for Windows Desktop vám umožní plně využít potenciál systému Windows s návrháři XAML, produktivním integrovaným vývojovým prostředím (IDE) a řadou programovacích jazyků, včetně jazyků C#, Visual Basic a C++. Můžete volit mezi technologií WPF (Windows Presentation Foundation), Windows Forms a Win32 a zaměřit se na plochu Windows se správnou technologií pro vaši aplikaci a vaše dovednosti.

17.8.1.1. Tvorba konsolové aplikace

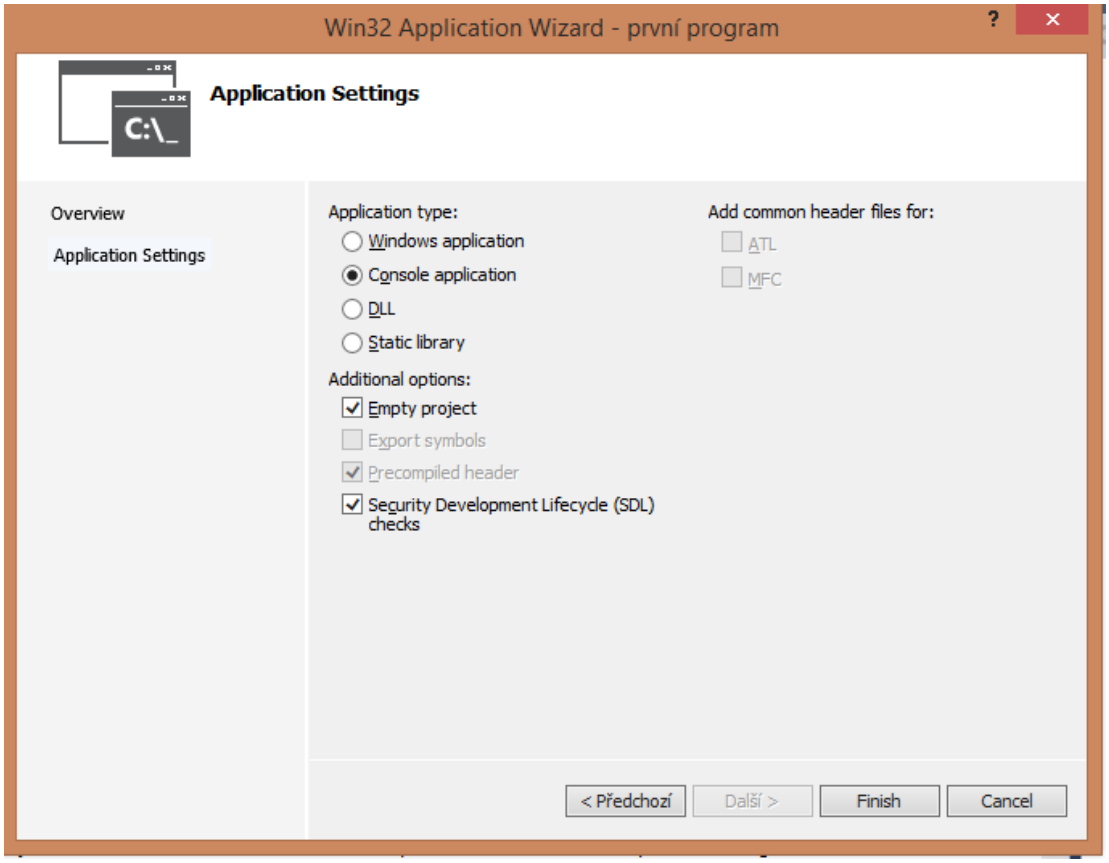
1. Soubor => nový => nový projekt
2. Šablony => C++ => Win32 => aplikace konzoly Win32



3. Pojmenovat
4. Ok

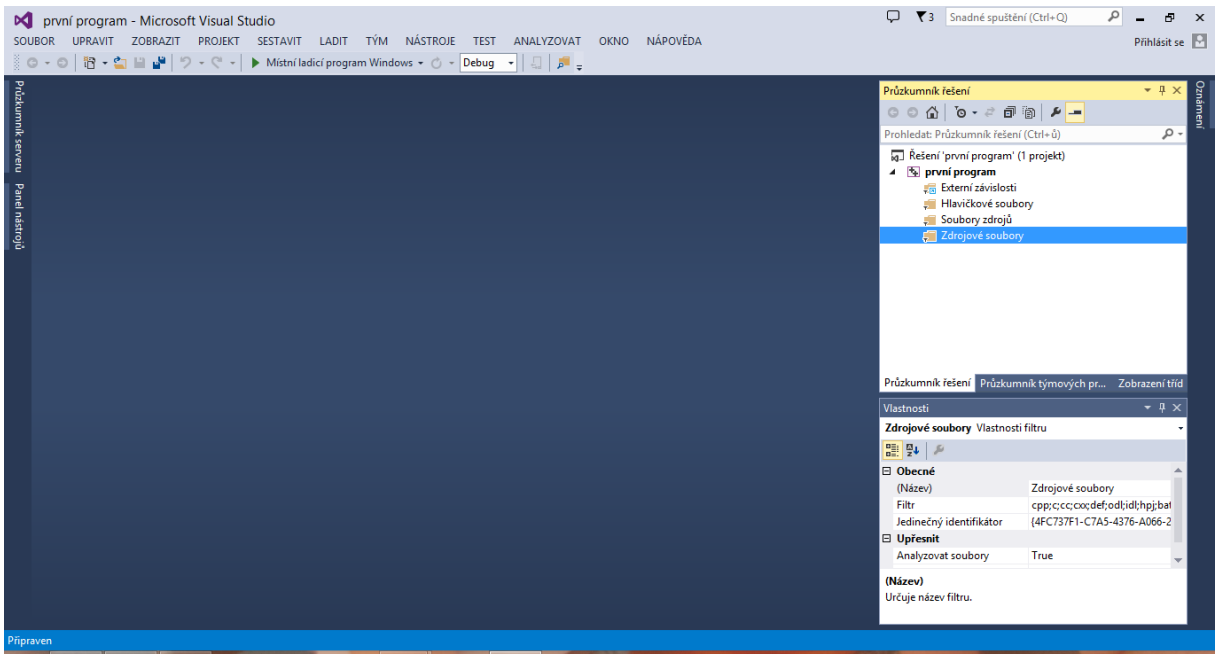
5. 

6. Další!!
7. Nastavit dle obrázku

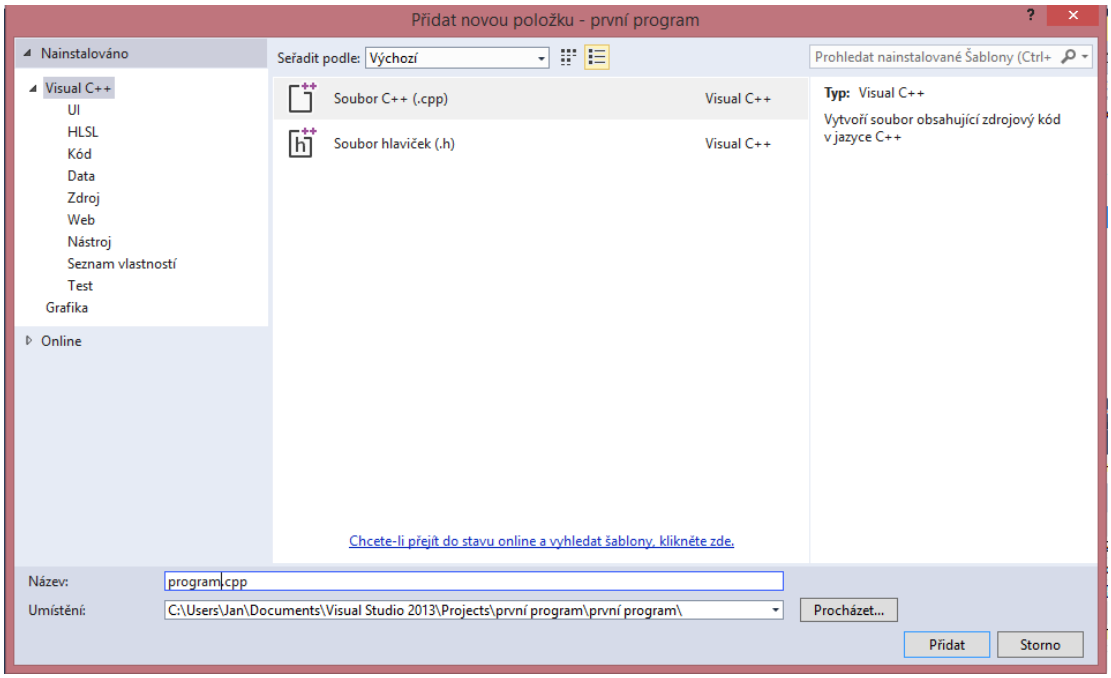


8. Finish

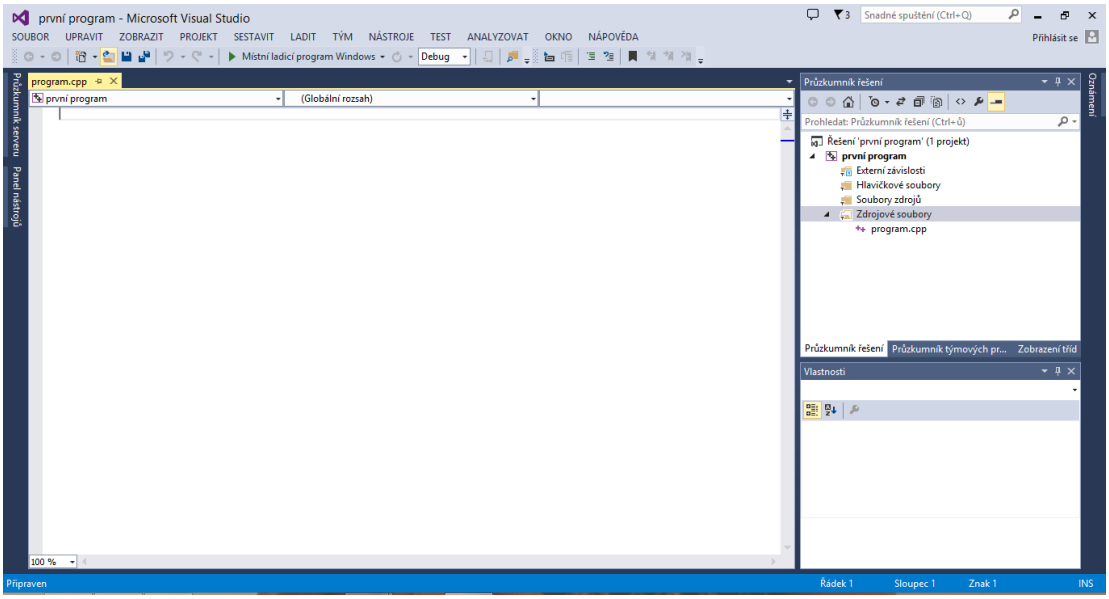
17.8.1.2. Psaní kódu



1. Klikneme pravým tlačítkem na zdrojové soubory
2. Zvolíme přidat => nová položka



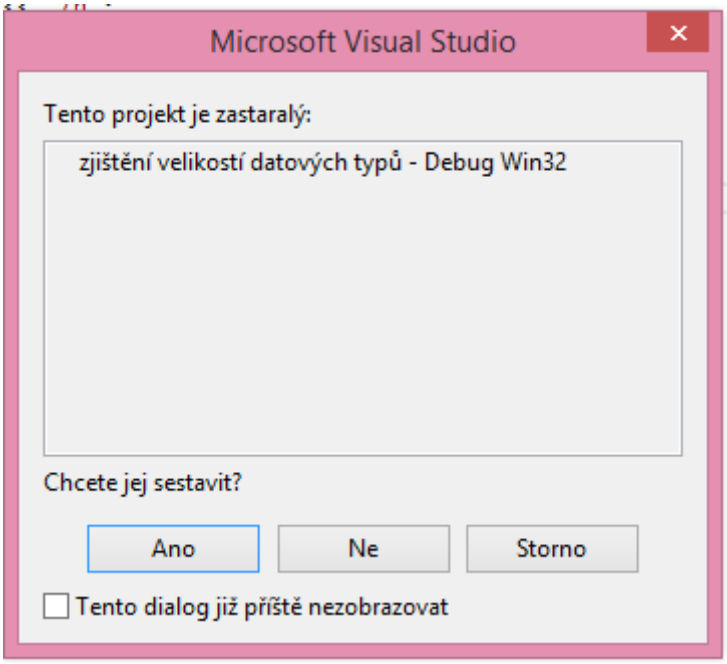
3. klikneme na soubor C++ a vyplníme název
4. Klikneme na přidat

5. 

6. Klepnout na uložit
7. Napsat zdrojový kód (text zadávaný na výstup psát bez diakritiky)
8. Průběžně ukládat
9. Uložit

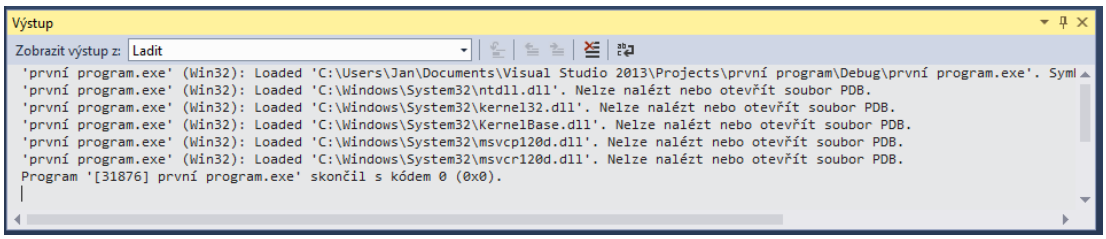
17.8.1.3. Sestavení projektu

✓ Dojde ke zpracování projektu

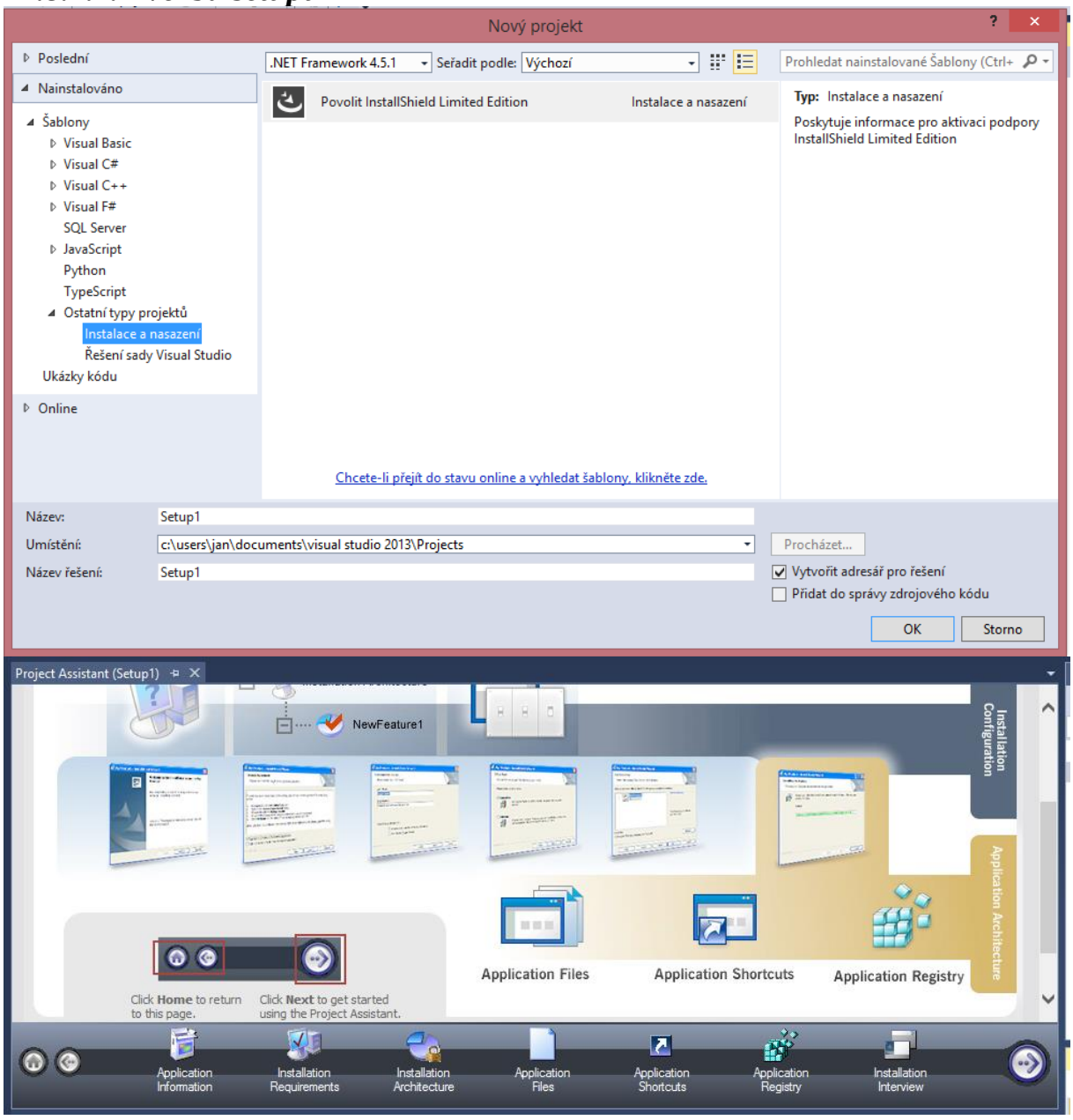
1. Uložíme vše (ctrl+s)
2. Klikneme na místní ladící program Windows
3. Zkontrolujeme, zda je v seznamu souborů modře označen soubor, který chceme kompilovat 

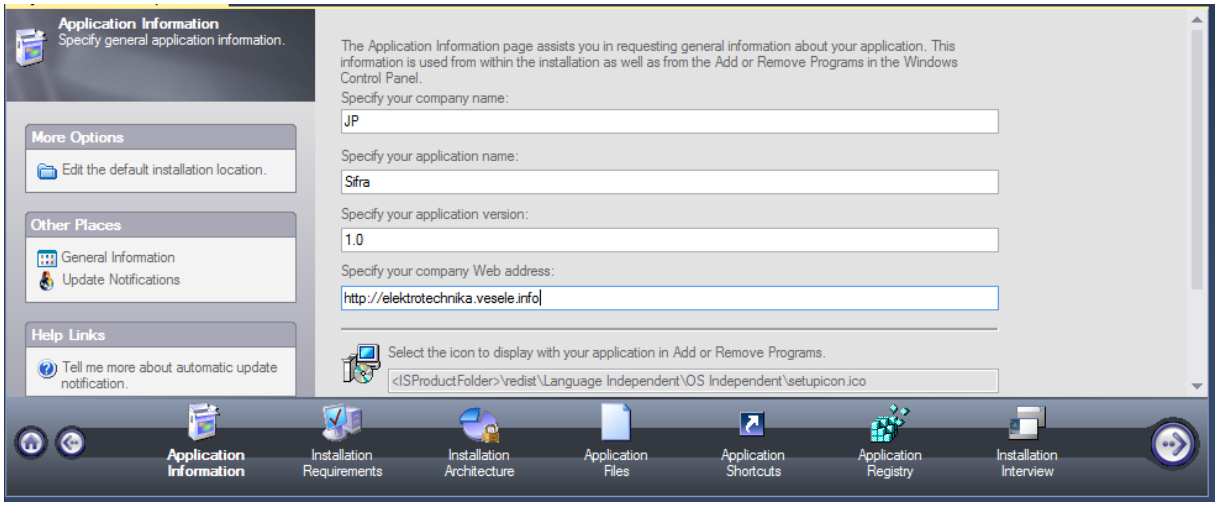
4. Dáme kompilovat

5. Vyčkáme
6. Dojde ke zpuštění našeho kódu, nebo zobrazení chyb
7. Pokud dojde k chybě, tak na chybu ve výstupu poklikáme dvojklikem a kurzor v editačním okně kódu se přesune na místo, kde se chyba nachází
8. Pročteme a vyhodnotíme log o běhu programu

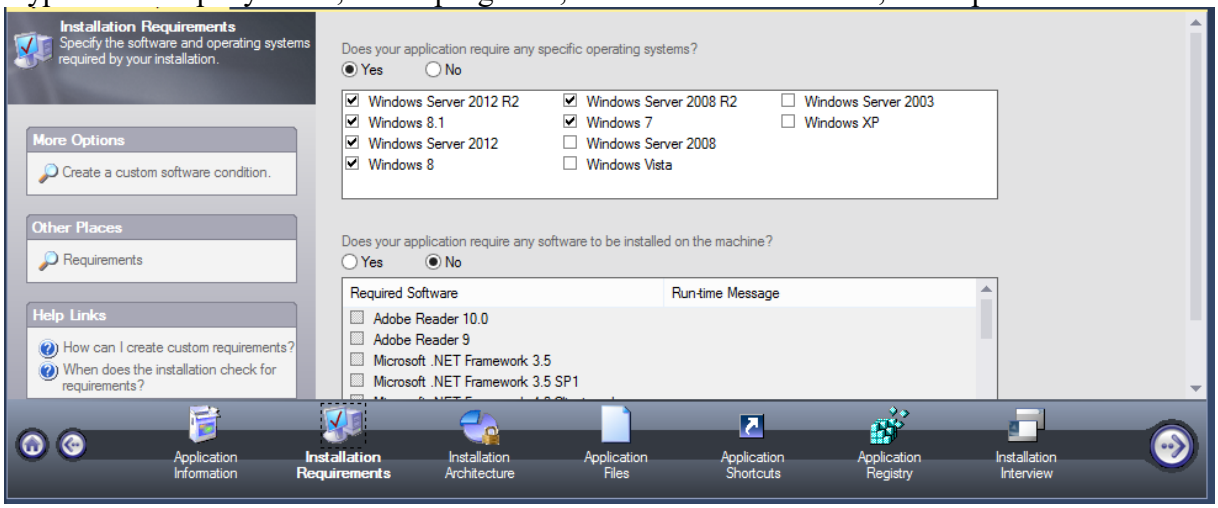


17.8.1.4. Tvorba setupu

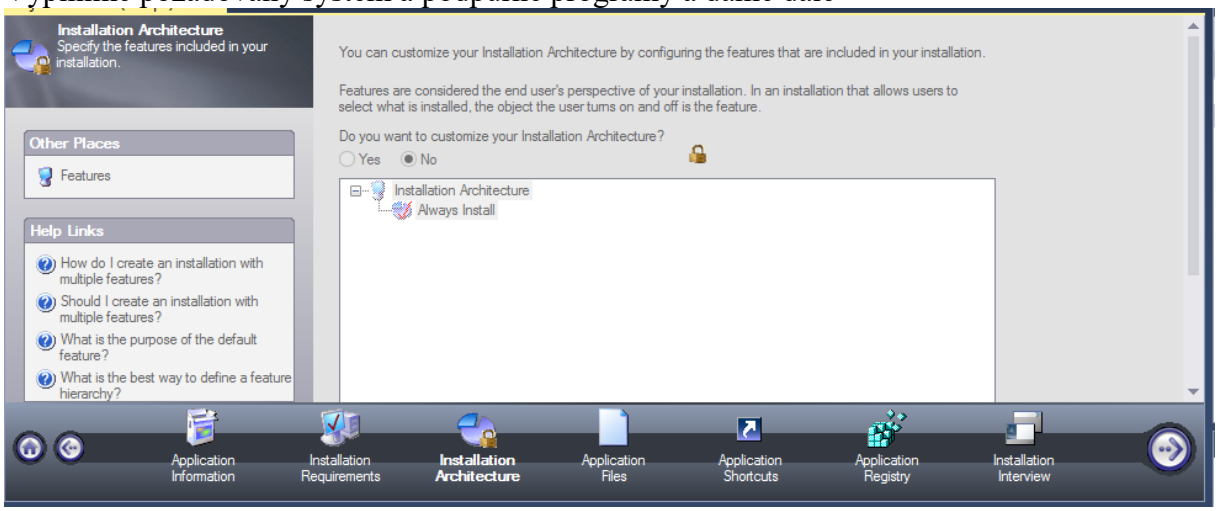




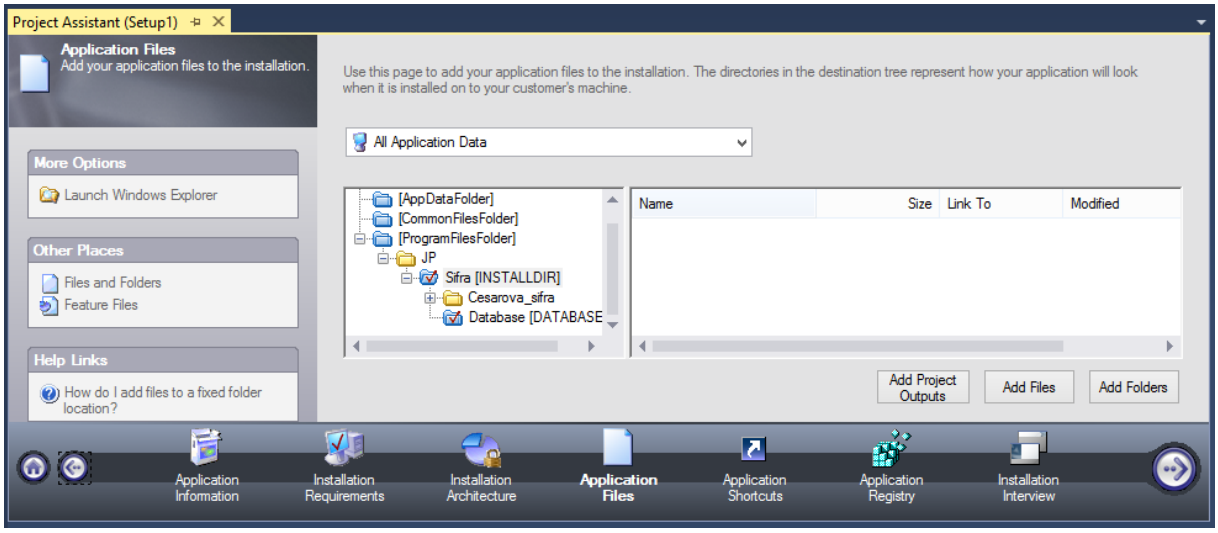
Vyplníme kompany name, název programu, verzi a webovou adresu, dáme pokračovat dále.



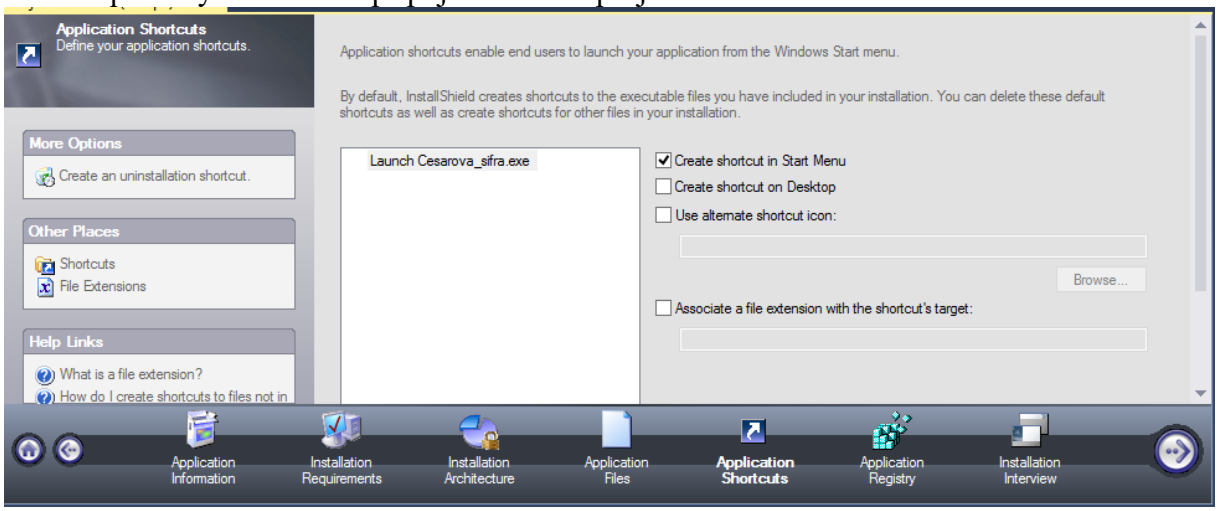
Vyplníme požadovaný systém a podpůrné programy a dáme dále



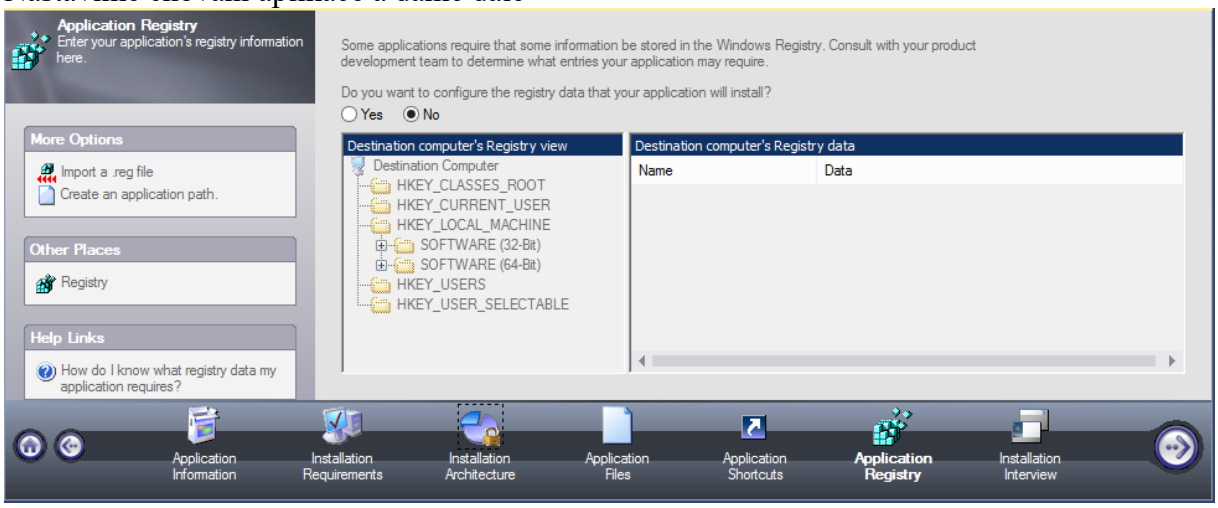
Přeskočíme krokem dále



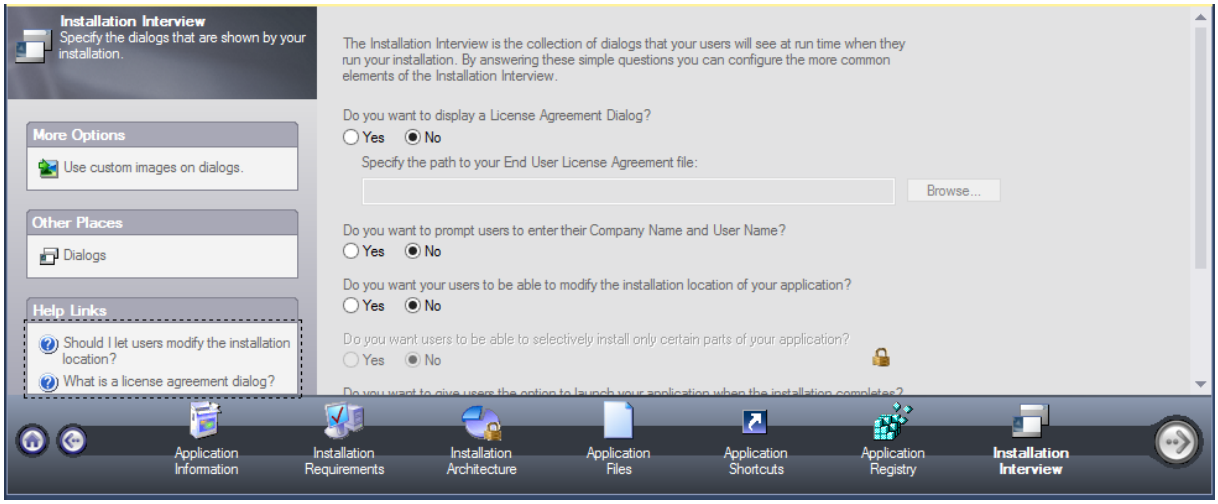
Pomocí položky add folders připojíme složku projektu visual studia a dáme dále



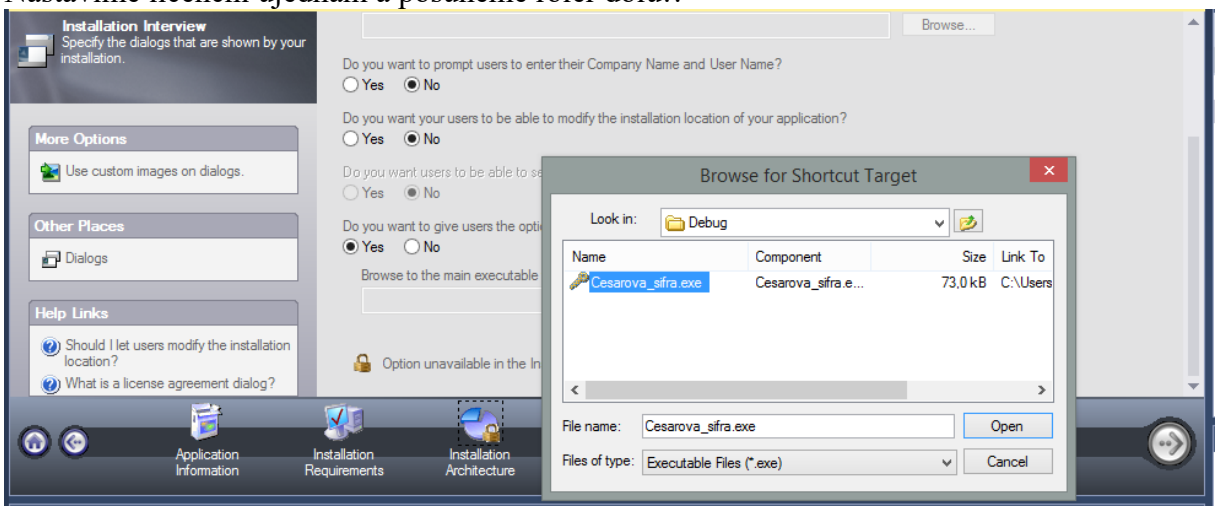
Nastavíme chování aplikace a dáme dále



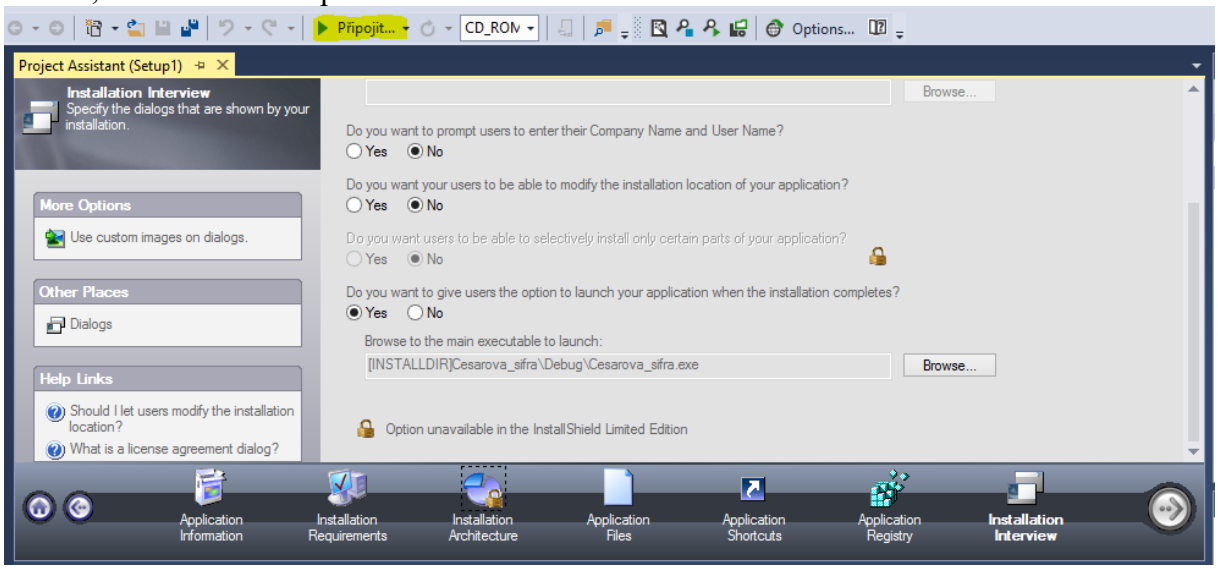
Pro běžné aplikace necháváme rozhodnout visualstudio, kam se bude zapisovat v registrech, dáme dále



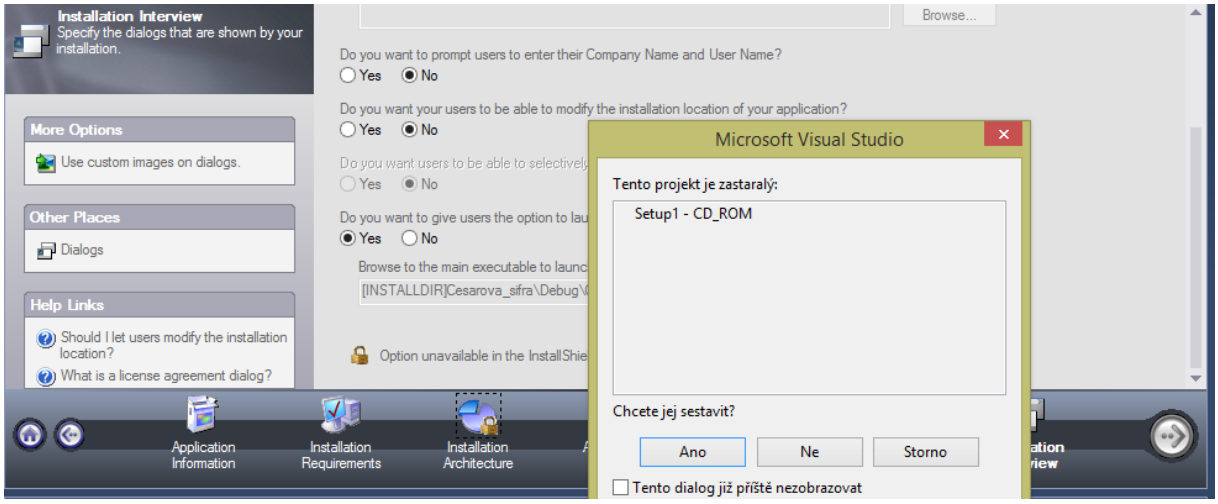
Nastavíme licenční ujednání a posuneme roler dolu!!



Nastavím dle obrázku, na místě adresáře vyvoláme browse a najdeme v položce debug náš exe soubor, následně dáme open



Zvolíme připojit

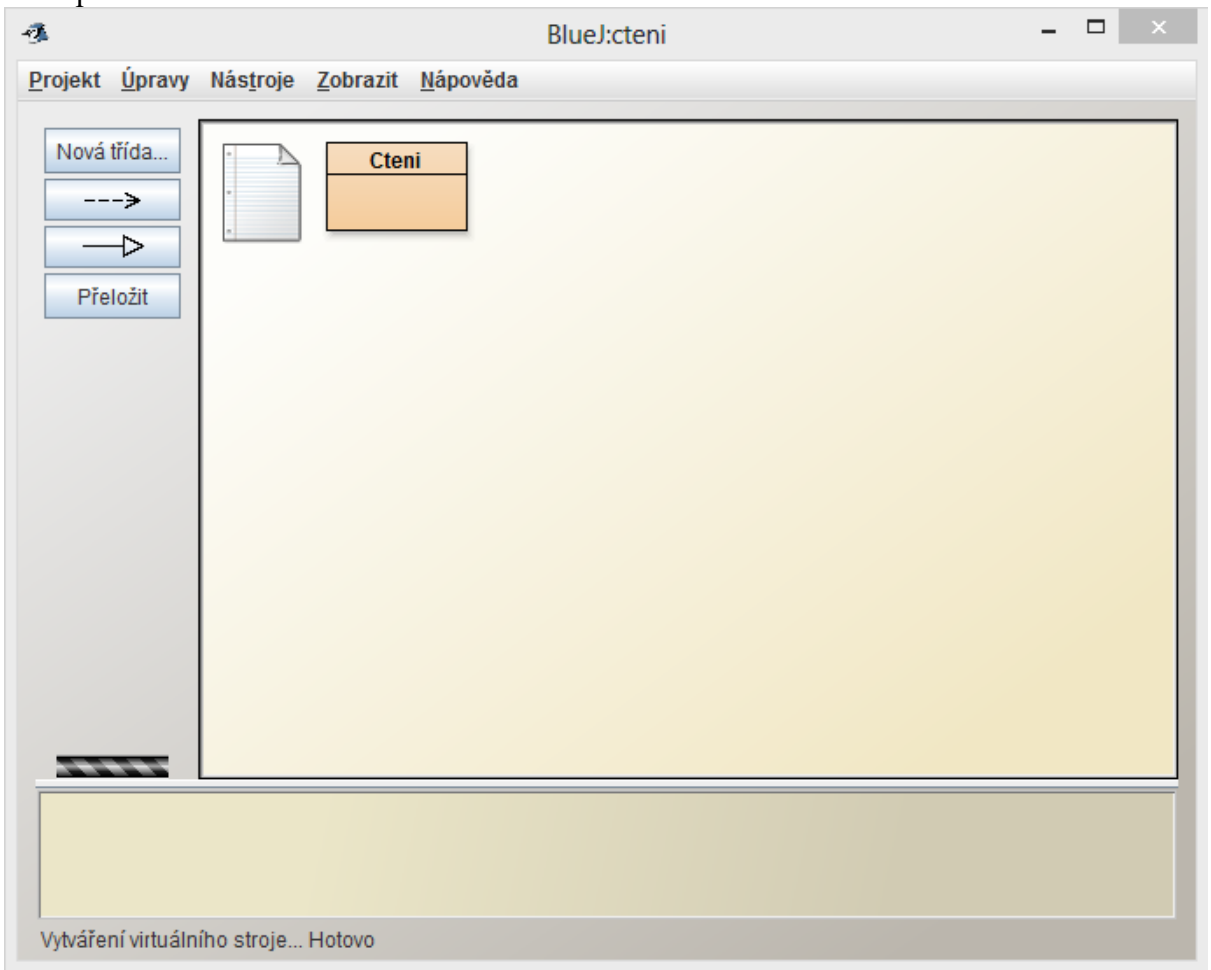


Zvolíme ano pro znovu sestavení. Následně vyhledáme instalační soubory, vše je hotovo ☺

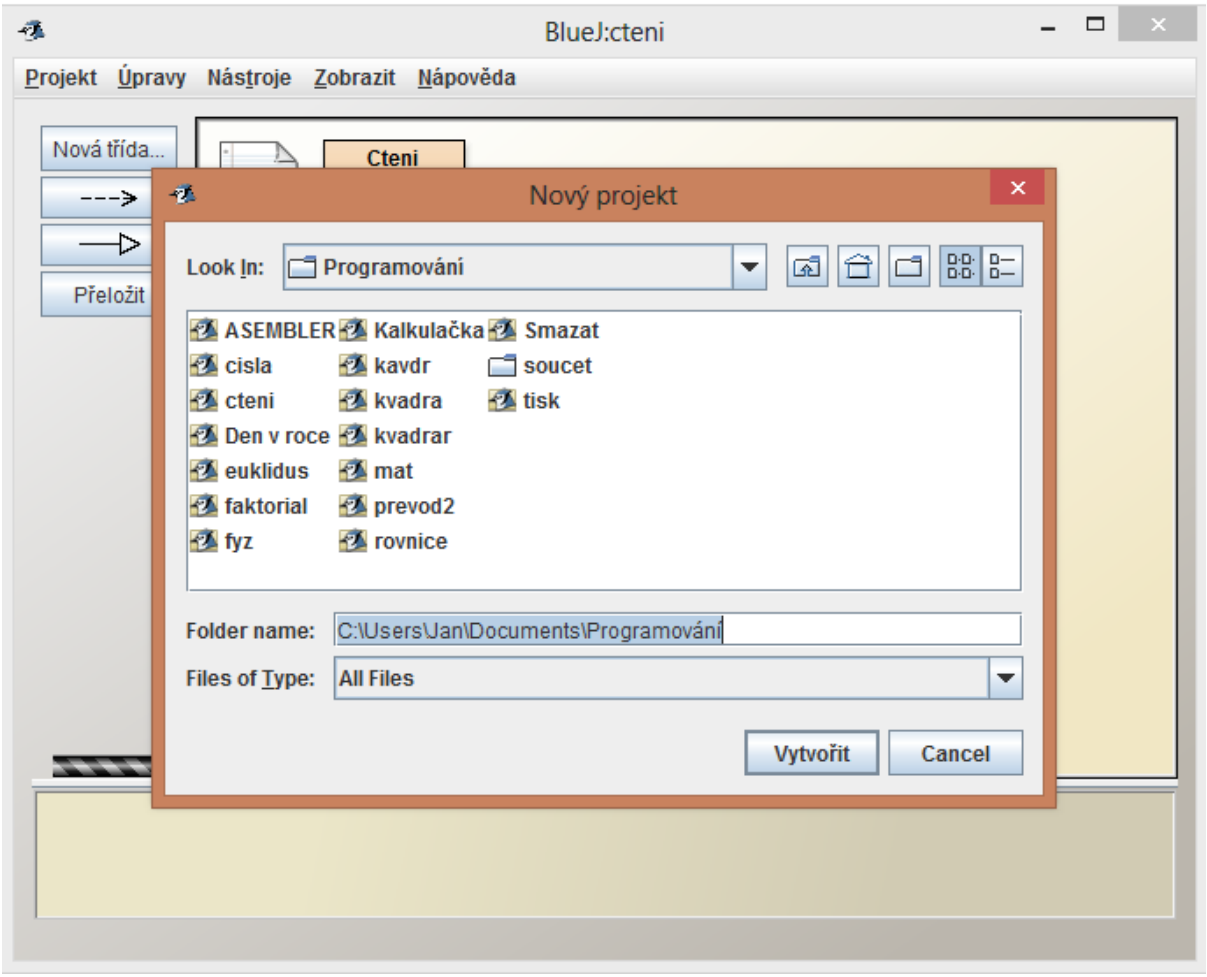
17.8.2. BlueJ

Toto vývojové prostředí je určeno pro vývoj jednoduchých školních programů v jazyce Java. Je zdarma ke stažení včetně češtiny na adrese: http://java.vse.cz/Java/BlueJInstalace .

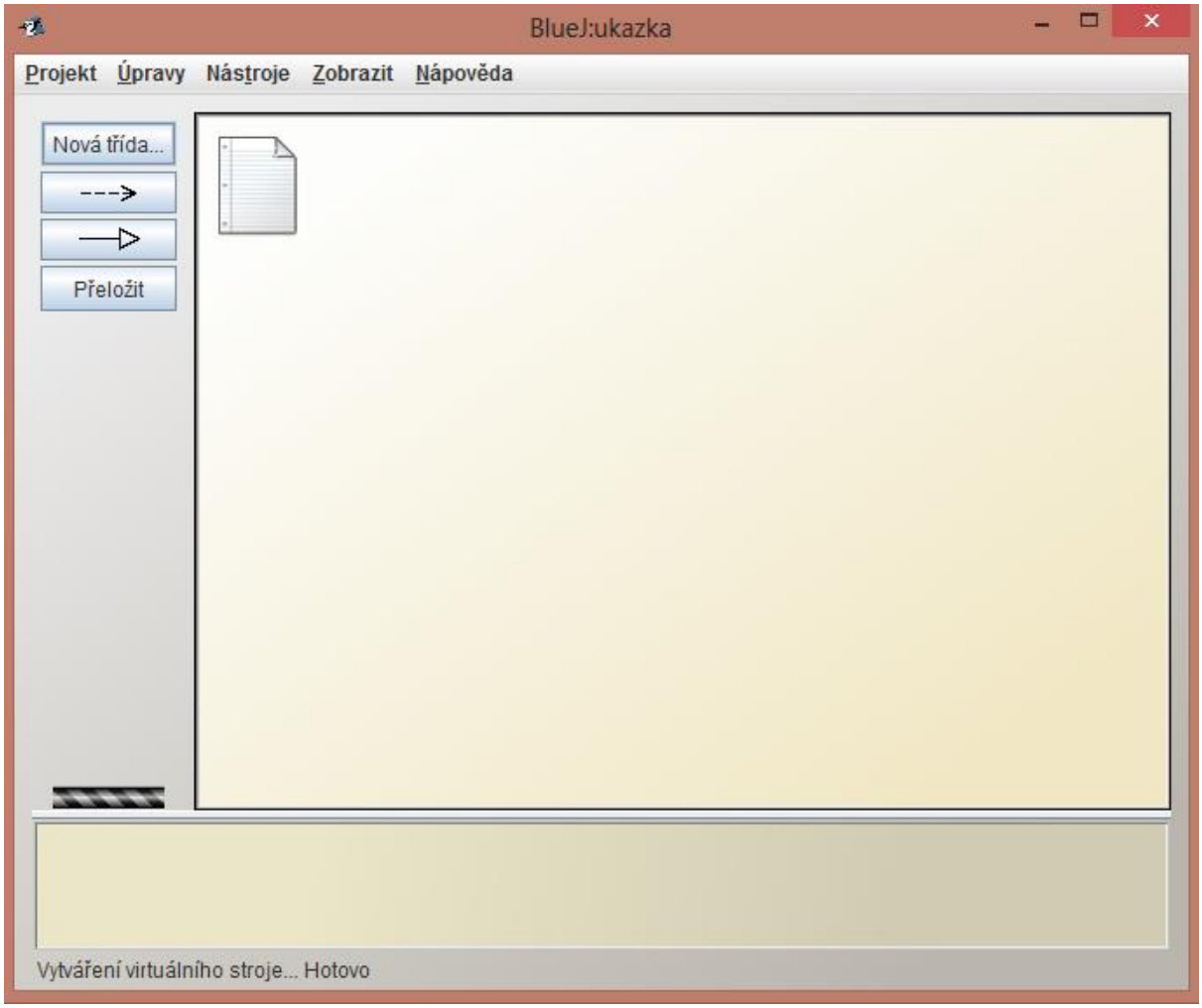
Po spuštění se objeví poslední otevřený projekt, pokud jde o první spuštění, tak zůstane okno prázdné.



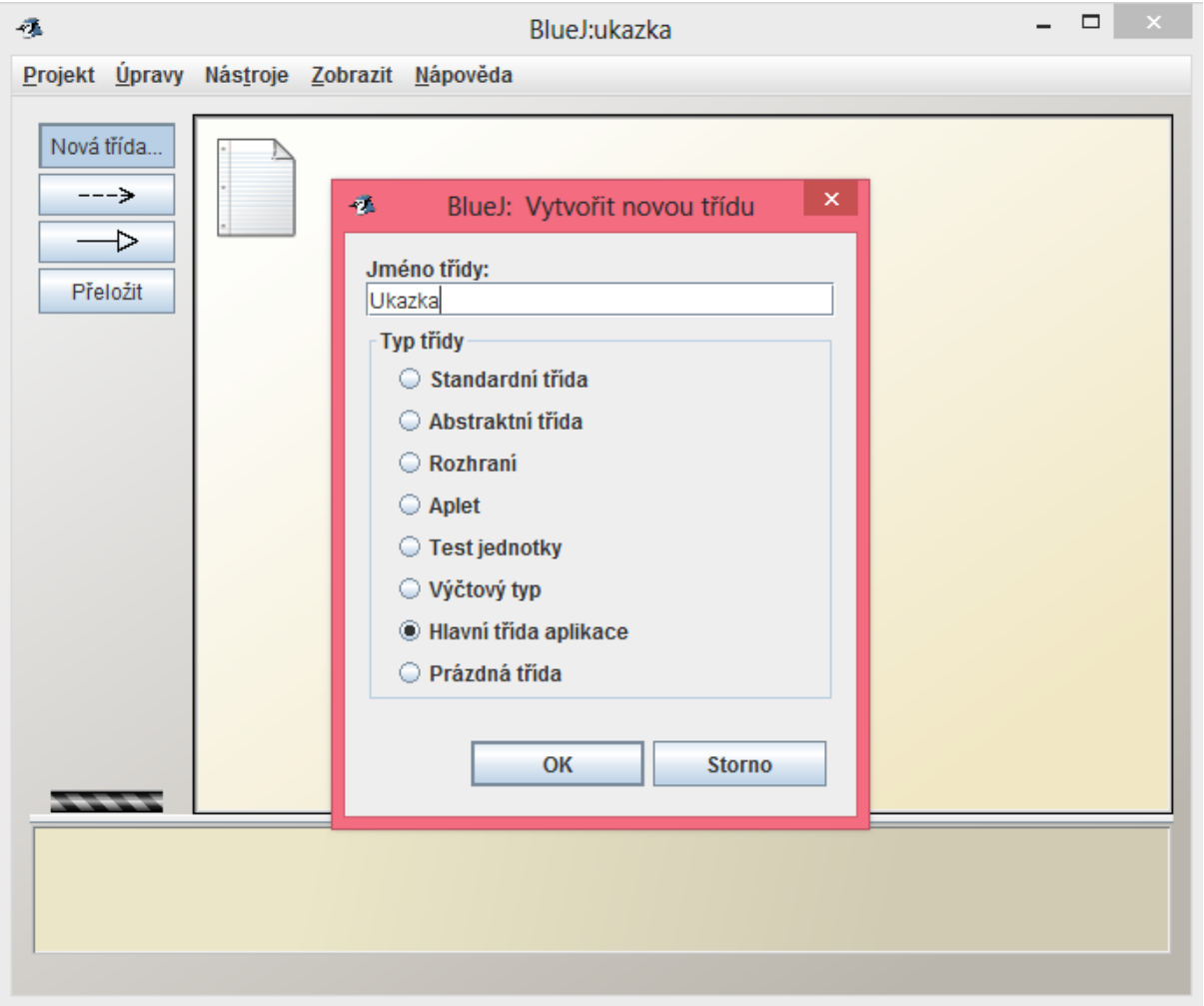
Nový projekt vytvoříme pomocí nabídky projekt a volby nový projekt.



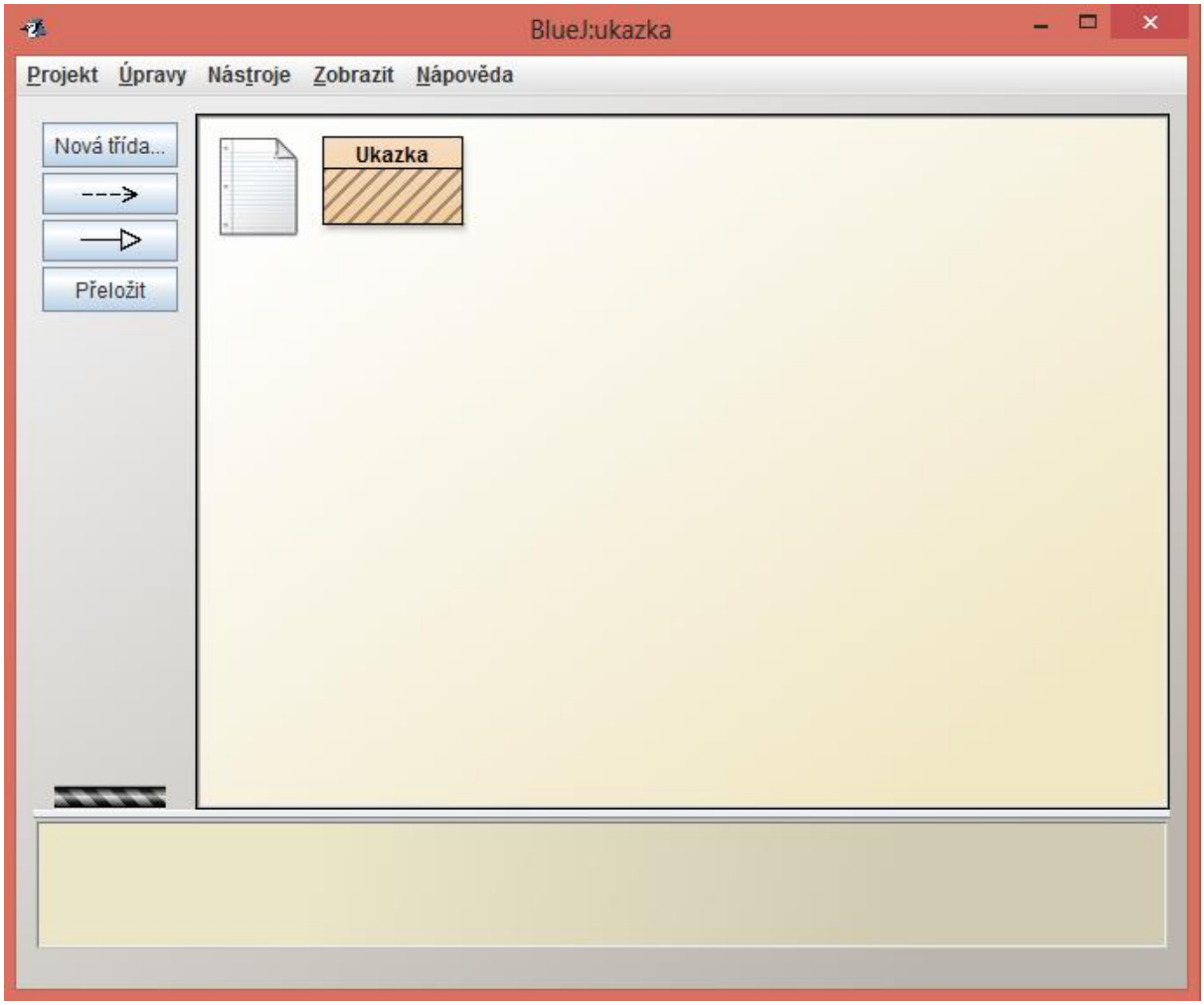
Zvolíme adresář a název projektu, který napíšeme do kolonky folder name. Vše potvrdíme stiskem vytvořit.



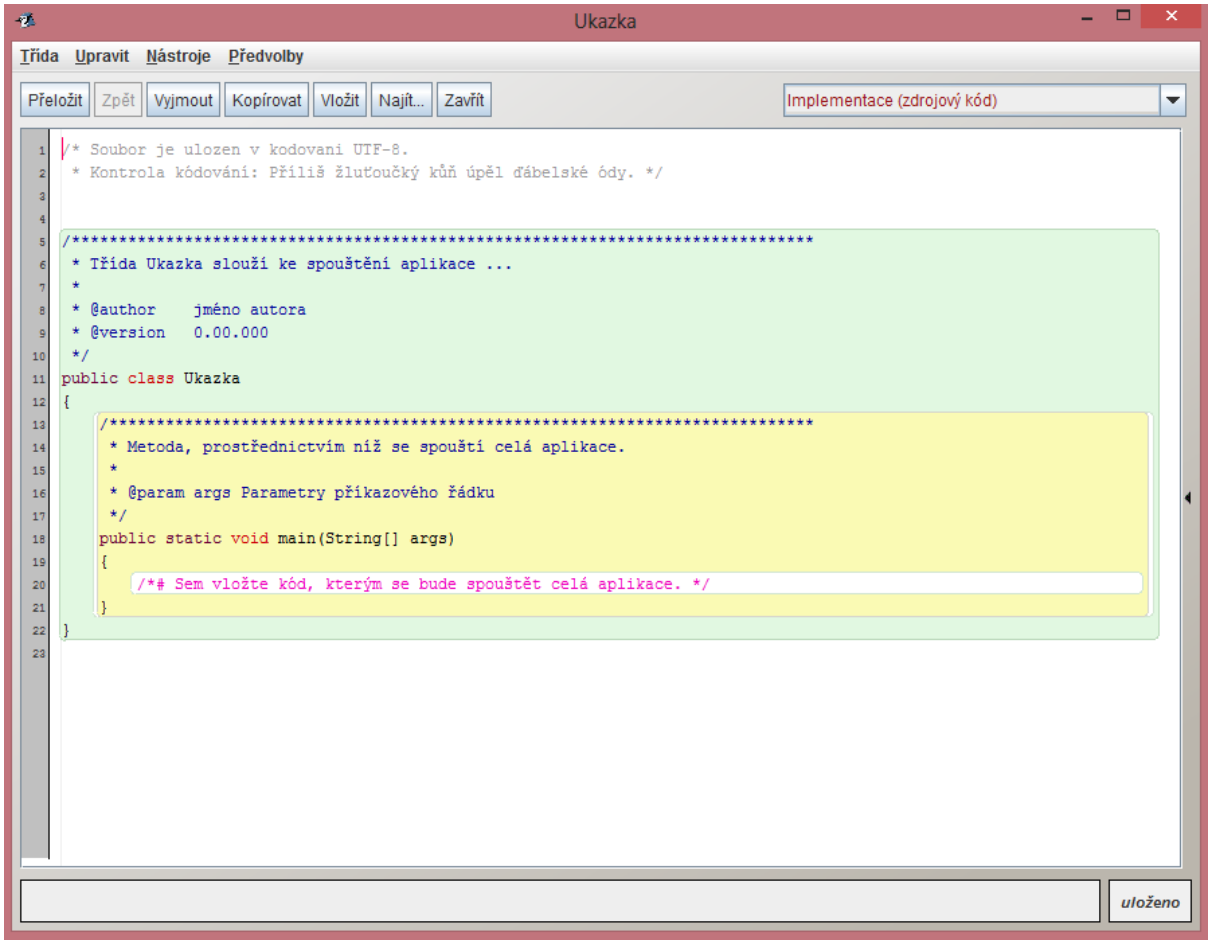
Následně stiskneme nabídku nová třída a nastavíme vše dle následujícího obrázku. Jméno třídy musí začínat velkým písmenem a musí být shodné s názvem projektu.



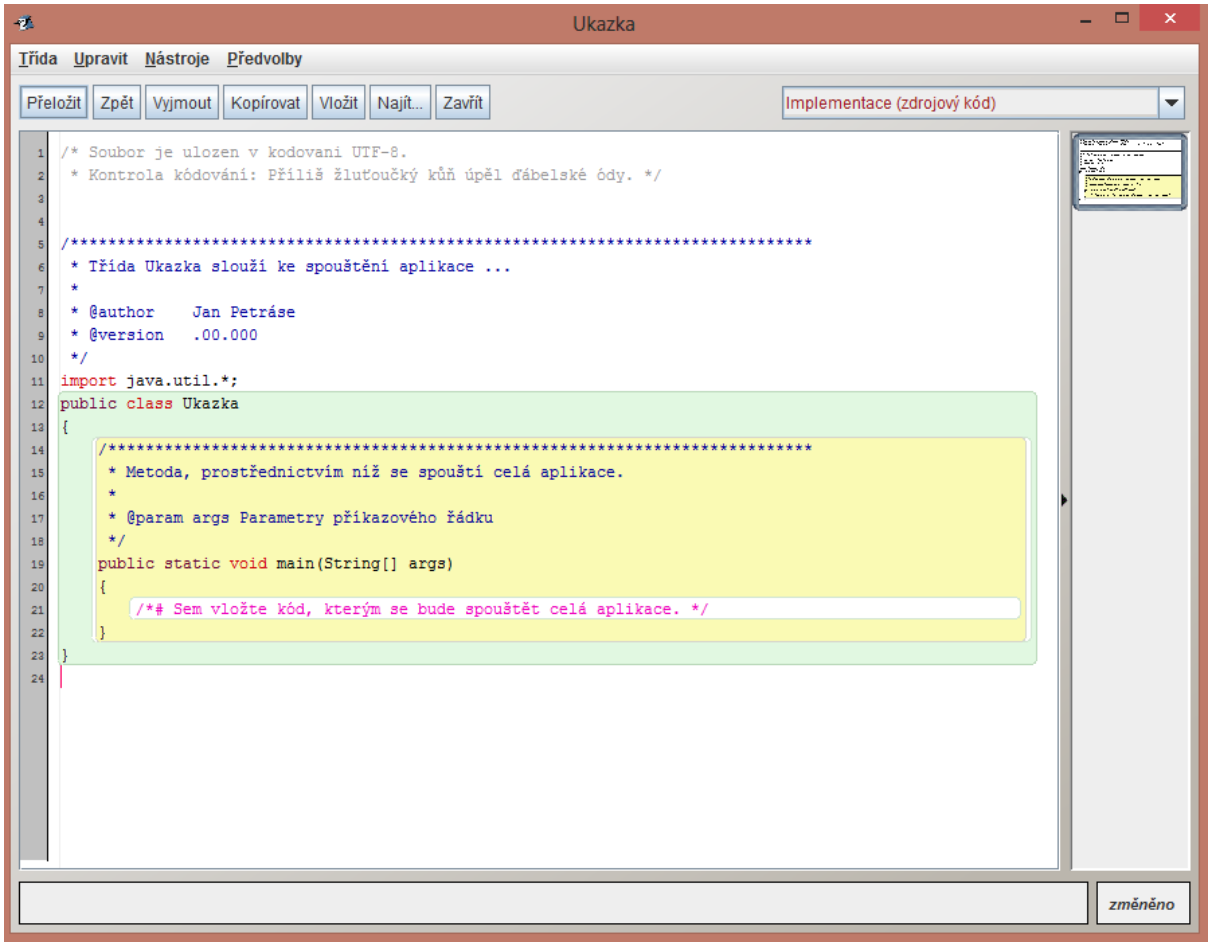
Potvrdíme stiskem OK.



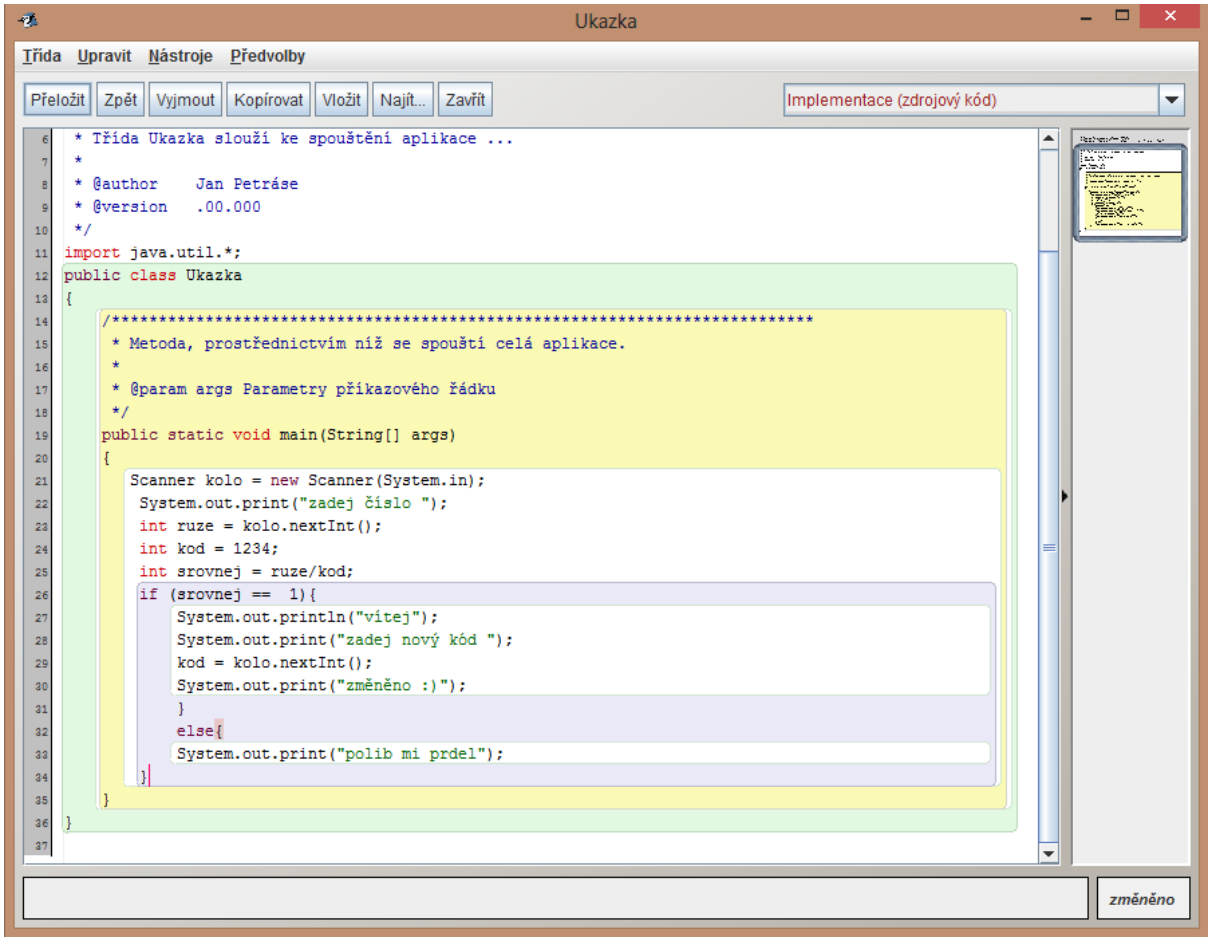
Dvojklikem na hnědý obdélník s černým šrafováním, který má na sobě nápis se jménem třídy, otevřeme editor.



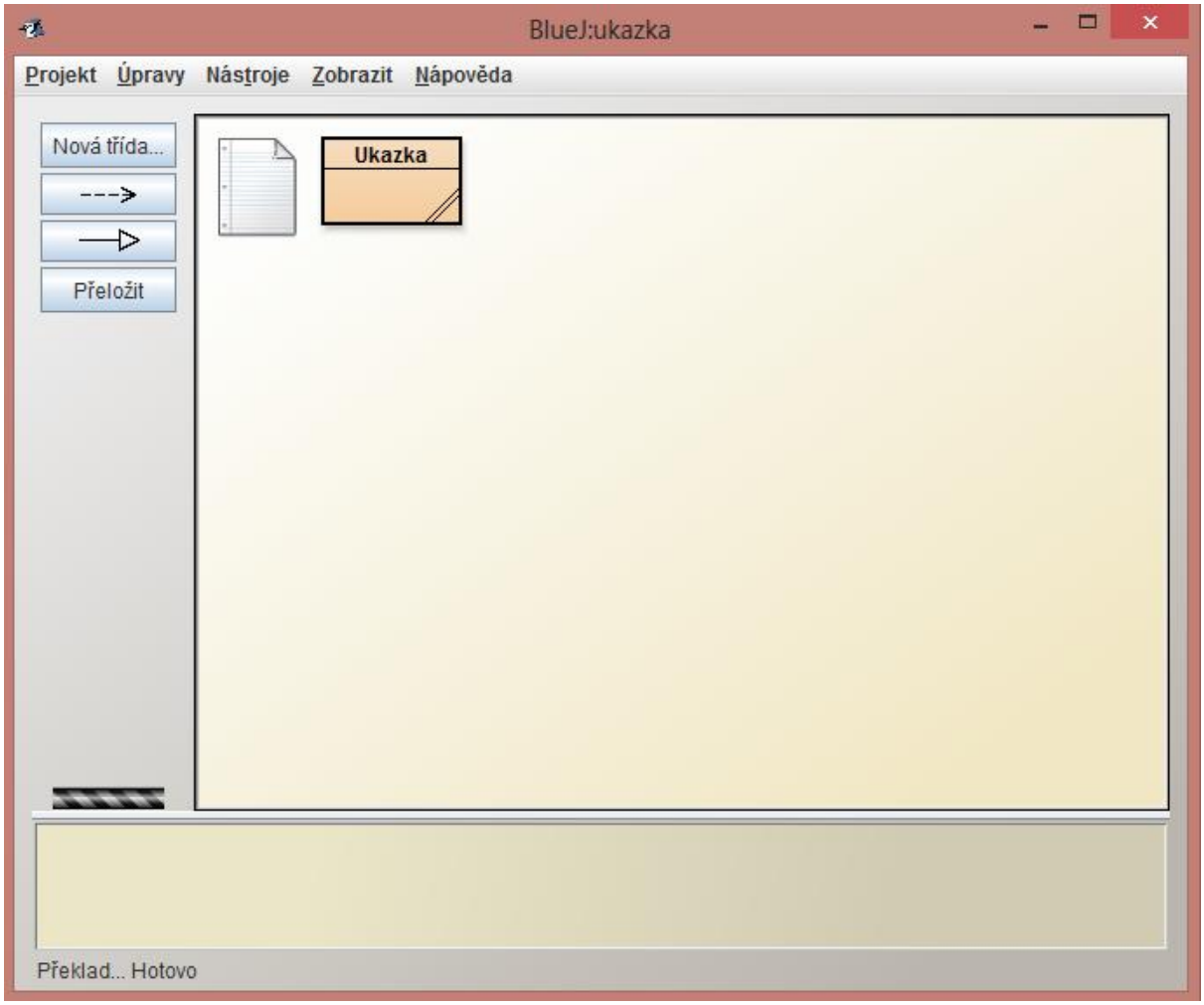
Vyplníme úvodní komentář a pod komentář nad položku public class Ukazka napíšeme příkaz import java.util.*;



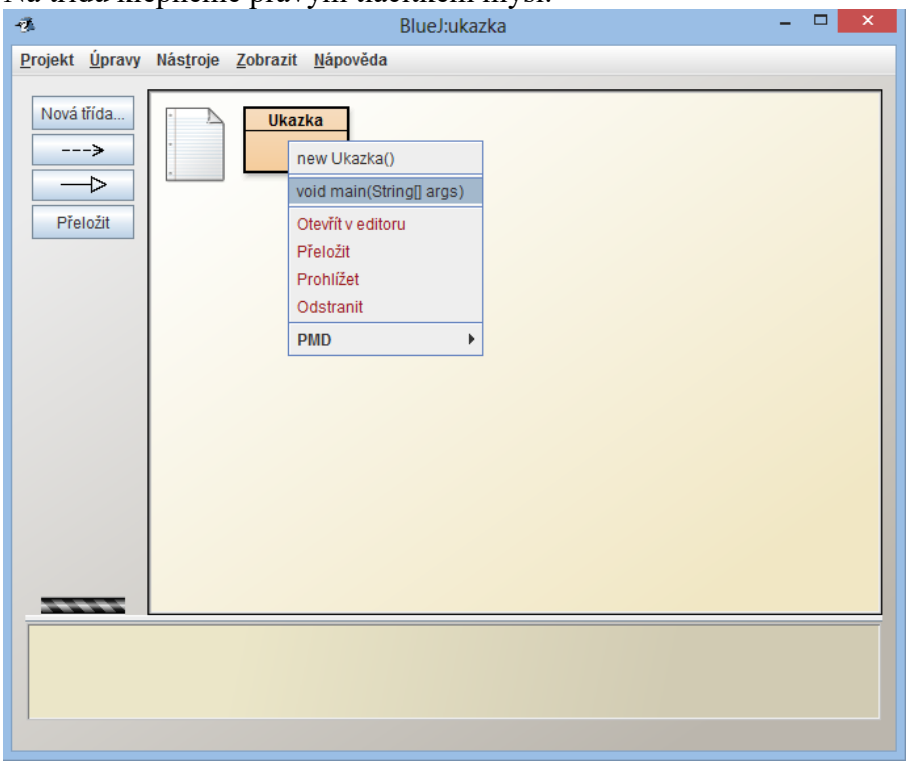
Na růžově určené místo napíšeme kód aplikace.



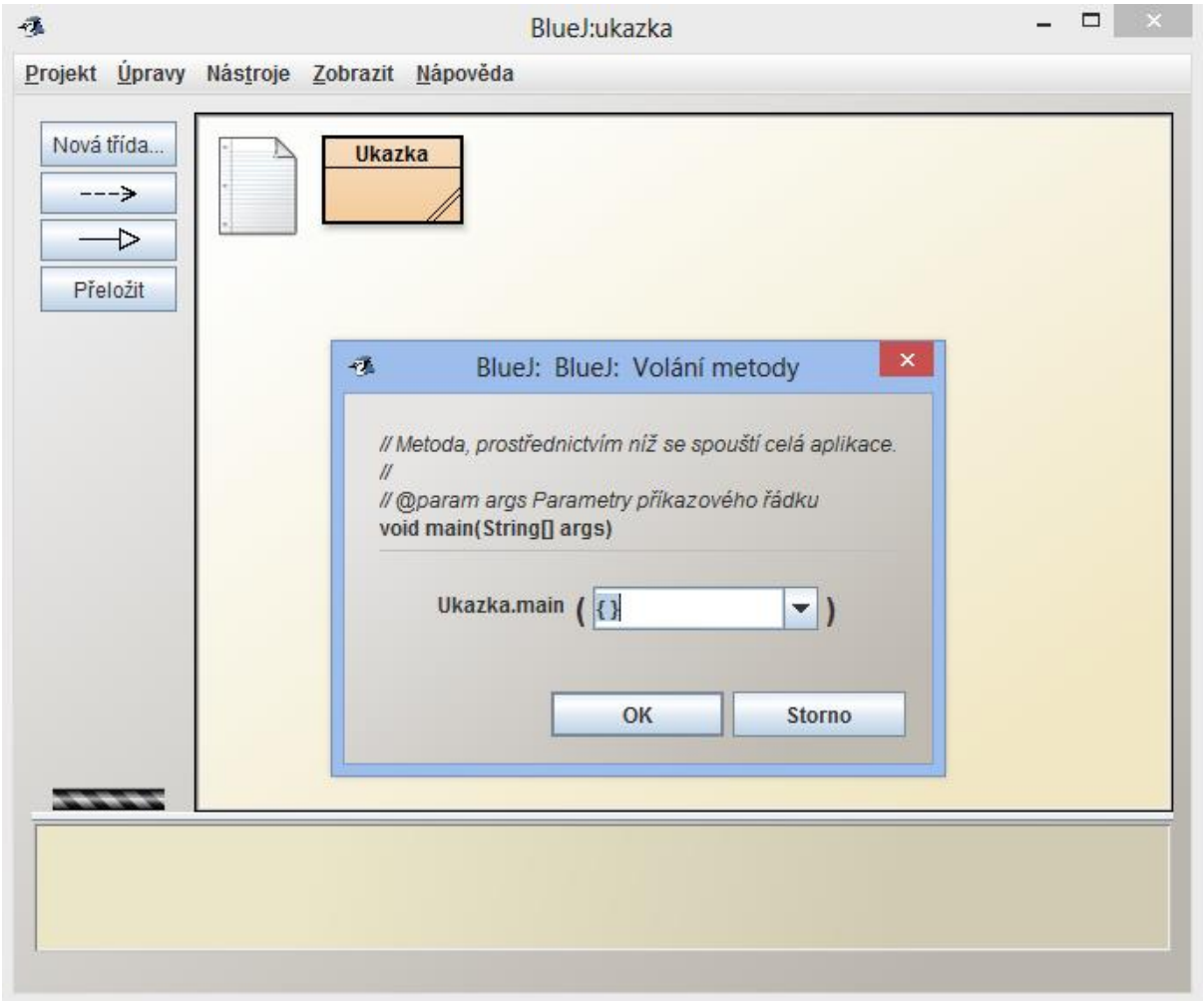
Následně zvolíme položku přeložit v horním menu a uzavřeme editor, pokud se neobjeví žádné chybové hlášení.



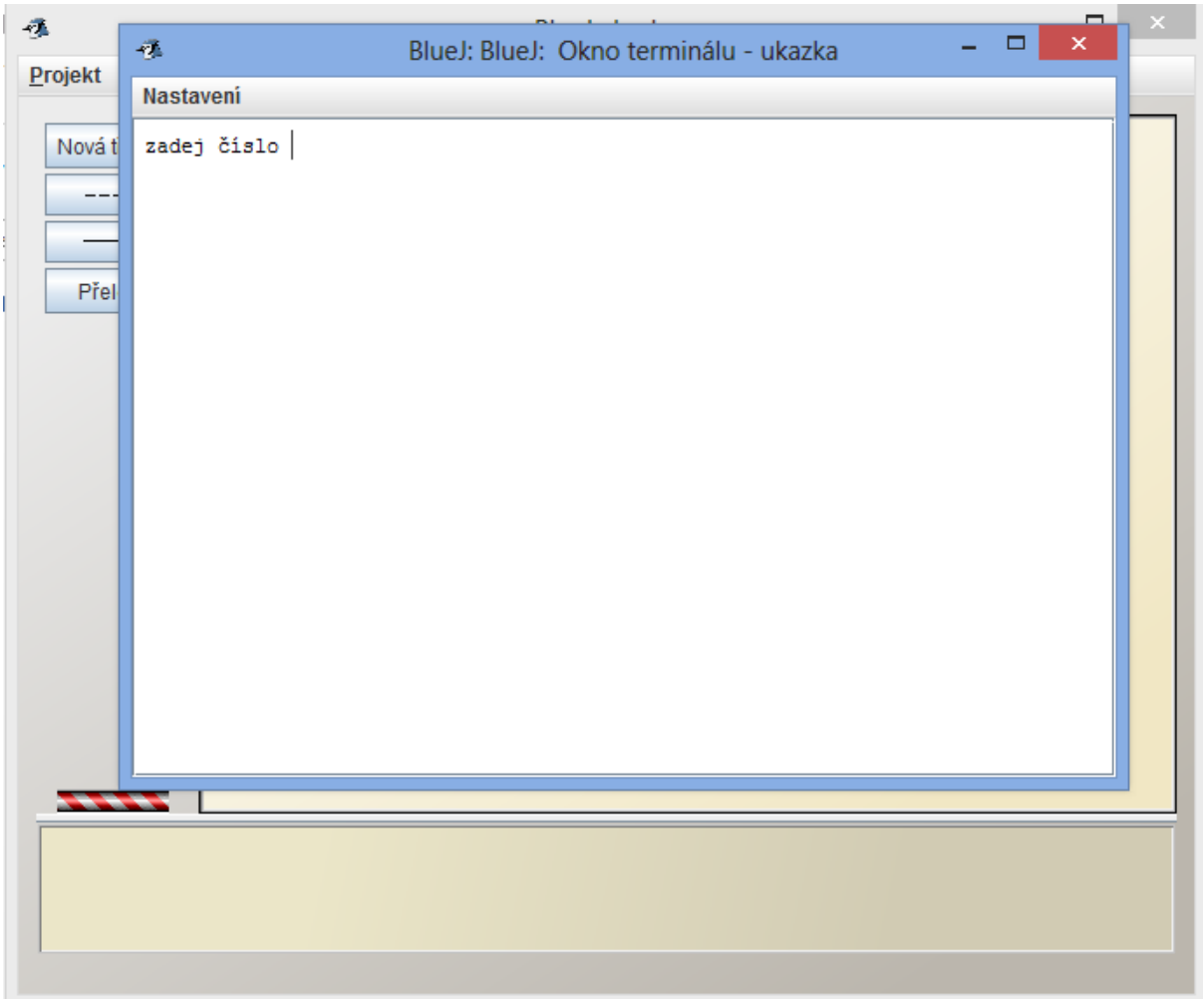
Na třídu klepneme pravým tlačítkem myši.



Zvolme vyznačenou položku void main…



Pro naše školní prográmky necháme vše bezezměny a stiskneme ok



Dojde k otevření okna proigramu
Po dokončení běhu programu okno jen uzavřeme.
Více toho potebova pro naše účely nebudeme.

17.8.3. Komponenty vývojového prostředí jazyka Java

17.8.3.1. Editor

Editor je program, který umožňuje psaní kódu a obsahuje prostředky, které usnadňují orientaci v kódu a jeho psaní.

17.8.3.2. Překladač

Je program, který přeloží program zapsaný v programovacím jazyce do strojového kódu jako celek a vytvoří spustitelný modul obvykle s příponou exe.

17.8.3.3. Interpret

Přeloží konkrétní příkaz a hned jej provede. Nevytváří modul s příponou (nevzniká soubor překladu programu). Tento prvek se užívá u hybridních jazyků, jako je Java. Program, který je takto spuštěn využívá prostředků alokovaných vývojovým prostředím.

17.8.3.4. Preprocsor

Je pomocný prvek, který se užívá u jazyků rodiny C, Basic a dalších. Tento prvek slouží k připojení obsahu knohoven, a které je v kódu odkazováno, k vlastnímu kódu. Teprve takto značně modifikovaný kód jde k překladači, který jej začně zpracovávat.

17.8.3.5. Linker

Je pomocný prvek, který se užívá u jazyků rodiny C, Basic a dalších. Linker propojí objekty z užívaných knihoven s jejich zdrojovými kódy, tady umožní finální tvorbu spustitelného souboru.

18. Jazyk Java

Patří do skupiny hybridních jazyků. Jde o vyšší jazyk, který je založen na objekově orientovaném přístupu k tvorbě programu.

18.1. Provádění programu

Program je přeložen do bajtkódu (bytecode), který je nezávislý na cílovém počítači (může být i ro mobilní zařízení). Soubor programu v podobě tytecode má příponu class. Program přeložený do bajtkódu se dále analyzuje a interpretuje virtuálním strojem Javy (Java Virtual Machine) - JVM (ochranné běhové prostředí, které ve strojovém kódu dává instrukce procesoru). Každá konfigurace hardware a OS, na kterém má program v Java běžet, musí mít svůj JVM.

18.2. Objekty

Je to základní prvek OOP. Vlastnosti objektů jsou ukládány do proměnných a každému objektu jsou naprogramovány (předdefinovány) vlastnosti. Objekt nabízí sám činnosti, po naprogramování je objekt hotov a nelze jej již doprogramovat. Každý objekt má podprogramy, které můžeme volat, tedy metody. Objekt je datová struktura složená z proměnných a podprogramů (metod). Metody definují chování objektu. Každý objekt je uzavřený celek, jež je instancí třídy, každý objekt je utvořen z třídy.

18.3. Třídy

Třída je datový typ (šablona), dle kterého se tvoří objekty. Každý objekt musí být vytvořen (iniciován/instancován) na základě určité třídy, která mu propůjčuje vlastnosti. Dá se tedy říct, že objekt je instancí (výskytem) třídy. Proměnné objektu se nazývají proměnné instance a metody jsou metody instance, při tvoření objektu jsou to kopie na základě třídy (u proměnných jsou to jiná paměťová místa a stejné označení). Ve třídě je možno definovat proměnné a metody statické (jsou společné (stejná místa v paměti) pro všechny objekty vytvořené na základě dané třídy).

Členy třídy jsou data a metody. Sama o sobě nemá schopnost něco realizovat, ale nelze bez ní utvořit objekt. Třídy jsou součástí jazyka. Součástí instalace Java je velké množství tříd, které seskupují se dle charakteru do balíků. Balík má stejnou strukturu jako adresáře, jediný rozdíl je, že názvy balíků a podbalíků se sice píší jako u adresářů, ale oddělují se tečkou nikoliv lomítkem (např.: java.lang). Souhrn balíků tvoří API (aplikační programové rozhraní). U NetBeans je nutno vytvořit napřed souhrnný balík a třídu, u BlueJ to nutné není, užívají se již nainstalované třídy. Popis tříd se dá otevřít přes nápovědu a odkaz na web oracle.com.

18.3.1. Třída Math

Třída Math je první třídou, na kterou jsme narazili. Nyní ještě nebudeme rozebírat, co to vlastně třída je, ale spokojíme se pouze s důsledkem, že nad ní můžeme volat operace. V případě třídy Math se jedná o operace matematického charakteru, které nám neposkytují uvedené operátory.

Pokud používáte vývojové prostředí, pak stačí napsat Math. a po napsání oné tečky vám IDE samo nabídne operace, kterých je tato třída schopná, informuje vás o parametrech, které operace vyžaduje a v neposlední řadě řekne, jaký návratový typ máte očekávat.

Jako následující bod si ukážeme tabulku se základními funkcemi, které z této třídy můžeme volat, ale třída jich obsahuje mnohem více.

Metoda	funkce	parametr	Typ výsledku
Abst (x)	IxI	Double, int, float, long	Odpovídá vstupu
Acos(x)	$Cos^{-1}(x)$	double	Double
Asin(x)	$Sin^{-1}(x)$	double	Double
Atan(x)	$Tg^{-1}(x)$	Double	Double
Atan2(xy)	Tg^{-1} y/x	double	Double
Exp(x)	e^x	Double	Double
Log(x)	Log(x)	Double	Double
Pow xy	x^y	double	Double
Sqrt(x)	2. odmocnina z x	Double	Double
toRadians	Stupně => rad	double	Double
E	Eulerovo číslo	Nemá	Double

Nesmíme při předávání parametrů zapomenout na to, že některé operace přetypování se provedou automaticky a jiné si musíme vyžádat. Technickým dovětkem je, že pokud operace vyžaduje více parametrů, tak je oddělujeme čárkou (a před jména proměnných nedáváme dle konvence znak ani $ (tato poznámka slouží především pro programátory přecházející z jazyka PHP)).

18.3.2. Třída string

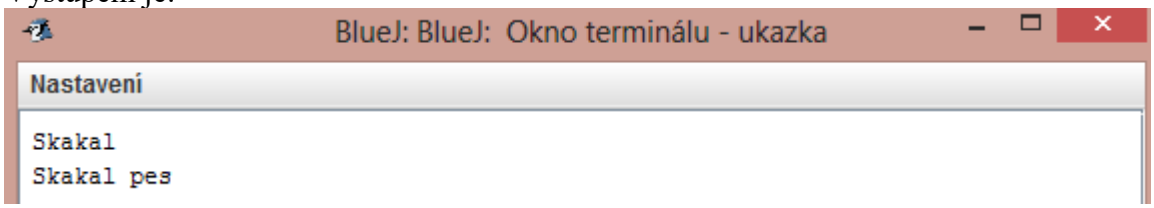
Naučili jsme se, že řetězce se dají vytvořit tak, že literál uzavřeme z obou stran do horních uvozovek. Co jsme si ale neřekli je, že se nad jednotlivými řetězci dají volat operace naprosto stejným způsobem, jako jsme tak činili nad třídou Math.

```
        /**********************************************************************
***********
         * Třída Ukazka slouží ke spouštění aplikace ...
         *
         * @author    Jan Petrásek
         * @version   1.00.000
         */
        import java.util.*;
        public class Ukazka
        {

/**************************************************************************
*
            * Metoda, prostřednictvím níž se spouští celá aplikace.
            *
            * @param args Parametry příkazového řádku
            */
           public static void main(String[] args)
           {
             String s = "Skakal pes pres oves"; //retezec
             s = s.substring(0, 6); //prvnich sest znaku
```

```
        System.out.println(s); //"Skakal"
        String concat = s + " pes"; //zretezeni vice retezcu
        System.out.println(concat); //"Skakal pes
    }
}
```

Výstupem je:



Tento příklad obsahuje jeden drobný chyták. Jednotlivé řetězce jsou immutable, což znamená, že jejich obsah nemůžeme nijak měnit (pro zkušenější mohu prozradit, že řetězec je implementován jako pole znaků). Tudíž všechny modifikující operace vytvářejí vždy nový řetězec a musíme proto odkaz na tento modifikovaný řetězec uložit znovu do proměnné (na druhé řádce).

18.4. Konvence jazyka

Java je case-sensitive (kejs senzitiv) (rozlišuje velká a malá písmena). Identifikátor (jméno) může být libovolně velký, může obsahovat pouze písmena anglické abecedy, čísla a podtržítka, ale vždy začíná písmenem. Jména tříd začínají velkým písmenem a jména metod a proměnných začínají malým písmenem. Jména proměnných a metod instance, které mají více slov, se píší jako jedno slovo a druhé až xté slovo začíná velkým písmenem (nazdarLidi). Pojmenované konstanty (textové, číselné, logické) se píší (celé jméno) velkými písmeny a slova jsou oddělena podtržítkem. Jména balíků se píší malým písmem a oddělují se tečkou. Každý program začíná komentářem => z vhodně napsaného se vytváří dokumentace. Komentáře je dobré užít i v průběhu psaní programu.

18.4.1. Komentáře

Programátor má na výběr celkem ze tří typů komentářů, záleží, k čemu je komentář určen. Ve většině vývojových prostředí editory komentáře ještě oddělují opticky tím, že je písmo zelené.

víceřádkový

- začíná */ a končí */

řádkový

- užívá se k příkazům
- začíná // a končí s koncem řádku

dokumentační

- začíná /** a končí /**
- text začíná @
- automaticky se z něj generuje dokumentace k programu

18.4.2. Struktura programu

Každý program v jazyce Java musí obsahovat alespoň jednu třídu, ve které bude alespoň jedna metoda. Třídy mohou být veřejné (přístupné i z jiných balíků) (publick), nebo privátní, a musí být uloženy do samostatného souboru. Jméno hlavní třídy musí být stejné jako jméno adresáře do, kterého je ukládána. U dalších tříd programu již není nutná shoda jména s adresářem. Hlavní metoda programu je též povinná (spouští program) a má danou i podobu, která je:
public static void main(String[] args)

Tělo třídy a metody jsou ohraničeny složenými závorkami, obdobně jako těla cyklů či větvení. Pokud chceme zobrazit text (textový řetězec) tak jej vždy píšeme k patřičnému příkazu do kulatých závorek a tex vždy začíná i končí uvozovkami.

18.5. Základní pojmy

18.5.1. Identifikátor

Identifikátorem v programovacím jazyce myslíme jméno. Na rozdíl od lidského jazyka má identifikátor v každém jazyce definovanou maximální délku (pro Javu je to 255 znaků).

Obdobně jako v angličtině se může skládat pouze z malých a velkých písmen anglické abecedy, číslic a podtržítka (které nahrazuje mezeru). Musí začínat písmenem a ze speciálních znaků je povolen znak dolar.

18.5.2. Klíčové slovo

Jde o slovo, které má svůj pevně daný význam v jazyce a není možné jej užít k pojmenování objektu, třídy atd. Příkladem takových slov jsou názvy příkazů, typů a tříd, tedy konkrétně některé příklady (if, else, final, while, main…). Pro přirovnání v češtině by mohlo být klíčovým slovem třeba sloveso (název slovního druhu (tedy třídy)), naproti tomu třeba Lucie již klíčovým slovem není. Klíčové slovo samo o sobě definuje neměnné vlastnosti či činnosti. Tedy u toho přirovnání k češtině, sloveso je vždy popisem nějaké činnosti a nejčastěji také přísudkem ve větě. Pokud se podívám na Lucii, tak podle tohoto slova o té ženě nevím nic, je zrzka, blondýna, hubená, vysoká, hodná?? Lucky nic nespojuje na rozdíl od sloves.

18.5.3. Oddělovač

Odděluje jednotlivá slova v příkazech a komentářích. Základním oddělovačem je mezera. Další oddělovače jsou například středník (odděluje příkazy), závorky, tečky, bílé (netisknutelné) znaky (mezera, tabulátor a enter).

18.5.4. Proměnná

Proměnná je pojmenovaná část paměti, kterou používáme k uložení informace. Vezmeme-li si jako příklad zaměstnance nějaké firmy, tak proměnná popisující jeho plat může mít hodnotu 25000, heslo k jeho uživatelskému účtu bude nenavidimPraci a příznak jeho pohlaví jeMuz bude nastaven na hodnotu pravada.

V Javě vždy při deklaraci proměnné určíme její typ. Tento typ později již nemůžeme nijak změnit (v rámci bloku, kde je proměnná platná). Pokud například prohlásíme o proměnné x, že je typu celé číslo, tak se do ní nesmíme později snažit umístit řetězec. Toto není nijak překvapivé pro začátečníka, případně pro člověka se zkušenostmi s jiným silně typovaným jazykem. Ale pro uživatele například PHP to může být šok (je dynamicky typované – typ proměnných se mění PHP automaticky v závislosti na datech). Silná typovost se proto může zdát být slabinou a zbytečným šikanováním programátora, ale ve skutečnosti velmi usnadňuje práci (vývojové prostředí může lépe radit a pomáhat) a zabraňuje značnému množství chyb z nepozornosti.

18.5.4.1. Celočíselné datové typy

Pro ukládání celých čísel Java obsahuje čtyři datové typy byte, short, int a long. Tyto typy se liší pouze v rozsahu hodnot, které mohou používat. Všechny obsahují jak kladná, tak záporná čísla (a nelze to ovlivnit).

Důležitou poznámkou je, že všechny celočíselné literály (tj. hodnoty přímo vepsané ve zdrojovém kódu) jsou implicitně typu int. Pokud proto chceme vložit do proměnné typu long

hodnotu, která je vyšší než maximální rozsah typu int, tak musíme na konec literálu přidat znak l nebo L.

Jaké hodnoty se tedy vejdou do jednotlivých typů celočíselných proměnných?

- ✓ byte: -128 až 127 (8 bitů)
- ✓ short: -32768 až 32767 (16 bitů)
- ✓ int: -2147483648 až 2147483647 (32 bitů)
- ✓ long: -9223372036854775808 až 9223372036854775807 (64 bitů)

18.5.4.2. Datové typy reálných čísel

Pro reálná čísla existují typy float ($-3{,}4^{38}$ až $3{,}4^{38}$) a double (-1.7^{308} až 1.7^{308}). Protože tyto typy musí mít velký rozsah (od obrovských čísel po velmi malé zlomky), používá se pro jejich reprezentaci plovoucí desetinné čárky. To znamená, že jsou čísla zapsána tak, aby jejich první platná číslice byla přímo za desetinnou čárkou (tzv. mantisa) a za tímto zápisem následuje číslo reprezentující exponent základu soustavy, kterým když přenásobíme mantisu, tak dostaneme uložené číslo. Takto to vypadá velmi složitě, ale z příkladů (v desítkové soustavě) je jasně patrné, jak to funguje.

$0{,}165e3 = 0{,}165*10^3 = 165$

$0{,}1e0 = 0{,}1*10^0 = 0{,}1$

$0{,}1e3 = 0{,}1*10^3 = 100$

Na tomto matematickém cvičení je důležité jedno poselství – typy float a double nejsou přesné, protože obsahují určitý fixní počet míst pro zápis mantisy a exponentu.

Poslední poznámkou pro reálná čísla je, že pro literály je implicitním typem double, proto pokud chceme říci kompilátoru, že je daný literál typu float, tak na konec literálu připojíme znak f nebo F.

18.5.4.3. Datový typ znak

Pro ukládání jednotlivých znaků v Javě slouží typ char, který zabírá v paměti 2 bajty místa. Pro vložení hodnoty znaku do proměnné lze použít buď celočíselného literálu, apostrofů ('A') nebo esc (unicode) sekvence ('\u0041').

Poznámka: Mezi použitím apostrofů a uvozovek je v Javě rozdíl. Zatímco 'A' znamená znak A, "A" znamená řetězec, který obsahuje jeden znak A.

18.5.4.4. Logické proměnné

Pro reprezentaci logických hodnot (pravda/nepravda) slouží typ boolean. Boolean může obsahovat buď hodnotu true – pravda, nebo false – nepravda.

18.5.4.5. Deklarace a inicializace

Pro vytvoření proměnné (deklaraci) a dosazení hodnoty do proměnné (inicializaci) se dá použít několik různých zápisů. Všechny zápisy jsou správné, některé jsou však správnější (viz komentáře). Následující kód obsahuje deklarace a inicializace všech datových typů, které jsme si dnes ukázali. Jak si asi všimnete, tak je za každým příkazem středník, to je v Javě povinnost. Povinností ovšem není mít na řádce pouze jeden příkaz, to je obecná kultura.

Zde již tedy slíbený kód, kde si zkusíme deklarovat proměnné a vypsat jejich hodnoty. Kód doporučuji přepsat ručně a opravdu si jej vyzkoušet.

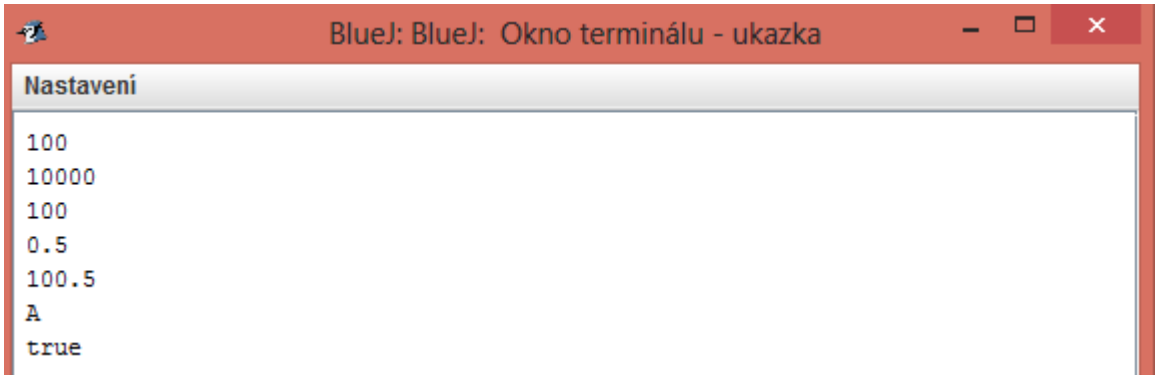
```
/**************************************************************************
 * Třída Ukazka slouží ke spouštění aplikace ...
 *
 * @author    Jan Petrásek
 * @version   1.00.000
 */
import java.util.*;
```

```
public class Ukazka
{

/*****************************************************************************
     * Metoda, prostřednictvím níž se spouští celá aplikace.
     *
     * @param args Parametry příkazového řádku
     */
    public static void main(String[] args)
    {
        int i1; //pouze deklarace
        i1 = 1500; //inicializace
        int i2 = 1500; //deklarace s inicializaci - doporuceny postup
        int i3, i4, i5; //deklarace vice promennych
        int i6 = 7, i7 = 15, i8 = 17; //toto take neni doporuceno
        byte b = 100; //byte
        short s = -30000; //short - zaporne cislo
        short s2 = 30000; //short - kladne cislo
        int i = 10000;
        long l = 100;
        long l2 = 150000000000l; //long - prilis velke cislo
        long l3 = 150000000000L; //long - varianta L
        double d = 100.5;
        double d2 = 0.5;
        double d3 = .5; //0.5
        double d4 = 5E-1; //0.5
        double d5 = 0.5e0; //0.5
        float f = 100.5f;
        float f2 = 100.5F;
        char ch = 'A';
        char ch2 = 65; // A (ASCII hodnota)
        char ch3 = '\u0041'; //esc sekvence
        boolean bool = true;
        boolean bool2 = false;
         /*****************************
           *       Priklad vypisu      *
         *****************************/
         System.out.println(b);
         System.out.println(i);
         System.out.println(l);
         System.out.println(d5);
         System.out.println(f2);
         System.out.println(ch3);
         System.out.println(bool);

    }
}
```

Pokud máme celý kód správně, tak bychom měli vidět následující:



18.5.4.6. Platnost proměnné

Každá proměnná je platná (a nesmí být opětovně deklarována) v rámci bloku. Blok je taková část kódu, která je uvnitř jednoho páru složených závorek. V případě vnořených bloků proměnná platí i ve všech vnitřních blocích. Ovšem v okamžiku, kdy opustíme blok, kde byla proměnná deklarována, ji již nesmíme použít (protože neexistuje).

V následujícím příkladu budeme používat pro bloky pouze osamocené znaky složených závorek. Toto je v Javě poměrně neobvyklé, protože se bloky téměř vždy vážou k cyklům, podmínkám, definicím metod a tříd, ale tato problematika nás teprve čeká.

```
/**********************************************************************
 * Třída Ukazka slouží ke spouštění aplikace ...
 *
 * @author    Jan Petrásek
 * @version   1.00.000
 */
import java.util.*;
public class Ukazka
{

/**********************************************************************
    * Metoda, prostřednictvím níž se spouští celá aplikace.
    *
    * @param args Parametry příkazového řádku
    */
   public static void main(String[] args)
   {
      int a = 5;
      System.out.println(a); //5
      { //vnitřní blok
         System.out.println(a); //proměnná je i ve vnitřním bloku platná
         int b = 10;
         System.out.println(b); //10
      } //konec vnitřního bloku
      System.out.println(b); //toto je chyba, zde proměnná již neexistuje
   }//konec bloku metody
}
```

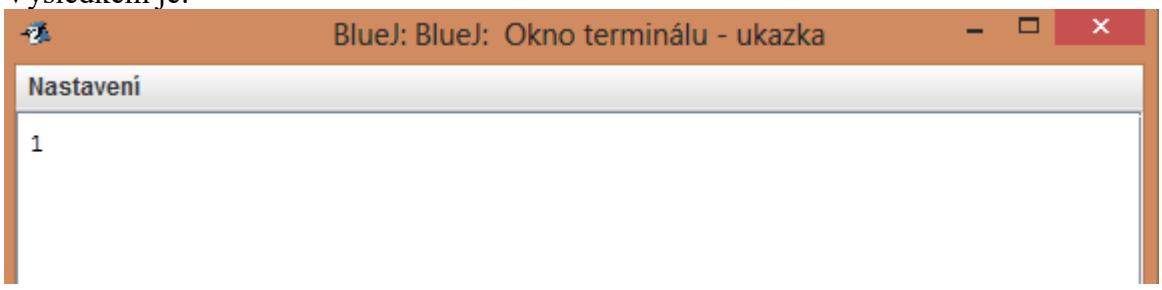
Tento kód se již nepřeloží, protože obsahuje syntaktickou chybu v podobě nedodržení platnosti proměnné b.

18.5.4.7. Přetypování proměnné

V určitých situacích potřebujeme změnit typ proměnné, což může nastat tehdy, chceme-li volat operaci, která přijímá celá čísla (int), ale naše proměnná je typu double. Některé konverze typu proběhnou automaticky (stačí přiřadit hodnotu do nového typu), u jiných o ně musíme explicitně zažádat. Do první skupiny patří ty změny, u nichž zaručeně nedojde k žádné ztrátě dat (byte → int, int je schopen pojmout více dat než byte), do té druhé ty změny, kde je ztráta dat možná (double → int, double má jak větší rozsah, tak obsahuje desetinné pozice). V případě explicitní konverze o tuto změnu zažádáme tím, že před proměnnou napíšeme do závorek typ, který si přejeme.

```
/*******************************************************************************
 * Třída Ukazka slouží ke spouštění aplikace ...
 *
 * @author    Jan Petrásek
 * @version   1.00.000
 */
import java.util.*;
public class Ukazka
{
/*******************************************************************************
   * Metoda, prostřednictvím níž se spouští celá aplikace.
   *
   * @param args Parametry příkazového řádku
   */
  public static void main(String[] args)
  {
     byte b = 5;
     int i = b; //automaticka konverze, int ma vetsi rozsah
     long l = 150000L;
     i = (int) l; //hrozi preteceni, musime sami o zmenu zazadat
     double d = 1.5;
     i = (int) d; //hrozi ztrata presnosti, o zmenu musime zazadat
     System.out.println(i); //vypise 1
     d = i; //v poradku, double ma vetsi rozsah nez integer
  }
}
```

Výsledkem je:



Pokud si seřadíme jednotlivé typy do řady, tak konverze proběhne vždy automaticky tehdy, je-li výchozí typ nalevo od cílového, v opačném případě musíme o konverzi zažádat (všimněte si, že typ boolean nelze přetypovat).

18.6. Operace

Aby program mohl fungovat, tak musí něco dělat. Jednotlivým dílčím činnostem říkáme operace, jde o přiřazení hodnoty do proměnné, provedení nějaké operace typu vypsání na monitor, načtení vstupu, nebo také matematické či logické operace. Informatika podobně jako fyzika matematizuje svět kolem nás, tudíž výkonné jádro programu tvoří matematické operace. Jako první si objasníme operaci výpisu na monitor, kterou jsem nám již několikrát dal do ukázkových kódů.

18.6.1. Výstup

Standartní výstup není žádnou vědou, stačí si zapamatovat, že k programu je nutné připojit vždy balík java.util a to celý, tak jak bylo zmíněno v předchozích kapitolkách. O výstup se potom postará jeden ze dvojice možných příkazů. První je: *Systém.out.print (….)*; a druhý je: *Systém.out.println (…)*;.

Jak tedy s nimi pracovat? Do kulatých závorek přijde proměnná, jejíž obsah má být zobrazen, či v uvozovkách zadaný text. Typ proměnné není omezen ani u jednoho z dvojice příkazů. Příkazy se liší tím, že …print nechá za ukončením výpisu kurzor na stejném řádku, zatím co …println po ukončení výpisu přesune kurzor na začátek následujícího řádku.

Pokud budeme chtít spojit více textových řetězců (text uzavřený do uvozovek) či proměnných nebo vypsat tex a proměnnou, tak dané bloky výpisu spojujeme pomocí znaménka +. Takže například: Systém.out.println („Výsledný počet studentů na přednášce je: „ + x);

Pokud budeme chtít tisknout výsledek za zadáním, kdy jsme pro tisk výzvy k zadání užili příkaz … print(…), tak je nutné začít další výpis tzv. esc sekvencí, která nám posune kurzor na začátek dalšího řádku. Esc sekvence toho umí mnohem více a již jsme se s nimi setkali u znakových proměnných. Nyní bych vám rád ukázal přehled základních esc sekvencí.

- ✓ \n – netisknutelné znaky
- ✓ \r – návrat kurzoru na začátek řádku)
- ✓ \\ - tisk obráceného lomítka
- ✓ \t – tabulátory
- ✓ \u a kód z unicodove tabulky slouží k tisku znaků, které nejsou na klávesnici

18.6.1.1. Formátovaný výstup

Text výstupu nemusí mít jen jednu podobu, tedy formát. Není problém, aby některé části textu byly tučně či kurzivou, nebo podtržené. Hlavně je možné nastavit počt zobrazovaných míst u čísel. Pro formátovaný výstup je k dispozici třída Formatter z balíku java.util a v ní je metoda format, která se volá objektem *Systém.out.format*. Základní parametr je textový řetězec uzavřený do uvozovek a hodnoty zadané konstantami, výrazem, proměnnýma, jsou v závorce (jako u print). Všechny parametry se oddělují čárkami. Součástí textového řetězce je formátovací maska, která je nahrazena v tisku příslušnou hodnotou (formátem). Neslučuje se plusy, parametry oddělujeme čárkami!! Každá maska začíná znakem % a končí písmenkem (d,e,f,g) a musí mít hodnotu, která ji nahradí, kromě %n (odřádkování). Formátovací masku nemusíme oddělovat mezerou od následujícího textu.

Základní je maska %d, která vypisuje celé číslo. Mezi % a d se může vložit číslice symbolizující počet míst, pokud je číslo kratší, tak se doplní mezerou zleva, tedy např.: Systém.out.format(„i = %d%n“,i) => i = -1233 (i=-1233). Pokud je počet míst čísla větší nežli požadovaný počet pro tisk, tak je požadavek pro tisk ignorován a tiskne se délka proměnné.

18.6.2. Vstup

Standartním vstupem je klávesnice. Z vyrovnávací paměti klávesnice (Buferu) probíhá čtení dat po bajtech a programátor musí vytvořit smysluplná čísla a znakové řetězce z přicházejících bajtů. Metody čtení jsou v balíku: java.util.Scanner, tedy opět se nám ukazuje nutnost připojit k programu balík java.util.

Dále musíme vytvořit objekt pro čtení (Scaner „sc“=new Scaner (systém.in), jehož jméno se užívá pro volání příslušných metod. Pro čtení celého čísla užijeme: *int i = sc.nextInt()* , kdy *i* je proměnná, kam se čte. Reálné číslo čte pomocí: *i = sc.nextDouble()* a znak se čte metodou: nextLine() (čte celý řádek) a na vytržení znaku musíme užít ještě charAt() (načtení jednoho znaku: *nextLine().charAt()*). Pokud užijeme takovýto zápis: Strink s= sc.next(); (čte znaky po první bílý (netisknutelný) znak). Když bychom chtěli změnit desetinnou čárku pro zadávání na desetinnou tečku, tak užijeme příkaz: *Sc.useLocale(Locale.US);*. Pro konstanty v programu platí, že se vždy píší s desetinnou tečkou, ale vstup se řídí pomocí nastavení OS, nebo jej můžeme sjednotit pomocí výše uvedeného příkazu. Při čtení čísel i znaků je nutno vytvořit zvlášť objekt pro číslo i pro znak.

Žravá metoda likviduje předešlé entery (anglicky greedy). Tato vlastnost je důležitá, protože znakem enter se odesílá, není-li nastaveno jinak, načtená posloupnost bytů. Pokud metoda není žravou, tak je nutné před dalším čtením na prázdno zavolat metodu nextLine, která sice není žravá, ale tím, že je na prázdno, tak zlikviduje zbylý enter. Znaky zadávané na klávesnici se čtou do vyrovnávací paměti jménem Buffer. Všechny metody (krom nextLine) čtou po první bílý znak (enter), který po předčtení zůstává v bufferu. Metody, které jsou žravé zbylý enter zlikvidují a pak čtou po další enter.

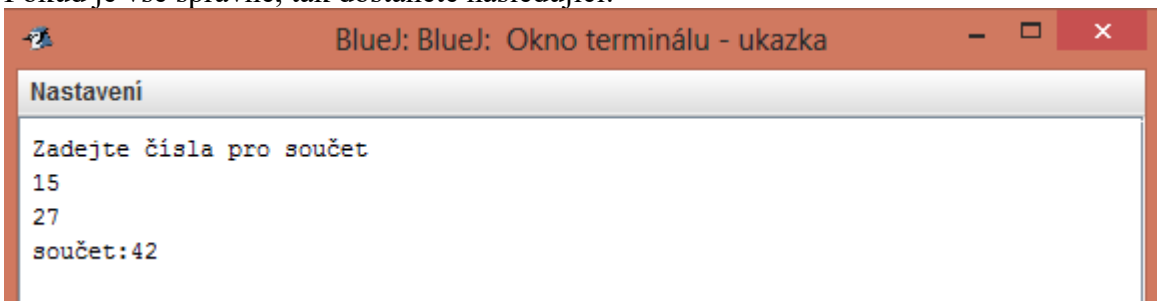
Žravé metody jsou číselné a metoda next().

Nyní si ukážeme jednoduchý program, který načte dvojici čísel a vypíše jejich součet, aby bylo vidět, jak v praxi vypadá čtení.

```
/*****************************************************************************
 * Třída Ukazka slouží ke spouštění aplikace ...
 *
 * @author    Jan Petrásek
 * @version   1.00.000
 */
import java.util.*;
public class Ukazka
{
/*****************************************************************************
  * Metoda, prostřednictvím níž se spouští celá aplikace.
  *
  * @param args Parametry příkazového řádku
  */
 public static void main(String[] args)
 {
  Scanner nevim = new Scanner (System.in);
  System.out.println("Zadejte čísla pro součet");
  int a=nevim.nextInt();
  int b=nevim.nextInt();
  int c;
  c=a+b;
  System.out.print("součet:"+c);
 }
```

}

Pokud je vše správně, tak dostanete následující:



18.6.3. Aritmetické

Pro aritmetické výpočty můžeme použít operátory +, -, *, /. Všechny tyto operátory mají očekávatelnou prioritu (tj. nejprve násobíme a dělíme, teprve až poté sčítáme a odčítáme), a lze při jejich použití využít uzávorkování.

Je důležité vědět, že pokud má některý operand typ double, tak je i druhý operand před provedením operace převeden na double, obdobně na long a float. Pokud užijeme celočíselný typ, tak dojde k odříznutí desetinné části, nikoli k zaokrouhlení na celé číslo!! Proto u dělení doporučuji pro výsledek užívat některý z reálných datových typů.

Velmi často se v programování dostaneme do situace, kdy je zapotřebí nad danou proměnnou vykonat aritmetickou operaci a výsledek opět uložit do původní proměnné. V kódu tím dochází k lehké redundanci. Z tohoto důvodu existují pro zkrácení zápisu operátory +=, -=, *= a /=. Toto je těžké na pochopení pro spoustu lidí, ale na nejprve si ukážeme konkrétní příklad a pak si jej vysvětlíme. Takže slíbený příklad: podprsenka += 3; . V praxi je to velmi užívané a jednoduché, představme si, že jsme slečna (tedy pokud fyzicky nejsme) a balíme si na služební cestu. V kufru máme nějaký počet podprsenek, který reprezentuje proměnná *a*, dejme tomu, že *a = 2*. Už máme dá se říct sbaleno, když nám zavolá šéf, a řekne, že tam budeme muset zůstat o týden déle. V takovém případě si musíme dobalit další kusy oblečení k tomu, které se již v kufru nachází. Takže vezmeme další tři podprsenky a vložíme je do kufru. Tím jsme dosáhli toho, že hodnota proměnné *a*, která reprezentuje počet podprsenek v kufru navýšila svoji hodnotu o ty tři přidané kusy, takže *a = 5*. Nicméně pořád je to proměnná a, nikoli p nebo q, pořád je to ta samá a reprezentuje to samé, tedy k původním obsahu jsme přičetli nové kusy a uložili to zpět na původní místo.

18.6.4. Inkrementace a dekrementace

Pro zvyšování (inkrementace) a snižování (dekrementace) hodnoty proměnné o 1 existuje zvláštní konstrukt (++ a -- zapsané buď před, nebo za název proměnné). Tímto se jednak ušetří místo, ale hlavně je možné tuto operaci použít uvnitř jiného aritmetického výrazu.

Pokud zapíšeme znaménka před proměnnou hovoříme o preinkrementaci (predekrementaci) – při vyhodnocování výrazu se nejprve provede změna hodnoty proměnné a až posléze se vyhodnotí výraz. Při opačném zápisu (proměnná++) – postinrekementaci (postdekrementaci) – se nejprve vyhodnotí výraz a až posléze dojde k přiřazení nové hodnoty.

18.6.5. Modulo

Modulo je samo o sobě zvláštní, setkáváme se s ním prvně již na základní škole, když se učíme ve druhé třídě dělit se zbytkem, ale dá se říct, že až po výuku programování je nám vlastně zatajeno, že jde o modulo a která část toho výsledku to vlastně je. Modulo není ve skutečnosti ničím jiným nežli zbytkem po celočíselném dělení. S modulem se setkáváme také třeba v čase, když chceme určit kolik je vlastně 175 minut, tak uděláme 175 : 60 celočíselně a

dostaneme 2, nicméně 2*60 je 120, tudíž tam něco chybí. Abychom nemuseli tento zbytek zjišťovat pomocí násobení a rozdílu, tak můžeme provést rovnou modulo, kde jej dostaneme. Pokud tedy provedeme 175 modulo 60, tak dostaneme 55. Jinak s funkcí modulo jsme již pracovali pro převodu do dvojkové soustavy, kdy nás vlastně zajímaly pouze výsledky funkce modulo.

Tato funkce se používá třeba ke zjištění dělitelnosti, pokud je číslo dělitelné jiným, tak výsledek x modulo y = 0, pokud je x dělitelné y. V Java zapisujeme příkaz k provedení této funkce operátorem %, tedy zjištění dělitelnosti by vypadalo takto: x % y == 0;, pokud tato rovnost platí, tak je číslo x dělitelné číslem y.

18.6.6. Operace se znaky

Znaky jsou v Javě uloženy jako čísla, která je reprezentují v ascii tabulce. Proto je možné třeba Caesarovu šifru vytvořit pomocí přičtení konstanty, nebo znak zaměnit pomocí inkrementace.

18.6.7. Binární operace

Pro úplnost operací si ještě uvedeme binární operace. Tyto neoperují s čísly v desítkové soustavě, jak jsme zvyklí, ale nad jednotlivými bity jejich dvojkového zápisu. Pro začátečníka není jejich přesná znalost důležitá. Důležitá je především znalost jejich existence, protože se čas od času hodí zejména tehdy, zabrousí-li člověk do něčeho nízkoúrovňovějšího.

- & Binární konjunkce
- | Binární disjunkce
- ^ Nonekvivalence ( ) XOR
- ~ Bitový doplněk (otočí všechny bity)
- << Posuv vlevo, zprava doplňováno nulami
- >> Posuv pravo, znaménkový bit je kopírován
- >>> Posuv pravo, zleva doplňováno nulami

18.7. Balíčky

Vzhledem k tomu, že v každém programu jsou minimálně desítky tříd a navíc používáme při programování knihovny, tak se velmi brzo dostaneme do situace, kdy pojmenujeme třídu duplicitním názvem. Abychom mohli tyto třídy vzájemně odlišit, tak Java používá balíčky.

Balíčky jsou logickým seskupením několika tříd, které se fyzicky projevuje jako adresář (každé veřejné třídě odpovídá jeden soubor). V rámci zdrojového souboru vždy deklarujeme balíček a volitelně vyjmenováváme třídy z ostatních balíčků, které používáme (import).

Pokud bychom třídy nevyjmenovali, tak k nim musíme vždy přistupovat pomocí plně kvalifikovaného jména, což je cesta k dané třídě v tečkové notaci (místo lomítek jsou použity tečky, pro import obsahu balíčku lze použít divokou kartu *, což se ale příliš nedoporučuje).

Java implicitně provádí import java.lang. * V tomto balíčku je například jak třída Math, tak třída String, proto jsme je doposud nemuseli nijak uvádět.

Samozřejmě jsme ale mohli v kódu k daným třídám přistupovat pomocí plně kvalifikovaného jména (což je ale poněkud rozvláčné).

18.8. Metody

Metoda je stejně jako třída tvořena hlavičkou a tělem. Hlavička začíná specifikátorem přístupu (public, private...) za ní následuje návratový typ, což je hodnota, kterou metoda musí vracet. Pokud metoda nemá vracet žádnou hodnotu, tak zde zapíšeme klíčové slovo void. Další položkou je název metody a v kulatých závorkách čárkou oddělené parametry volání.

Názvy metod zapisujeme camelCasem a měly by být slovesem (vyjadřovat akci). Pro otevírací závorky bloků platí všeobecná konvence – jsou vždy na stejné řádce jako deklarace, ke které níž se vztahují. Zavírací závorky jsou vždy na samostatném řádku na úrovni prvního písmene deklarace. Ukážeme si v praxi.

```
public class Animal
{
/**     * Druh zvířátka     */
 public String kind;
 /**     * Zvuk, který vydává     */
public String sound;
/**     * Metoda, která vypíše pozdrav zvířátka     */
public void sayHello()
  {
    System.out.println("Já jsem " + kind + " a delam " + sound);
  }
        }
```

Protože se jedná o instanční metodu, která se vztahuje ke konkrétní instanci (a jejím datům), tak ji vždy musíme volat přímo na daném objektu, nikoliv na třídě (tak, jak jsme volali metody třídy Math).

```
Animal micina = new Animal();
micina.kind = "kocka";
micina.sound = "mnaaaauuu";
micina.sayHello();
```

18.8.1. Návratové hodnoty

Pokud chceme, aby metoda vracela nějakou hodnotu, pak musíme jednak v hlavičce specifikovat její typ a zároveň musíme v těle metody uvést příkaz return, za kterým následuje proměnná, kterou chceme navrátit. V případě metod s návratovým typem void můžeme (ale nemusíme) pro ukončení vykonávání metody uvést příkaz return, za nímž již přímo následuje středník (konec příkazu). Po příkazu return je vždy metoda okamžitě ukončena, proto za ním již nesmí následovat žádný kód. Metoda ale smí obsahovat více těchto příkazů, pokud jsou v různých větvích programu, čehož budeme využívat později, až se naučíme vytvářet podmínky.

18.8.2. Předávání parametrů

Parametry volání jsou dvojice typ – název proměnné, které oddělujeme čárkou. Je velmi důležité vědět, že zatímco se primitivní datové typy předávají hodnotou (tj. dojde k jejich zkopírování), tak objektové typy se předávají odkazem (samotná reference se ale předává hodnotou). Z toho plyne, že pokud provedeme v metodě změnu hodnoty primitivní proměnné předané parametrem, tak se mimo funkci tato změna neprojeví. Pokud ovšem provedeme změnu na předaném objektu, tak se změna projeví (protože objekt samotný existuje pouze v jedné kopii), naopak přiřazení jiného objektu do reference by nemělo efekt (došlo by k přiřazení do kopie → originál mimo metodu by zůstal nedotčen).

18.8.2.1. Příklad

Jako příklad předáváni parametrů a návratu hodnot si zatím nemůžeme ukázat žádnou „chytrou“ metodu, protože nám k tomu chybí jazykové prostředky (cykly, podmínky). Proto si ukážeme jednu velmi důležitou zásadu objektově orientovaného programování – zapouzdření (encapsulation).

Tato zásada říká, že bychom se měli snažit ukrývat co možná nejvíce informací o implementaci, které uživatelé našich knihoven nepotřebují znát.

V praxi to pro naši třídu Animal znamená, že její členy (kind, sound) budou soukromé (private) a budeme k nim zvenčí přistupovat pomocí veřejných metod. U těchto proměnných je sice šance na změnu jejich implementace diskutabilní, ale uvažujme, že bychom nechali původní variantu (s přímým přístupem) a naše třída by časem evolvovala ve skutečně unikátní kousek technologie, který by při nastavení zvuku syntetizoval také zvuk reálný a uložil jej do nové proměnné.

V tento okamžik bychom se dostali do prekérní situace, kdy bychom již nemohli zakázat přístup do proměnné (přímý přístup již používají stovky zoologických zahrad, tisíce programů, miliony uživatelů) a zároveň přímý přístup neumožňuje žádnou notifikaci syntetizátoru, aby vytvořil nový zvuk při změně textu. Pokud bychom tedy měli například vránu, která by měla řetězec "krááá" a zvuk a změnili bychom napřímo řetězec na "haf" (vrána se naučila štěkat), tak nám ale ve KRRÁÁÁÁ zvuku bude instance vrány pořád krákat.

Kdybychom naproti tomu od počátku přistupovali pomocí metody, tak bychom do ní pouze přidali kód na vytvoření zvuku a vše by fungovalo jak má.

Buďme ale trochu při zemi. Bohatě by stačilo, abychom měli váhu zvířete v kilogramech a zároveň mnoho složité logiky, která by tuto informaci využívala, a v tento okamžik přišel od zadavatele pokyn, že máme v rozhraní třídy poskytovat váhu v imperiálních jednotkách (librách). S přímým přístupem neřešitelné (nutnost přepsat veškerou logiku na libry). S přístupem přes metody stačí přidat přepočet a vnitřek třídy může fungovat beze změny.

```
/**
 * Zviratko, ktere uz ma i nejake operace
 * @author Petrásek
 */
public class Animal {
    /**
     * Druh zviratka
     */
    private String kind;
    /**
     * Zvuk, ktery vydava
     */
    private String sound;

    /**
     * Metoda, ktera vypise pozdrav zviratka
     */
    public void sayHello()
    {
     System.out.println("Ja jsem " + getKind() + " a delam " + get-
Sound());
    }

    /**
     * @return the kind
     */
    public String getKind()
    {
        return kind;
```

```
    }

    /**
     * @param kind the kind to set
     */
    public void setKind(String kind)
    {
        this.kind = kind;
    }

    /**
     * @return the sound
     */
    public String getSound()
    {
        return sound;
    }

    /**
     * @param sound the sound to set
     */
    public void setSound(String sound)
    {
        this.sound = sound;
    }
}
```

18.2.1.2. Gettery a settery

Jak jste si asi všimli, tak názvy metod, které ukládají novou hodnotu proměnné začínají na set a názvy metod, které získávají hodnotu proměnné na get a v obou případech pak následuje jméno proměnné (s velkým počátečním písmenem). Tzv. gettery a settery jsou velmi důležitou konvencí, kterou využívají mnohé frameworky (ty si můžete zatím představit jako velké knihovny). Jedinou výjimkou z této konvence jsou proměnné typu boolean, název jejichž getteru začíná na is (isFriendly()).

Dobrou zprávou při použití vývojového prostředí je, že tyto metody nemusíme psát, budou nám na vyžádání vygenerovány. V Netbeans klikneme pravým tlačítkem do kódu a z menu vybereme Refactor/Encapsulate Fields. Pomocí wizzardu pak naklikáme, které metody chceme nechat vytvořit.

Na tomto místě si ještě povíme o klíčovém slovu this (jenž je součásti generovaných setterů), které slouží jako ukazatel na aktuální instanci. Tímto slovem dochází k rozlišení, zdali používáme proměnnou parametru metody nebo proměnnou dané instance. Kdybychom this nepoužili, tak by v případě setteru u sound došlo k uložení řetězce opět do proměnné parametru (tj. nemělo by to na instanci žádný vliv).

18.9. Podmínky

Základním konstruktem je samotná podmínka – příkaz *if* – za kterým následuje booleovský výraz v kulatých závorkách a blok (nebo jeden příkaz), který se vykoná, je-li podmínka splněna. Podmínka slouží k větvení kódu, buď neúplnému (příkaz *if*), nebo úplnému (kombinace příkazů *if* a *else*). Základní syntaxe podmínek umožňují jeden příkaz pro jenu větev, pokud chceme využít více příkazů, tak je nutné užít složených závorek, do kterých uzavřeme větev. Nepsaný zvyk je uzavírat tělo větve do složených závorek vždy. Pokud píši úplné větvení, tak větev za příkazem *else* se provede, pokud není splněna podmínka.

Příklad neúplného větvení:

```
if(i % 3 == 0) //pokud je číslo dělitelné třemi
{
   System.out.println("Číslo je dělitelné třemi!");
}
```

Příklad úplného větvení:

```
if(i % 3 == 0) //pokud je číslo dělitelné třemi
{
    System.out.println("Číslo je dělitelné třemi!");
}
else
{
    System.out.println("Číslo není dělitelné třemi");
}
```

Pokud chceme postupně ověřit více podmínek stejné úrovně, tak tyto konstrukty složíme za sebe a vznikne složenina *else if(*podmínka*)*. Všimněme si, že *if* je zde v roli onoho jednoho příkazu, který můžeme vepsat přímo za *else*. Jen je nutné si uvědomit jednu důležitou věc, každé *else* se přiřadí k nejbližšímu *if*, platí to hlavně pro vnořené podmínky.

```
int i = (int) (Math.random()*10); //náhodné číslo 0 až 9
if(i % 3 == 0) //pokud je číslo dělitelné třemi
{
      System.out.println("Cislo je delitelne tremi!");
}
else if(i % 2 == 0)
{
      System.out.println("Cislo je sude");
}
else
{
      System.out.println("Cislo neni delitelne tremi, ani neni
sude");
}
```

18.9.1. Ternární výraz

Slouží ke zjednodušení (zkrácení) přiřazovací konstrukce if-else. Jeho interpretace je následující. Pokud platí podmínka před otazníkem, pak operátor vrátí první z odpovědí (oddělené dvojtečkou). Pokud podmínka není uspokojena, pak vrátí druhou odpověď.

Zapisuje se tedy: `podmínka ? odpověď1 : odpověď2;`

Technicky je sice možné skládat řadu ternárních operátorů do sebe a zajistit tak komplexnější rozhodování, ale jedná se o praktiku, jež velmi znepřehledňuje kód, a kterou je z tohoto důvodu zdraví nebezpečné používat.

```
public static String createGreeting(boolean male, boolean married)
{
    String base = "Dear ";
    if(male) // je to muž
    {
        return base + "Mr.";
    }
```

```
        Else //je to žena
       {
            return base + (married ? "Mrs." : "Ms.");
       }
        /*
         * takto by to také šlo zapsat, ale už je to poněkud nepřehledné
         * případě použiti složitějších logických podmínek a vice slože-
ných
         * ternárních operátoru by to byl konec světa
         */
       //return base + (male ? "Mr." : married ? "Mrs." : "Ms.");
   }
```

18.9.2. Příkaz swith

Méně častou možností, jak řídit tok, je příkaz *switch* (přepínač). *Switch*, jak již název napovídá, umožňuje na základě hodnoty proměnné přepínat mezi několika větvemi programu. Proměnná, která složí jako přepínač, může být následujících typů: char, byte, short, int. Tento seznam je platný pro Javu 1.6, ve verzi 1.7 by se měl rozšířit o řetězec.

Samotný zápis switche se od podmínek v některých aspektech liší. S podmínkami analogicky definujeme blok nejprve příkazem *switch*, za kterým následuje v kulatých závorkách proměnná, podle které přepínáme, a samotný blok příkazu.

V tomto bloku ovšem nalezneme vždy nejprve návěští *case*, za ním hodnotu proměnné, která odpovídá danému stavu přepínače a dvojtečku. Za dvojtečkou následují příkazy odpovídající tomuto stavu a příkaz *break*, který slouží k opuštění bloku *switch*. Pokud bychom jej z jakéhokoliv důvodu neuvedli nebo na něj zapomněli, tak se provede i následující case blok.

Příklad:

```
int day = (int) (Math.random() * 7); //čísla 0-6
switch (day)
{
    case 0:
        System.out.println("Je pondeli");
        break;
    case 1:
        System.out.println("Je utery");
        break;
    case 2:
        System.out.println("Je streda");
        break;
    case 3:
        System.out.println("Je ctvrtek");
        break;
    case 4:
       System.out.println("Je patek");
       break;
    case 5:
       System.out.println("Je sobota");
       break;
    case 6:
       System.out.println("Je nedele");
       break;
}
```

18.10. Konstanty

Pro definování konstant používáme klíčové slovo *final* v kombinaci se slovy *public* a *static*. Takto definované proměnné budou veřejné, neměnné a sdílené prostřednictvím třídy. Konstanty pojmenováváme striktně velkými písmeny a čísly, jednotlivá slova v jejich názvu oddělujeme podtržítky.

```
public static final double PI = 3.141592654;
public static final double EULER_NUMBER = 2.718281828;
```

18.11. Cykly

18.11.1. While

Nejjednodušším druhem cyklu je klíčové slovo *while*, po kterém následuje podmínka v kulatých závorkách a blok, který chceme opakovat. Kód umístěný v tomto bloku se provádí tak dlouho, dokud je splněna podmínka.

18.11.1.1. Příklad

Funkce faktoriál čísla n je definována na přirozených číslech jakožto součin všech čísel 1 až n. Protože tato funkce roste obrovskou rychlostí, tak výsledek nebudeme předávat jako celočíselný typ int, ale jako typ double. Double má jednak výrazně větší rozsah hodnot, které může reprezentovat, a má ochranu proti přetečení. Pokud k maximální hodnotě integeru přičteme číslo 1, tak přetečeme zpět do záporných čísel. V případě double dostaneme hodnotu +inf.

```
public static double factorial(int nr)
{
    if(nr < 0) return -1;
    double result = 1;
    int i = 2;
    while(i <= nr)
    {
        result *= i;
        i++;
    }
    return result;
}
```

18.11.2. Do while

Obdobou cyklu *while* je konstrukce *do while*. Fakticky se jedná o totožný příkaz s tím rozdílem, že před blok píšeme klíčové slovo *do* a za blok uvádíme slovo *while* následované podmínkou a středníkem. Podmínka ukončení cyklu je proto kontrolována až po provedení bloku a kód v bloku uvedený tedy proběhne vždy alespoň jednou.

18.11.2.1. Příklad

Euklidův algoritmus slouží k nalezení nejvyššího společného dělitele dvou čísel. Algoritmus v každém svém kroku (iteraci cyklu) vydělí se zbytkem první číslo A druhým číslem B. Pokud zbytek není nulový, tak se do A přiřadí číslo B a zbytek po dělení se přiřadí do právě uvolněné proměnné B, a celá procedura se opakuje. V okamžiku, kdy je zbytek po dělení nulový, tak je v B uložen nejvyšší společných dělitel původních dvou čísel. Pro čísla 133 a 15 by byl postup následující:

$$133 = 8 \cdot 15 + 13$$
$$15 = 1 \cdot 13 + 2$$

$$13 = 6 \cdot 2 + 1$$
$$2 = 2 \cdot 1 + 0$$

Nejvyšším společným dělitelem 133 a 15 je číslo 1.

```
public static int gcd(int a, int b)
{
        if (a < 1 || b < 1) return -1;
        int remainder = 0;
        do
        {
            remainder = a % b; //v tento okamžik v poslední iteraci
platí ona podmínka, že zbytek == 0
             a = b; //ale kvuli dalsi pripadne iteraci posunujeme pro-
menne
             b = remainder; //v b je proto 0, v a je gcd
        }
    while (b != 0);
        return a;
}
```

18.11.3. for

Nejsložitějším, ale nejpoužívanějším cyklem, je smyčka *for*. V kulatých závorkách se za klíčovým slovem skrývá hned trojice příkazů oddělených středníkem. První z nich je deklarace řídicích proměnných cyklu, druhým je podmínka provedení cyklu (tj. ta podmínka, kterou známe z *while*). Posledním příkazem je změna hodnoty řídicích proměnných, která se provede vždy po provedení cyklu.

V první části příkazu můžeme hromadně deklarovat více proměnných a v poslední části můžeme čárkou oddělit více přiřazení nových hodnot. Tohoto postupu se ale obvykle nevyužívá, protože znepřehledňuje kód.

Výhodou tohoto cyklu je, že se všechny potřebné části deklarují na jednom místě a nemůže se stát, že bychom zapomněli například na změnu hodnoty proměnných.

18.11.3.1. Příklad

Fibonacciho posloupnost byla vymyšlena italským matematikem dvanáctého století Leonardem Pisanem jako model přírůstku králíků. Tato posloupnost se řídí těmito pravidly:

- Na začátku se narodí 1 pár králíků.
- Králíci se mohou množit od druhého měsíce života.
- Každý produktivní pár zplodí měsíčně jeden pár králíků.
- Králíci neumírají, pokud jednou začnou plodit, tak plodí pořád.

Matematicky zapsáno tedy spočítáme hodnotu n-tého čísla jako:

$$F(n) = F(n-1) + F(n-2), \quad F(0) = 0, \quad F(1) = 1$$

```
public static int fibonacci(int index)
{
        if (index == 0)
        {
            return 0;
        }
        if (index == 1)
        {
            return 1;
        }
```

```
            int result = 0; //výsledek

            int prePre = 0; //předminule číslo
            int pre = 1; //minule číslu
            for (int i = 1; i < index; i++)//počítáme od druhého indexu
            {
                result = prePre + pre; //výsledek je součet dvou minulých
čísel
                prePre = pre; //v dalším kroku je minule předminulým
                pre = result; //a výsledek je minulým číslem
            }
            return result;
    }
```

18.11.4. Příkazy break a continue

Vykonávání bloku všech zde zmíněných cyklů lze ovlivnit pomocí příkazů *break* a *continue*. Příkaz *break* ukončí vykonávání cyklu a program bude pokračovat za uzavírací závorkou jeho bloku. V případě *continue* cyklus přejde ihned do další iterace.

```
//Vypíše lichá čísla na intervalu <1; 10>
for (int i = 1; i <= 10; i++)
{
    if(i % 2 == 0) continue;
    System.out.println(i);
}

//Vypíše čísla 1 až 6
for (int i = 1; i <= 10; i++)
{
    if(i == 7) break; //způsobí ukončeni vykonávaní cyklu
    System.out.println(i);
}
```

18.11.4.1. Návěští

Break a *continue* se vztahují vždy k bloku, ve kterém jsou obsaženy, což je velmi omezující v případě vnořených cyklů. U těchto občas chceme pokračovat (continue) v až některém z nadřazených cyklů, případně chceme tento nadřazený cyklus opustit (break).

K tomuto v Javě existují návěští, což jsou pojmenování jednotlivých cyklů. Tyto názvy deklarujeme před deklaraci daných cyklů a píšeme za ně vždy dvojtečku.

Pokud se chceme odvolat na nějaký obalující cyklus, tak v příkazech break a continue můžeme název návěští vepsat přímo za tato klíčová slova.

```
Outer:
for (int i = 1; i <= 10; i++) {
    Inner:
    for(int j = 0; j < 20; j++){
        System.out.println(i*j);
        if(i*j == 140) continue Inner;
        else if(i*j == 150) break Outer;
    }
}
```

18.12. Pole

Pole je skupina proměnných stejného typu, které jsou v paměti alokovány za sebou (tj. bez mezer). K těmto proměnným můžeme přistupovat pomocí jejich indexů (pořadí v této posloupnosti, počítáno od 0). Z tohoto způsobu alokace je zřejmé, že pole má fixní délku, kterou nemůžeme nijak změnit (protože paměť v oblasti za polem může být obsazena jiným objektem).

V Javě lze vytvořit pouze pole primitivních datových typů a referencí na objekty, nikoliv však pole objektů samotných. Tato skutečnost není nijak limitující, pouze to znamená, že zatímco jsou reference umístěny jedna za druhou, tak samotné objekty mohou být roztroušeny napříč pamětí.

Technickou poznámkou určenou především pro ty, kteří znají jazyky C a C++ je, že v Javě nelze překročit meze pole (které jsou hlídány virtuálním strojem), v případě použití vyššího indexu, než je velikost pole, program havaruje na výjimce ArrayIndexOutOfBoundsException.

V okamžiku, kdy pole vytvoříme, tak jsou jeho hodnoty přednastaveny do výchozího stavu. V případě celočíselných typů se jedná o hodnotu 0, u typu boolean o hodnotu false, a v případě referencí o ukazatel do prázdna null.

Při deklaraci pole nejprve uvedeme typ hodnot, které do něj budeme ukládat, za něj pár hranatých závorek (označení, že se jedná o pole) a název proměnné tohoto pole. Při inicializaci za rovnítko vepíšeme klíčové slovo new následované typem hodnoty, která bude v poli uložena a v hranatých závorkách uzavřenou celočíselnou hodnotu, jež definuje, jak bude pole velké.

```
//pole osmi integeru
int[] array = new int[8];
//pole čtyř řetězců
String[] stringArray = new String[4];
```

Pro přístup k hodnotám uloženým v poli používáme hranaté závorky vepsané za jméno proměnné daného pole. V těchto závorkách uvádíme index (pořadí počítané od nuly včetně, protože nulou je označena první hodnota v poli) zvolené hodnoty.

```
//pole osmi integeru
int[] array = new int[8];
array[0] = 5; //první hodnota je 5
array[1] = 3; //druha je 3
array[2] = array[1]; //do třetí přiradíme hodnotu na indexu 1, tj. 3

//pole čtyř řetězců
String[] stringArray = new String[4];
stringArray[0] = "Skakal";
stringArray[1] = "pes";
stringArray[2] = "pres";
stringArray[3] = "oves";
System.out.println(stringArray[0]); //skakal
```

Takováto inicializace hodnotami je velmi zdlouhavá a nešikovná. Proto máme při deklaraci s inicializací možnost vytvořit pole alternativním způsobem, kdy na pravou stranu přiřazení napíšeme do složených závorek čárkami oddělené hodnoty, na něž má být pole na jednotlivých indexech inicializováno.

```
String[] stringArray2 = {"Skakal", "pes", "pres", "oves"}; //pole
čtyř řetězců
```

Pro výpis prvků pole můžeme také použít klasický *for* cyklus a proměnnou *length*, která specifikuje délku, na níž bylo pole inicializováno. Proměnnou *length* nám poskytuje objekt pole.

```
/**
 * Vypíše cela čísla uložená v danem poli
 * @param array pole
 */
public static void print(int[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.println(array[i]);
    }
}
```

18.12.1. Vícerozměrná pole

Vícerozměrné pole – pole polí – použijeme, potřebujeme-li postihnout vícerozměrnost dat, která popisujeme. Příkladem je Sudoku.

Při deklaraci vícerozměrných polí postupujeme obdobně jako u pole jednorozměrného, jediným rozdílem je počet závorek, které uvádíme (co dvojice závorek – to jeden rozměr). Na pravé straně přiřazení vždy musíme uvést velikost prvního rozměru (samotného pole polí). Velikost ostatních rozměrů vyplnit nemusíme, ale Java pak tyto rozměry nezinicializuje (to musíme udělat ručně).

Neuvedení dalších rozměrů se může hodit, pokud chceme vytvořit nějakým způsobem nepravidelnou matici – například ve tvaru trojúhelníku.

Pro vícerozměrná pole platí následující poučka: dvojrozměrné pole je časté, trojrozměrné je neobvyklé a více jak trojrozměrné je obvykle chyba návrhu aplikace (jenom málokdo si umí taková data představit).

```
//pole 3 řádky, 4 sloupce
int[][] array2d = new int[3][4];

int[][] array2d2 = new int[3][]; //druhý rozměr neuvádíme
for(int i = 0; i < array2d2.length; i++){ //trojúhelníková matice
    array2d2[i] = new int[i + 1];
}
```

Metoda pro výpis dvojrozměrného pole:

```
/**
 * Vypíše matici (dvojrozměrné pole)
 * @param array matice
 */
public static void print(int[][] array) {
    for (int i = 0; i < array.length; i++) { //průchod pres pole poli
        for (int j = 0; j < array[i].length; j++) { //průchod samotným polem (řádkem)
            System.out.print(array[i][j] + " "); //bez odřádkovaní
        }
        System.out.println(""); //odřádkovaní
    }
}
```

18.12.2. Bubble sort

Nyní si ukážeme první skutečný algoritmus sloužící k seřazení hodnot. Bubble sort je jedním ze tří základních řadících algorimů (Bubble sort, Selection sort, Insertion sort) a je patrně tím nejčastějším způsobem, který laika napadne, když dostane za úkol seřadit pole.

Analogií tohoto algoritmu jsou různě velké bublinky ve vodě. Ty větší stoupají rychleji než ty menší a proto jako první vystoupá ta největší bublinka, pak druhá největší atd.

Bubble sort v každém svém kroku porovnává dva sousední prvky, a pokud je ten větší zařazen vlevo od menšího, tak je prohodí a pokračuje na dalším indexu, kde postup opakuje. Proto pokud byl první porovnávaný prvek největší (nejlehčí), tak se probublá až na konec pole. Pokud byl v poli nějaký jiný lehčí, tak se na konec pole probublá on. Tak či tak, po prvním průchodu je na konci pole ten správný prvek. Nyní již zbývá pouze tento postup n-1 krát (poslední prvek je seřazen triviálně) zopakovat na redukovaném poli (tj. bez již seřazené části) a celé pole bude seřazeno od nejmenší hodnoty po tu nejvyšší.

```
public static void bubbleSort(int[] array)
{
    for (int i = 0; i < array.length - 1; i++)
    {
        for (int j = 0; j < array.length - i - 1; j++)//postupně redukujeme problém
```

```
                {
                    if (array[j] > array[j + 1]) //pokud je vyšší prvek
spatně zaražen
                    {
                        int tmp = array[j]; //tak jej prohodíme
                        array[j] = array[j + 1];
                        array[j + 1] = tmp;
                    }
                }
            }
        }
```

18.13. Dědičnost

Dědičnost nám umožňuje vytvářet hierarchie tříd, ve kterých můžeme o libovolném uzlu říct, že je speciálním případem libovolného ze svých předků. V reálném světě bychom řekli, že člověk je typem savce, auto je dopravním prostředkem, případně že voda je druhem nápoje.

Z druhé strany si ale řekněme, co je špatně použitou dědičností. Nikdy nemůžeme dědit letadlo od racka z důvodu, že má také křídla. Také nemůžeme dědit židli od člověka z titulu, že má člověk má dvě nohy, židle čtyři, tudíž stačí v podtypu dvě nohy přidat a vše bude v pořádku.

18.13.1. Třída Object

S dědičností jsme se již setkali, aniž bychom o tom věděli. Říkali jsme si, že každá třída má určité metody, které jsou již definovány. Jmenovitě jsme si zmiňovali metody *equals* a *hashCode* pro porovnávání objektů a metodu *finalize*, která je zavolána jako poslední pomazání *garbage collectorem* před likvidací objektu. Tyto metody naše objekty implicitně zdědily z třídy *java.lang.Object* (*dokumentace*).

Protože *Object* implicitně dědí všechny třídy, které explicitně nerozšiřují jinou třídu, tak je tato třída kořenem hierarchie všech tříd v Javě.

18.13.2. Extends

Pro vytvoření podtypu stačí v hlavičce třídy ihned po jejím názvu uvést klíčové slovo *extends* a název rozšiřované třídy. Takto vytvořená třída zdědí všechny nesoukromé (včetně package friendly, je-li rozšiřující třída ve stejném balíčku) metody a třídní proměnné předka, které může znovu deklarovat a překrýt.

Naopak třída nezdědí soukromé a statické metody svého předchůdce, protože ty se vztahují pouze ke konkrétní třídě předka. Metody označené jako koncové (*final*) potomek sice zdědí, ale nemůže je překrýt.

18.13.2.1. Specifikátory přístupu

Každý objekt potomka můžeme přetypovat na předka. Z toho plyne, že při překrývání metod můžeme jejich viditelnost pouze uvolňovat. Představme si, že by to takto nefungovalo a zpřísnili bychom přístup v potomkovi z *public* na *private*. Pak by nám stačilo objekt potomka přetypovat na předka, který má veřejný přístup a zavolat tuto v potomkovi soukromou metodu.

18.13.2.2. Vícenásobná dědičnost

V Javě můžeme dědit vždy maximálně od jednoho objektu. Toto je rozdíl oproti jiným objektově orientovaným jazykům, které často toto omezení nemají. Je ale třeba říci, že se do situace, ve které potřebujeme vícenásobnou dědičnost dostaneme pouze velmi zřídka, jelikož Java má kromě klasické dědičnosti ještě tzv. rozhraní (*interface*), kterých třída může realizovat libovolné množství.

18.13.2.3. Super

Při jednotlivých voláních metod podtypů často narazíme na to, že nechceme celou metodu překrýt, pouze k ní chceme přidat další funkcionalitu. V tento okamžik můžeme zavolat *super.jmenoMetody()*, čímž zavoláme funkcionalitu předka. Analogicky k řetězení konstruktorů *this()* můžeme volat konstruktor předka voláním *super()* – toto volání musí být v rámci konstruktoru potomka vždy na prvním místě.

Dalším aspektem volání konstruktoru v rámci hierarchie je, že není příliš vhodné volat kteroukoliv metodu, která může být překryta. Při volání konstruktoru se objekt vytváří postupně z vrchu hierarchie (tzn. od třídy *Object*). Pokud bychom proto volali metodu, která je překrytá v některém z potomků, a která zde využívá fieldy, které ještě nemohly být inicializovány, tak bychom se mohli dočkat havárie programu.

18.13.2.4. Příklad

Mějme třídu zaměstnance (*jméno*, *věk*), která má pouze jednu metodu *work()*, jež identifikuje daného zaměstnance a vypíše, že pracuje (místo tohoto suchého výpisu si můžeme představit složitou aplikační logiku).

Z této třídy pak dědí třída ředitele. Ředitel obsahuje navíc kolekci podřízených zaměstnanců. V metodě *work()* tyto zaměstnance jmenuje (můžeme si představit, že na nich volá složitou aplikační logiku a skládá výsledky jejich práce).

```
package jpz12.example1;

/**
 * Třida zaměstnance
*/
public class Employee {
    protected String name;
    protected int age;

    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }


    public void work(){
        System.out.println("Jsem zamestnanec " + name + " ( " + age +
" )"
                + " a pracuji na dulezitem ukolu");
    }

    /**
     * @return the age
     */
    public int getAge() {
        return age;
    }

    /**
     * @param age the age to set
     */
    public void setAge(int age) {
        this.age = age;
    }

    /**
     * @return the name
     */
```

```
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }
}
```

Na třídě zaměstnance nenalezneme nic překvapivého. Definuje jeden konstruktor s dvěma parametry – což mj. znamená, že třída nemá implicitní bezparametrický konstruktor, ten bychom museli v případě potřeby dodefinovat.

Třída má dále dva fieldy, které reprezentují věk a jméno zaměstnance. K těmto fieldům přistupujeme pomocí getterů a setterů, abychom tak lépe zakryli vnitřní implementaci třídy.

Samotná metoda *work* funguje přesně dle specifikace (vypíše jméno pracovníka, jeho věk a to, že usilovně pracuje).

```
package jpz12.example1;

/**
 * Třida ředitele (zaměstnance, který má ještě specificky úkol)
*/
public class Director extends Employee {
    private Employee[] employees;

    public Director(String name, int age, Employee[] employees) {
        super(name, age); //volame konstruktor predka
        this.employees = employees;
    }

    @Override //rikame prekladaci, ze prekryvame metodu
    public void work() {
        super.work(); //volame metodu work predka

        System.out.println("Prave ridim tyto lidi: ");
        for (int i = 0; i < employees.length; i++) {
            if (i != 0) { //pred prvnim zamestnancem neni ve vypisu
carka
                System.out.print(", ");
            }
            System.out.print(employees[i].getName());
        }
        System.out.println(""); //nakonec odradkujeme
    }

    /**
     * @return the employees
     */
    public Employee[] getEmployees() {
        return employees;
    }

    /**
     * @param employees the employees to set
     */
    public void setEmployees(Employee[] employees) {
        this.employees = employees;
    }
```

```
}
```

Třída ředitele má přístup k fieldům age a name, jelikož byly specifikované jako *protected* (tj. mají zakázaný přístup z vnějšího světa, ale povolený přístup z potomků). Mohli bychom je tedy nastavit přímo v konstruktoru třídy *Director*. Tím bychom ale zbytečně duplikovali kód, v případě změny bychom pak museli měnit kód na dvou místech, čímž bychom si zdvojnásobili šanci, že uděláme nějakou chybu. Proto v konstruktoru třídy *Director* nejprve zavoláme konstruktor předka a poté pouze nastavíme zbylé fieldy (pole zaměstnanců).

V metodě *work()* nejprve zavoláme *work()* třídy zaměstnance, čímž dojde k onomu výpisu. Poté pokračujeme v práci a vypíšeme podřízené zaměstnance.

@Override

Také si všimněme anotace *@Override*, pomocí které říkáme překladači, že překrýváme metodu. Její přítomnost není povinná, vše by fungovalo i bez ní, ale v případě, že bychom udělali typografickou chybu (a metodu v potomkovi nedopatřením pojmenovali jinak), tak překladač zjistí, že jsme chtěli překrývat, ale nepřekrýváme a program nepřeloží. Tímto dojde k odhalení špatně chyby, která je za určitých okolností poměrně špatně dohledatelná.

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Employee em1 = new Employee("Pepa Smetak", 18);
        Employee em2 = new Employee("Franta Prodavac", 20);
        Employee em3 = new Employee("Karel Opravar", 40);

        em1.work();
        em2.work();
        em3.work();

        Employee[] array = {em1, em2, em3}; //vytvoříme pole tří zaměst-
nanců

        Director dir = new Director("Petr Podnikatel", 35, array);
        dir.work();
    }
```

Výstupem z tohoto kódu je:

```
Jsem zamestnanec Pepa Smetak ( 18 ) a pracuji na dulezitem ukolu
Jsem zamestnanec Franta Prodavac ( 20 ) a pracuji na dulezitem ukolu
Jsem zamestnanec Karel Opravar ( 40 ) a pracuji na dulezitem ukolu
Jsem zamestnanec Petr Podnikatel ( 35 ) a pracuji na dulezitem ukolu
Prave ridim tyto lidi:
Pepa Smetak, Franta Prodavac, Karel Opravar
```

18.14. Polymorfizmus

Jak jsme si již několikrát řekli, tak můžeme libovolnou metodu překrýt v potomkovi vlastní implementací. Pokud pak nad daným objektem tuto metodu zavoláme, dojde vždy k vykonání překrývajícího kódu. A to bez ohledu na to, jestli k metodě přistupujeme pomocí reference na objekt předka nebo na objekt potomka (jehož třída obsahuje ono překrytí). Této vlastnosti je dosaženo pomocí pozdní vazby (late binding), kdy je typ objektu, na němž bude metoda volána, rozhodnut až za běhu programu, nikoliv během jeho kompilace.

Toto ovšem platí pouze pro překrývání metod, nikoliv pro jejich přetěžování. Pokud budeme mít na daném objektu dvě metody, které se budou lišit pouze parametrem (jedna pro předka, druhá pro potomka), tak bude volána vždy ta metoda, která má v parametru stejný typ, jako je aktuální reference na objekt. Volání přetížených metod je totiž rozhodováno v době překladu, kdy ještě nemusí být jasné, jestli bude reference ukazovat na objekt předka nebo potomka.

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Employee em1 = new Employee("Pepa Smetak", 18);
        Employee em2 = new Employee("Franta Prodavac", 20);
        Employee em3 = new Employee("Karel Opravar", 40);
        Employee[] array = {em1, em2, em3}; //vytvorime pole tri zamest-
nancu

        Director dir = new Director("Petr Podnikatel", 35, array);

        //Pole, ktere ma 4 ukazatele na predky
        Employee[] array2 = {em1, em2, em3, dir};

        //for each cyklus pro pruchod pres vsechny prvky kolekce (pole)
        for (Employee employee : array2) {
            /* metoda je vzdy zavolana na referenci typu predka,
             * ale i tak dojde k volani prekryte metody na objektu po-
tomka (Director)
             */
            employee.work();
            /**
             * Zatimco u volani metody na objektu dojde vzdy k volani
"spravne" metody,
             * tj. te prekryte, tak pri volani pretizene metody se dle
parametru
             * rozhodne vzdy pro volani metody s parametrem koresponduji-
cim pri prekladu
             * tj. v tomto pripade vzdy pro metodu s parametrem ukazatele
na predka (bez
             * ohledu na to, ze se jedna o objekt potomka)
             */
            testOverloading(employee);
        }
    }

    private static void testOverloading(Employee employee) {
        System.out.println("Method with employee parameter");
    }

    private static void testOverloading(Director director) {
        System.out.println("Method with employee parameter");
    }
```

Pak je výstupem rovněž:

```
Jsem zamestnanec Pepa Smetak ( 18 ) a pracuji na dulezitem ukolu
Method with employee parameter
Jsem zamestnanec Franta Prodavac ( 20 ) a pracuji na dulezitem ukolu
Method with employee parameter
Jsem zamestnanec Karel Opravar ( 40 ) a pracuji na dulezitem ukolu
Method with employee parameter
Jsem zamestnanec Petr Podnikatel ( 35 ) a pracuji na dulezitem ukolu
Prave ridim tyto lidi:
Pepa Smetak, Franta Prodavac, Karel Opravar
Method with employee parameter
```

18.15. Abstraktní třídy

Stejně jako v minulé kapitole si představme, že máme firmu, kde pracují zaměstnanci mnoha různých profesí (ředitele, sekretářky, uklízečky...). Tito zaměstnanci sdílí určité generické chování, které je společné všem profesím, například všichni zdraví na potkání své kolegy.

Zaměstnanci se ale v určitých aspektech liší. Všichni sice pracují (metoda *work()*), ale každá profese dělá něco trochu jiného. Minule jsme tuto situaci řešili tak, že jsme vytvořili třídu obecného zaměstnance, ze které dědili všichni specifičtí činitelé.

Tentokrát ovšem v naší firmě pracují pouze specifičtí zaměstnanci – to znamená, že neexistuje (nemůže existovat) žádná instance obecné profese. Z tohoto hlediska by bylo velmi nešikovné implementovat metody společné nadtřídy (když stejně nebudou nikdy využity). Zároveň by toto nijak nezamezilo konstrukci onoho obecného zaměstnance, který v naší firmě nemůže pracovat (z definice příkladu).

18.15.1. Klíčové slovo abstract

Pro řešení této situace použijeme klíčové slovo *abstract*. Pokud toto slovo vepíšeme do hlavičky třídy, tak z ní nelze vytvářet instance (tím zamezíme vytvoření instance obecného zaměstnance).

Důležitější je ale jeho použití u metod abstraktních tříd. U takto označených metod můžeme vynechat jejich tělo (a napsat místo něj pouze středník), čímž deklarujeme, že tuto metodu musí mít implementovanou všichni neabstraktní potomci. Tohoto později využijeme při polymorfním volání nad jednotlivými objekty

```
/**
 * Abstraktni trida zamestnance
 * @author
 */
public abstract class Employee {
    protected String name;
    protected int age;

    /**
     * Abstraktni tridy samozrejme maji konstruktor, aby sly zinicia-
lizovat
     * jejich fieldy
     * @param name jmeno zamestnance
     * @param age vek zamestnance
     */
    public Employee(String name, int age) {
        this.name = name;
        this.age = age;
    }

    /**
     * Necha zamestnance pracovat.
     * Tato metoda je abstraktni - nema telo, jeji implementace bude
vynucena
     * v prvni neabstraktni tride, ktera oddedi tuto tridu.
     */
    public abstract void work();

    /**
     * Abstrakni tridy taktez mohou mit neabstraktni metody
     */
    public void sayHello(){
        System.out.println("Dobry den, jmenuji se " + name + " a je
mi " + age + " let.");
```

```
        }

        /**
         * @return the age
         */
        public int getAge() {
            return age;
        }

        /**
         * @param age the age to set
         */
        public void setAge(int age) {
            this.age = age;
        }

        /**
         * @return the name
         */
        public String getName() {
            return name;
        }

        /**
         * @param name the name to set
         */
        public void setName(String name) {
            this.name = name;
        }
    }

    /**
     * Trida reditele
     * @author
     */
    public class Director extends Employee {
        private Employee[] employees;

        public Director(String name, int age, Employee[] employees) {
            super(name, age); //volame konstruktor predka
            this.employees = employees;
        }

        @Override //rikame prekladaci, ze prekryvame metodu
        public void work() {
            System.out.println("Prave ridim tyto lidi: ");
            for (int i = 0; i < employees.length; i++) {
                if (i != 0) { //pred prvnim zamestnancem neni ve vypisu
carka
                    System.out.print(", ");
                }
                System.out.print(employees[i].getName());
            }
            System.out.println(""); //nakonec odradkujeme
        }

        /**
         * @return the employees
         */
        public Employee[] getEmployees() {
            return employees;
```

```
        }

        /**
         * @param employees the employees to set
         */
        public void setEmployees(Employee[] employees) {
            this.employees = employees;
        }
    }
    /**
     * Trida reprezentujici sekretarku
     * @author
     */
    public class Secretary extends Employee{

        public Secretary(String name, int age) {
            super(name, age);
        }

        /**
         * Zde je implementace prace zamestnance (v tomto pripade sekre-
tarky)
         */
        @Override
        public void work() {
            System.out.println("Usilovne pracuji, varim kafe, pripravuji
materialy, pisu dopisy.");
        }

    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Employee em1 = new Secretary("Karolina Krasna", 18);
        Employee em2 = new Secretary("Petra Povolna", 20);
        Employee[] array = {em1, em2}; //vytvorime pole tri zamestnancu

        //Objekt reditele
        Employee dir = new Director("Petr Podnikatel", 35, array);

        //Pole techto zamestnancu
        Employee[] array2 = {em1, em2, dir};

        //for each cyklus pro pruchod pres vsechny prvky kolekce (pole)
        for (Employee employee : array2) {
            /*
             * Pomoci polymorfismu se zavola vzdy ta spravna metoda
             */
            employee.sayHello();
            employee.work();
            System.out.println(""); //odřádkujeme po každém zaměstnanci
        }
    }
```

A výstup je:

```
Dobry den, jmenuji se Karolina Krasna a je mi 18 let.
Usilovne pracuji, varim kafe, pripravuji materialy, pisu dopisy.

Dobry den, jmenuji se Petra Povolna a je mi 20 let.
```

```
Usilovne pracuji, varim kafe, pripravuji materialy, pisu dopisy.

Dobry den, jmenuji se Petr Podnikatel a je mi 35 let.
Prave ridim tyto lidi:
Karolina Krasna, Petra Povolna
```

18.16. Rozhraní

Druhou velmi podobnou konstrukcí jsou rozhraní (klíčové slovo *interface*). Základním rozdílem je, že interface obsahuje pouze konstanty a metody bez těla (v tomto případě je neoznačujeme slovem *abstract*). Rozhraní je kontrakt, který specifikuje operace, které má třída splňovat, a který se již nezabývá tím, jak toho bude konkrétně dosaženo.

Velkou výhodou rozhraní oproti abstraktním třídám je, že každá třída může implementovat až mnoho rozhraní, avšak vždy maximálně jednu třídu.

Zatím co pro dědění tříd (a dědění interfaců mezi sebou) využíváme v hlavičce klíčové slovo *extends*, tak pro implementaci rozhraní používáme slovo *implements*.

18.16.1. Konvence

Pro pojmenovávání rozhraní neexistuje žádná široce uznávaná konvence. Časté je využívání přídavných jmen pro rozhraní (*Serializable*, *Movable* atp.), což ale nelze v mnoha případech dodržet (neexistuje přídavné jméno, které by onen kontrakt vystihovalo). Další častou konvencí je využívání přípon *Interface* a *Iface*. Poslední častou variantou je nepoužít žádnou příponu pro rozhraní, ale pojmenovat třídu implementující rozhraní jeho názvem s příponou *Impl* (rozhraní: *Server*, třída: *ServerImpl*).

18.16.2. Příklad

```
package jpz13.example2;

/**
 * Rozhraní pro dopravní prostředky pohupující se vpřed
 * @author
 */
public interface Movable {
    /**
     * Rychlostni limit, jimz se mohou dopravni prostedky pohybovat
     */
    public static final int SPEED_LIMIT = 50;

    /**
     * Pohyb prostredku vpred
     */
    public void moveForward();
}

package jpz13.example2;

/**
 * Rozhrani pro dopravni prostredky, ktere mohou i couvat
 * @author
 */
public interface Reversible {
    /**
     * Operace couvani
     */
    public void reverse();
}
package jpz13.example2;
```

```
        /**
         * Trida vlaku
         * @author
        */
        public class Train implements Movable{
            public void moveForward() {
                System.out.println("Pohybuji se vpred rychlosti " + SPEED_LI-
MIT);
            }
        }

        package jpz13.example2;

        /**
         * Trida auta, ktere se muze pohybovat vpred a vzad
         * @author
         */
        public class Car implements Movable, Reversible{

            public void moveForward() {
                System.out.println("Pohybuji se vpred rychlosti" + (SPEED_LI-
MIT - 5));
            }

            public void reverse() {
                System.out.println("Couvam rychlosti " + (SPEED_LIMIT/10));
            }
        }
```

18.17. Výjimky

Doposud jsme tiše předpokládali, že v našich programech vše půjde dle plánu a nikdy nedojde k nějaké nepředvídané situaci. Pod takovouto situací si můžeme představit chyby programu způsobené programátorem – dělení nulou, sáhnutí mimo rozsah pole, pokus o volání metody na *null*ové referenci a podobně. Druhou skupinou pak mohou být neplatné uživatelské vstupy – pokud uživatel předá do kolonky věk řetězec, pokusí se uložit soubor někam, kam nemá přístup a jiné. Poslední skupinou jsou chyby mimo kontrolu a moc programátora – vyčerpání paměti, zásah operačního systému atd.

Při všech těchto situacích (a mnohých dalších) dojde k vyvolání výjimky, která způsobí okamžité přerušení vykonávání kódu a přechod vlákna do místa, kde je daná situace ošetřena. Pokud takové místo neexistuje, tak je toto vlákno ukončeno.

18.17.1. Fail-fast

V našich programech máme prozatím pouze jedno vlákno. To znamená, že neošetření výjimky vyústí v ukončení celého programu. Sice se může zdát, že bychom se proto měli snažit program za každých okolností zachránit, ale není tomu tak. Obvykle je nejlepší nechat program ihned spadnout, dokud nedošlo k nenávratným škodám na datech, nežli později opravovat běsnění programu, který se dostal do stavu, se kterým jsme nepočítali.

18.17.2. Typy výjimek

Výjimky samotné jsou objekty, které dědí ze speciální hierarchie, jejímž kořenem je třída *Throwable* (*dokumentace*). *Throwable* dále rozšiřují třídy *Error* a *Exception*.

18.17.2.1. Error

Výjimky typu *Error* (dokumentace) bychom neměli v žádném případě ošetřovat – značí totiž kritickou chybu (nedostatek zdrojů pro práci virtuálního stroje, přetečení zásobníku, nenalezení potřebné třídy při classloadingu atp.) – a většině případů je ani ošetřit nemůžeme.

18.17.2.2. Runtime exception

Mezi výjimky dědící z *Exception* patří také podtřída *RuntimeException* (dokumentace). Runtime exception jsou výjimky, které sice nejsou kritické z hlediska samotné možnosti pokračování aplikace (na rozdíl od Error), ale přesto se velmi často neošetřují. Značí totiž obvykle chyby, které způsobil sám programátor (neplatný index pole, volání nad *null*ovým ukazatelem...).

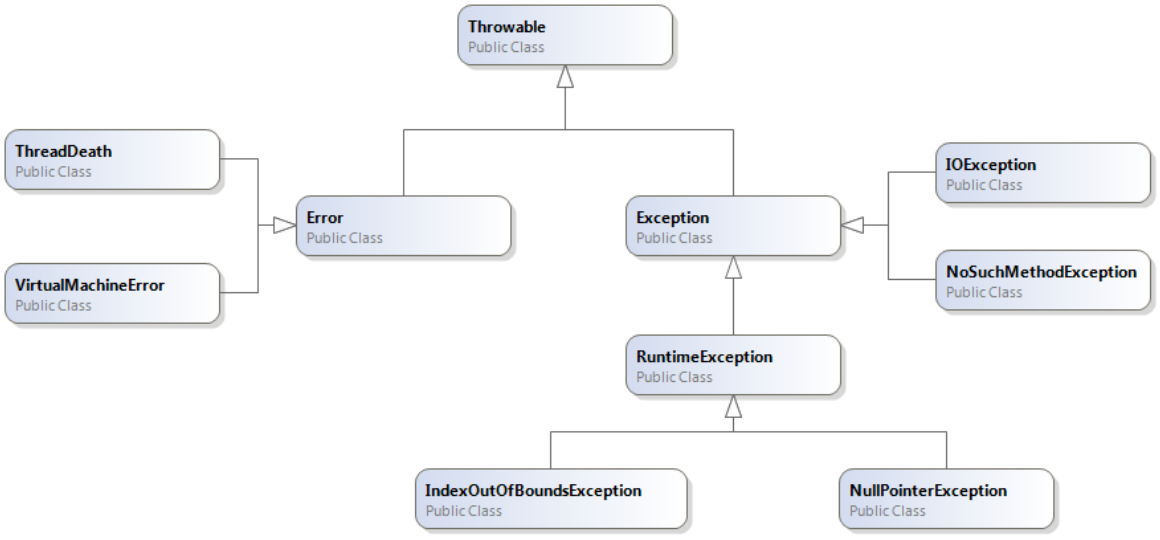
Zvláštní vlastností těchto výjimek je, že nemusíme deklarovat v hlavičce metody možnost jejich vyvolání (klíčové slovo *throws*). Tím je usnadněno vybublání chyby skrz program a jeho případné ukončení.

18.17.2.3. Exception

Zbylé výjimky dědící přímo z *Exception* (dokumentace) značí ty situace, které by se sice neměly stávat, ale na které jsme schopni adekvátně zareagovat. Pokud uživatel zadá neplatný adresář, tak mu vyhubujeme a řekneme, že to má zkusit ještě jednou. Obdobně jsme schopni vyřešit situaci, kdy uživatelské rozhraní nenalezne knihovny potřebné pro *uživatelský motiv* (*look and feel*) na daném systému. V tomto případě můžeme přejít k výchozímu nastavení uživatelského rozhraní.

Protože se jedná, jak jsme si již řekli, o chyby, ze kterých se program může snadno zotavit, tak je vždy musíme uvést v hlavičce metody, jež je může vyvolat. To učiníme pomocí klíčového slova *throws* a seznamu jmen tříd vyvolávaných výjimek. Ve volající metodě se pak musíme rozhodnout, zda-li výjimky ošetříme, nebo necháme vybublat dál (opět je uvedeme v hlavičce).

Tyto výjimky označujeme jako kontrolované (*checked exceptions*).



18.17.3. Vyvolávání výjimek

Výjimku můžeme vyvolat v našem kódu prostřednictvím příkazu *throw* následovaným objektem výjimky.

```
package jpz14;
/**
 * Demonstace vyjimek
```

```
 * @author
 */
public class Main {
    /**
     * @param args the command line arguments
     *
     * Jelikoz metoda throwException vyvolava kontrolovanou vyjimku a
my
     * ji opet nezpracovavame, tak ji musime opet deklarovat,
     * aby probublala ven (a ukoncila vlakno programu)
     */
    public static void main(String[] args) throws Exception {
        throwException();
    }

    /**
     * Metoda, ktera pouze vyvola vyjimku typu Exception
     * Jelikoz se nejedna o runtime Exception a zaroven ji
     * okamzite nezpracovavame, tak ji musime deklarovat v hlavicce
     * metody za klicovym slovem throws
     */
    public static void throwException() throws Exception{
        throw new Exception("Ja jsem zprava vyjimky");
    }
}
```

Výstupem pak je:

```
Exception in thread "main" java.lang.Exception: Ja jsem zprava vy-
jimky
        at jpz14.Main.throwException(Main.java:25)
        at jpz14.Main.main(Main.java:15)
```

Na výstupu tohoto příkladu neodchycené výjimky vidíme tzv. stack trace. Jedná se o výpis volání metod na systémovém zásobníku v okamžiku, kdy došlo k výjimce. Tento výstup nám bude významně pomáhat při odstraňování chyb v aplikaci.

Ve vývojových prostředích můžeme obvykle na jednotlivé záznamy kliknout a dostat se tak přímo na jednotlivé řádky kódu, které byly volány v okamžiku vzniku výjimky.

18.17.4. Try-catch-finally

Každou výjimku bez ohledu na její typ můžeme zpracovat. K tomu používáme bloky *try*, *catch* a *finally*. V bloku *try* uvedeme sekci, která je kritická a vyvolává výjimku. Až v mnoha blocích *catch* můžeme postupně zpracovávat případné výjimky dle jejich typu. Volitelný blok *finally* obsahuje kód, který se vyvolá vždy, bez ohledu na to, k čemu dojde v bloku *try* (dokonce se vyvolá i případě, že dojde k opuštění metody prostřednictvím příkazu *return*).

```
    /**
     * Metoda demonstrujici odchytavani vyjimek
     */
    public static void catchException() {
        try {
            int a = 10 / 0; //vyvola java.lang.ArithmeticException (Run-
time vyjimka)
        } catch (RuntimeException e) {
            e.printStackTrace(); //vypiseme zasobnik na error vystup
            System.out.println("Runtime exception odchycena.");
        } catch (Exception e) {
            e.printStackTrace(); //vypiseme zasobnik na error vystup
            System.out.println("Odchycena vyjimka tridy Exception");
            System.out.println("Tento blok se vypise, pokud nedojde k od-
chyceni"
```

```
                    + "vyjimky v nejakem z predchozich bloku.");
        } finally {
            System.out.println("Obsah bloku finally bude zpracovan
vzdy.");
        }
        System.out.println("Vyjimka byla zpracovana, zde muzeme pokraco-
vat");
    }
```

A výstupem jest:

```
Runtime exception odchycena.
java.lang.ArithmeticException: / by zero
Obsah bloku finally bude zpracovan vzdy.
Vyjimka byla zpracovana, zde muzeme pokracovat
        at jpz14.Main.catchException(Main.java:26)
        at jpz14.Main.main(Main.java:11)
```

Ve výstupu si všimněme, že jsou promíchány jednotlivé výpisy. Nejedná se však o chybu. Pouze o dva různé výstupy (standardního a chybového) vypsané do jedné konzole.

18.17.5. Obvyklé zpracování výjimek

Nyní si představíme několik modelových situací zpracování výjimek, při kterých děláme více než pouhé ošetření pomocí *try-catch-finally*.

18.17.5.1. Neošetření

Základní strategií pro výjimky typu *Error* a velkou část runtime výjimek je jejich neošetření. K chybám (*Error*) by nemělo docházet nikdy, jsou to kritické situace z hlediska aplikace a jakákoliv snaha o záchranu je marná a navíc by mohla způsobit více škody než užitku (můžeme se leda pokusit chybu zalogovat, případně zkusit uložit stav aplikace, ale nikdy bychom neměli pokračovat dále).

V případě *RuntimeException* není situace kritická z hlediska aplikace (pouze z hlediska programu). V řadě případů můžeme aplikaci zachránit odchycením výjimky (ale v tom případě bychom měli spíše použít nějakou kontrolovanou výjimku). Druhým aspektem je, že se jedná vesměs o chyby programátora, které by měly být do značné míry eliminovány při testování aplikace a zároveň nebývá jejich náprava možná.

18.17.5.2. Přebalení na kontrolovanou výjimku

Často se stává, že vyvolaný typ výjimky neodpovídá situaci, ve které se program nachází. Příkladem může být uživatelské rozhraní a pole určené pro číslo. V naší aplikaci pak zadaný řetězec chceme převést na číslo pomocí volání metody *parseInt* třídy *Integer*. Pokud ovšem uživatel zadá řetězec, který číslu neodpovídá, tak metoda vyhodí výjimku *NumberFormatException* (*dokumentace*), která dědí od *RuntimeException*.

18.17.5.2.1. Příklad

Nyní jsme v situaci, kdy je vhodné tuto běhovou výjimku odchytit a vytvořit novou kontrolovanou výjimku odpovídajícího typu. Při konstrukci kontrolované výjimky nesmíme zapomenout předat konstruktoru původní běhovou výjimku. Novou obalující výjimku nakonec vyvoláme pomocí klíčového slova *throws*.

Tímto postupem zajistíme potřebnou konverzi typu výjimky, aniž bychom ztratili informaci předávanou původní výjimkou (obalující výjimka vypíše vždy i výjimku obsaženou). Zároveň přinutíme volající metodu nějakým způsobem nastalou situaci vyřešit (ta může vyčistit špatně vyplněné pole a upozornit uživatele, že má zadat číselnou hodnotu).

18.17.5.3. Přebalení výjimky na RuntimeException

Obdobně se také můžeme dostat do situace, kdy metoda sice vyvolá kontrolovanou výjimku, ale již nemůžeme udělat nic pro napravení dané situace, případně ji chceme nechat probublat až do nějaké vyšší vrstvy aplikace, ve které řešíme výjimky bez ohledu na jejich typ. V tento okamžik je vhodné výjimku přebalit do *RuntimeException* (nebo její libovolné podtřídy) a tuto poté vyvolat.

18.17.5.3.1. Příklad

```
      /**
       * Demonstace vyjimek
       * @author
       */
      public class Main {

          /**
           * @param args the command line arguments
           */
          public static void main(String[] args) {
              int i = 0;
              boolean valid = false;
              do {
                  System.out.println("Zadejte cislo (v rozsahu int):");
                  try {
                      i = readNumber();
                      valid = true; //cislo je v poradku
                  } catch (InvalidInputException ex) {
                      System.out.println("Zadal jste neplatne cislo!");
                  }
              } while (!valid);
              System.out.println("Zadal jste cislo: " + i);
          }

          /**
           * Precte cislo ze standardniho vstupu (konzole)
           * @return Ciselna hodnota vstupu uzivatele
           * @throws InvalidInputException Pokud uzivatel zada neplatny
vstup (ktery neni cislo)
           */
          private static int readNumber() throws InvalidInputException {
              Scanner s = new Scanner(System.in);
              try {
                  return s.nextInt();
              } catch (InputMismatchException e) { //Runtime vyjimka, kte-
rou scanner reaguje na neplatny vstup
                  throw new InvalidInputException(e); //prebalime ji na
checked exception
              }
          }
      }

      /**
       * Kontrolovana vyjimka vlastniho typu oznacujici, ze v nasem
priklade doslo k chybe
       * na vstupu
       * @author
       */
      class InvalidInputException extends Exception {

          public InvalidInputException(Throwable cause) {
```

```
            super(cause);
        }
    }
```

18.18. Vnořené třídy

Nejjednodušší variantou zanořených typů jsou vnořené třídy/rozhraní (*nested/embedded class/interface*). Jsou to typy, které jsou definovány v rámci jiné třídy mezi jejími metodami a jsou uvozeny klíčovým slovem *static*. Vnořené třídy definované v rámci rozhraní jsou statické implicitně.

V případě vnořených tříd a rozhraní se jedná pouze o pojmenovávací konvenci, kterou dáváme najevo, že k sobě obalovaný a obalující typ nějakým způsobem patří, ale zároveň mohou existovat nezávisle na sobě.

Pro vnořené třídy (a veškeré zanořené typy obecně) platí, že mají přístup k soukromým proměnným obalujícího typu, stejně jako má obalující typ přístup k jejich proměnným.

18.18.1. Příklad

Příkladem může být mobilní telefon a *SIM* karta. Telefon může existovat bez *SIM* karty, stejně tak *SIM* karta může existovat bez mobilního telefonu. Zároveň ale cítíme, že mezi těmito objekty existuje určitá pevnější vazba.

```
/**
 * Trida mobilniho telefonu
 * @author
 */
public class CellPhone {

    /**
     * International Mobile Equipment Identity
     *
     * Field je final - imai nelze zmenit
     */
    private final String imei;
    private SIMCard card;

    /**
     * Konstruktor, kazdy telefon ma imai
     * @param imei
     */
    public CellPhone(String imei) {
        this.imei = imei;
    }

    /**
     * @return the imai
     */
    public String getImei() {
        return imei;
    }

    /**
     * @param card the card to set
     */
    public void setCard(SIMCard card) {
        this.card = card;
    }

    @Override
```

```
        public String toString() {
            String phoneNumber = null;
            if (this.card != null) {
                phoneNumber = "Obsahuji SIM kartu s telefonnim cislem: "
+ card.phoneNumber + "."; //primy pristup k fieldum vnitrni tridy
            } else {
                phoneNumber = "Neobsahuji zadnou SIM kartu.";
            }
            return "Jsem mobilni telefon a mam imei: " + imei + ". " +
phoneNumber;
        }

        /**
         * Vnorena trida SIM karty
         */
        public static class SIMCard {

            private String phoneNumber;

            public SIMCard(String phoneNumber) {
                this.phoneNumber = phoneNumber;
            }

            /**
             * Setter na telefonni cislo (coz snad v realnem svete jde :-
])
             * @param phoneNumber
             */
            public void setPhoneNumber(String phoneNumber) {
                this.phoneNumber = phoneNumber;
            }

            public String getPhoneNumber() {
                return phoneNumber;
            }
        }
    }
```

Třídu *CellPhone.SIMCard* můžeme *import*ovat úplně stejným způsobem, jako to děláme s třídami různých balíčků. Záleží pouze na vkusu programátora, zda-li si přeje tímto způsobem zdůraznit vazbu, nebo jestli bude volat třídu přímo.

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        /**
         * Vytvorime telefon, jak jsme zvykli
         */
        CellPhone phone = new CellPhone("050277-22-222751-3");

        /**
         * Vytvorime sim kartu, je zvykem pouzit teckovaci notaci z oba-
lujici tridy
         * Jinak zde zadny rozdil neni. Vnorenou tridou pouze rikame, ze
se jedna
         * o SIMCard k mobilnimu telefonu (ne k cemukoliv jinemu, kdyby
ji to nahodou
         * take melo).
         */
        CellPhone.SIMCard card = new CellPhone.SIMCard("77623456");
        phone.setCard(card);
```

```
        /**
         * Nechame vypsat objekt telefonu, pripominam, ze se zavola me-
toda toString
         */
        System.out.println(phone);
    }
```

Výstupem je:

```
    Jsem mobilni telefon a mam imei: 050277-22-222751-3. Obsahuji SIM
kartu s telefonnim cislem: 77623456.
```

18.19. Anonymní třídy

Anonymní třída (*anonymous class*) je lokální třída na jedno použití (jelikož je na daném místě rovnou použita, tak ji není zapotřebí pojmenovávat – proto anonymní).

Možnost použít danou třídu pouze jednou se může na první pohled jevit jako nešikovná, ale v reálném světě nepotkáte téměř žádnou pravověrnou lokální třídu, pouze třídy anonymní. Je to dáno především tím, že lokální třídy jsou z principu téměř vždy pouze na jedno použití – kdyby nebyly, tak bychom je buď vytvořili jako běžnou třídu, nebo alespoň jako třídu vnitřní.

18.19.1. Konstrukce

Samotná syntaxe konstrukce anonymní třídy je poněkud zvláštní. Nejprve uvádíme klíčové slovo *new*, jelikož rovnou vytváříme instanci této třídy. Následuje název abstraktní třídy, nebo rozhraní, které anonymní třída implementuje a kulaté závorky, do kterých můžeme vepsat argumenty volání případného konstruktoru (je-li anonymní třída odvozena od abstraktní třídy). Poslední položkou jsou složené závorky, jež obsahují metody abstraktní třídy nebo rozhraní, které musíme doimplementovat.

Technicky vzato lze vytvářet anonymní třídy také od neabstraktních tříd a v definici překrýt některé metody. Tuto konstrukci jsem však zatím v žádném reálném projektu nepotkal.

18.19.1.1. Příklad

Při použití anonymní třídy by předchozí příklad vypadal následovně (takto byste jej nejpravděpodobněji potkali v reálném kódu):

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        String[] array = {"žluťoučký kůň", "dědek", "Máj", "májka", "cha-
lupa", "cihla", "řízek", "Řezno"};

        Arrays.sort(array, new Comparator(){
            public int compare(Object o1, Object o2) {
                Collator c = Collator.getInstance(new Locale("cs",
"CZ"));
                return c.compare(o1, o2);
            }
        }); //strednik ukoncuje prikaz Arrays.sort()

        for(String s : array){
            System.out.println(s);
        }
    }
```

18.20. Spojový seznam

Spojový seznam je datový kontejner, který se skládá z mnoha lineárně provázaných uzlů. Každý uzel obsahuje část obsahující uloženou entitu a z ukazatel na další prvek (jednosměrně zřetězený seznam), případně i ukazatel na prvek předchozí (obousměrně zřetězený seznam).

18.20.1. Rozhraní (spojového) seznamu

Nejprve si specifikujme operace, které musí splňovat každý seznam. Tato abstrakce nám v případě potřeby umožní nahradit spojový seznam jinou implementací.

```
/**
 * Rozhrani, jez definuje operace, ktere musi splnit kazda implementace seznamu
 * @author
*/
public interface ListIface {

    /**
     * Vlozi na konec seznamu prvek
     * @param i prvek k vlozeni
     */
    public void add(int i);

    /**
     * Vrati prvek na indexu i
     * @param i index prvku
     * @return prvek na indexu i
     */
    public int get(int i);

    /**
     * Smaze prvek na indexu i
     * @param i index mazaneho prvku
     */
    public void remove(int i);

    /**
     * Dotaz na delku seznamu
     * @return delka seznamu
     */
    public int size();
}
```

18.20.2. Implementace

Samotný spojový seznam budeme implementovat jako jednosměrně zřetězený s ukazateli na první a poslední prvek. Ukládané hodnoty budou primitivního typu *int*.

```
/**
 * Jednosmerne zretezeny spojovy seznam
 * @author
 */
public class MyLinkedList implements ListIface {

    private Node first;
    private Node last;
    private int size;

    /**
```

```
     * Konstruktor spojoveho seznamu
     */
    public MyLinkedList() {
        this.size = 0;
    }

    /**
     * Vnitrni trida reprezentujici jeden uzel spojoveho seznamu
     */
    private class Node {

        private int value;
        private Node next;

        private Node(int value) {
            this.value = value;
        }
    }
}
```

Třídu spojového seznamu pojmenujeme *MyLinkedList*, aby nedocházelo ke kolizi jmen s již předpřipravenou třídou *java.util.LinkedList* (dokumentace). Třída bude implementovat operace, které jsme si definovali v rozhraní *ListIface*.

Objekt spojového seznamu bude obsahovat ukazatel na první uzel *first*, ukazatel na poslední uzel *last* a hodnotu *size* obsahující počet prvků uložených v seznamu.

Reference na první uzel použijeme pro průchod skrz seznam (je jednosměrný). Reference na poslední prvek zjednoduší operaci ukládání nové hodnoty, protože nebudeme muset napřed proiterovat celý seznam (vkládáme nakonec). Obdobným způsobem proměnná size zjednodušší dotaz na délku seznamu.

Konstruktor obsahuje pouze jedinou operaci – inicializaci proměnné *size* na hodnotu 0.

18.20.2.1. Třída Node

Základem spojového seznamu je soukromá vnitřní třída *Node*. Tuto třídu jsme vytvořili jako vnitřní a soukromou, protože ji takto v duchu zásad zapouzdření (*encapsulation*) dokonale skryjeme před okolním světem.

Node obsahuje uloženou hodnotu *value* a referenci na další prvek *next* (*null*ový v případě, že se jedná o poslední uzel). *Gettery* a *settery* jednotlivých proměnných nejsou nevyhnutelně nutné, jelikož má k typu *Node* přístup pouze třída *MyLinkedList* (a ta může použít přímý přístup).

18.20.2.1.1. Metoda add

```
/**
 * Vlozi na konec seznamu prvek
 * @param i prvek k vlozeni
 */
public void add(int i) {
    Node n = new Node(i);
    if (size == 0) {
        this.first = n;
        this.last = n;
    } else {
        this.last.next = n;
        this.last = n;
    }
    size++;
}
```

Metoda *add* přidá zadanou hodnotu na konec seznamu. Jako první musíme vytvořit nový uzel seznamu pro předávanou hodnotu. Pokud je seznam prázdný, tak se nově přidávaný uzel stane prvním (*first*) a posledním (*last*) prvkem seznamu. Pokud není seznam prázdný, tak vybereme pomocí reference *last* poslední prvek seznamu a do jeho proměnné *next* vložíme referenci na nově přidávaný uzel. Jako poslední krok v každém případě inkrementujeme proměnnou *size*.

18.20.2.1.2. Metoda get

Metoda *get* vrátí prvek na zadaném indexu seznamu. Nejprve ošetříme chyby, které mohou nastat – předání buď příliš vysokého nebo příliš nízkého (záporného) indexu. Na obě chyby zareagujeme adekvátní nekontrolovanou výjimkou (jedná se o chybu programátora).

Nyní pro iterujeme celý seznam až na daný index a vrátíme hodnotu uschovanou v daném uzlu.

```
      /**
       * Vrati prvek na indexu i
       * @param i index prvku
       * @throws IndexOutOfBoundsException pokud je i vyssi nebo rovna
delce seznamu
       * @throws IllegalArgumentException pokud je i zaporne
       * @return prvek na indexu i
       */
      public int get(int i) {
          if (i >= size) {
              throw new IndexOutOfBoundsException("Mimo meze index:" + i +
", size:" + this.size);
          }
          if (i < 0) {
              throw new IllegalArgumentException("Index mensi nez 0");
          }
          Node curr = first;
          for (int j = 0; j < i; j++) {
              curr = curr.next;
          }
          return curr.value;
      }
```

18.20.2.1.3. Metoda remove

```
      /**
       * Smaze prvek na indexu i
       * @param i index mazaneho prvku
       * @throws IndexOutOfBoundsException pokud je i vyssi nebo rovna
delce seznamu
       * @throws IllegalArgumentException pokud je i zaporne
       */
      public void remove(int i) {
          if (i >= size) {
              throw new IndexOutOfBoundsException("Mimo meze index:" + i +
", size:" + this.size);
          }
          if (i < 0) {
              throw new IllegalArgumentException("Index mensi nez 0");
          }
          if (i == 0) {
              first = first.next;
          } else {
              Node curr = first;
              for (int j = 0; j < i - 1; j++) { //najdeme predchozi
                  curr = curr.next;
```

```
            }
            curr.next = curr.next.next; //a mazany prvek vynechame
            if (i == size - 1) { //pokud mazeme posledni
                last = curr;
            }
        }
        size--;
    }
```

Při odstraňování prvku opět nejprve zkontrolujeme platnost zadaného indexu. Pokud mažeme první prvek, tak nám stačí změnit ukazatel na první prvek seznamu (*first*) na prvek následující. V ostatních případech nalezneme prvek, který je v seznamu před mazaným prvkem (označíme jej *curr*) a změníme jeho ukazatel na *next* na prvek, který je za mazaným prvkem (mazaný prvek vynecháme). Pokud jsme mazali poslední prvek, pak je nyní posledním prvkem *curr*. Na závěr dekrementujeme proměnnou *size*.

Pro úplnost si připomeňme, že poté, co na daný uzel zmizí poslední reference, bude odstraněn pomocí *garbage collectoru*.

18.20.2.1.4. Metoda size

```
    /**
     * Dotaz na delku seznamu
     * @return delka seznamu
     */
    public int size() {
        return this.size;
    }
```

Metoda *size* pouze vrátí předpočítanou délku seznamu.

18.20.2.1.5. Metoda toString

```
    /**
     * Klasicka toString metoda, vraci textovou reprezentaci objektu
     * @return textova reprezentace objektu
     */
    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        Node curr = first;
        for (int i = 0; i < this.size; i++) {
            builder.append(curr.value);
            builder.append(" ");
            curr = curr.next;
        }
        return builder.toString();
    }
```

Nakonec ještě překryjeme metodu *toString* zděděnou z *Object*. Tato metoda vrací textovou reprezentaci daného objektu.

V této metodě využíváme *StringBuilder* (*dokumentace*), což je obdoba *ArrayListu* pro znaky. Kdybychom používali pouze operátor zřetězení +, tak bychom každým jeho použitím vytvořili nový řetězec, který bychom v zápětí zahodili (jakožto vstup nového zřetězení). Tento postup by byl velmi nehospodárný.

18.20.3. Příklad

```
    /**
     * Jednosmerne zretezeny spojovy seznam
     * @author
     */
    public class MyLinkedList implements ListIface {
```

```
        private Node first;
        private Node last;
        private int size;

        /**
         * Konstruktor spojoveho seznamu
         */
        public MyLinkedList() {
            this.size = 0;
        }

        /**
         * Vlozi na konec seznamu prvek
         * @param i prvek k vlozeni
         */
        public void add(int i) {
            Node n = new Node(i);
            if (size == 0) {
                this.first = n;
                this.last = n;
            } else {
                this.last.next = n;
                this.last = n;
            }
            size++;
        }

        /**
         * Vrati prvek na indexu i
         * @param i index prvku
         * @throws IndexOutOfBoundsException pokud je i vyssi nebo rovna
delce seznamu
         * @throws IllegalArgumentException pokud je i zaporne
         * @return prvek na indexu i
         */
        public int get(int i) {
            if (i >= size) {
                throw new IndexOutOfBoundsException("Mimo meze index:" +
i + ", size:" + this.size);
            }
            if (i < 0) {
                throw new IllegalArgumentException("Index mensi nez 0");
            }
            Node curr = first;
            for (int j = 0; j < i; j++) {
                curr = curr.next;
            }
            return curr.value;
        }

        /**
         * Smaze prvek na indexu i
         * @param i index mazaneho prvku
         * @throws IndexOutOfBoundsException pokud je i vyssi nebo rovna
delce seznamu
         * @throws IllegalArgumentException pokud je i zaporne
         */
        public void remove(int i) {
            if (i >= size) {
                throw new IndexOutOfBoundsException("Mimo meze index:" +
i + ", size:" + this.size);
```

```
        }
        if (i < 0) {
            throw new IllegalArgumentException("Index mensi nez 0");
        }
        if (i == 0) {
            first = first.next;
        } else {
            Node curr = first;
            for (int j = 0; j < i - 1; j++) { //najdeme predchozi
                curr = curr.next;
            }
            curr.next = curr.next.next; //a mazany prvek vynechame
            if (i == size - 1) { //pokud mazeme posledni
                last = curr;
            }
        }
        size--;
    }

    /**
     * Dotaz na delku seznamu
     * @return delka seznamu
     */
    public int size() {
        return this.size;
    }

    /**
     * Klasicka toString metoda, vraci textovou reprezentaci objektu
     * @return textova reprezentace objektu
     */
    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        Node curr = first;
        for (int i = 0; i < this.size; i++) {
            builder.append(curr.value);
            builder.append(" ");
            curr = curr.next;
        }
        return builder.toString();
    }


    /**
     * Vnitrni trida reprezentujici jeden uzel spojoveho seznamu
     */
    private class Node {

        private int value;
        private Node next;

        private Node(int value) {
            this.value = value;
        }
    }
}

/**
 * @param args the command line arguments
```

```
 */
public static void main(String[] args) {
    ListIface l = new MyLinkedList();

    l.add(0);
    l.add(10);
    l.add(20);
    l.add(30);
    l.add(40);

    //vypiseme spojovy seznam pomoci toString metody
    System.out.println(l);
    System.out.println(l.get(2)); //20


    l.remove(1); //odstranime prvek na prvnim indexu
    System.out.println(l.get(2)); //30

    l.remove(0);
    System.out.println(l);
    System.out.println(l.size()); //3
}

0 10 20 30 40
20
30
20 30 40
3
```

V kódu volání používáme na levé straně přiřazení rozhraní *ListIface*. Stejně tak bychom postupovali v libovolném jiném případě, kdy by vytvářená entita měla vyhovující interface.

Tomuto postupu se říká *programování proti rozhraní*. Výhody jsou poměrně jasné – pokud se rozhodneme změnit implementační třídu, tak ji vyměníme na jediném místě (tam kde vytváříme instanci) a zbytek programu může zůstat totožný. Kdybychom *programovali proti implementaci* – implementační třída by se objevovala na levých stranách přiřazení, v hlavičkách metod a podobně – tak bychom v případě výměny této třídy museli přepsat skoro celý program.

18.21. Generika

Javovská generika (*generics*), přidaná v Javě 1.5, řeší tento problém. Umožňují vytvářet parametrizované datové typy, kterým při jejich konstrukci (volání) předáme typový parametr označující třídu, se kterou mají pracovat.

Pokud bychom tedy chtěli do seznamu ukládat *Integer*y, tak předáme patřičný typový parametr, pokud se o dva řádky níže rozhodneme pro nový seznam řetězců, předáme typ *String*. Takto parametrizovaný seznam bude mít také silnou typovou kontrolu – do seznamu řetězců nevložíme číslo (a naopak), stejně tak metoda *get* vždy vrátí objekt předaného typu.

18.21.1. Parametrizování třídy/rozhraní

Třídu/rozhraní parametrizujeme pomocí čárkou oddělených typových parametrů umístěných do špičatých závorek ihned za název třídy/rozhraní. Typový parametr je řetězec, který bude při kompilaci nahrazen předaným typem.

18.21.1.1. Pojmenování typového parametru

Pojmenovávání typových parametrů podléhá stejným pravidlům jako názvy proměnných (musí začínat písmenem, dolarem (**$**) nebo podtržítkem (**_**)). Dle konvence používáme pro jména typových parametrů pouze velká písmena (často pouze jedno velké písmeno).

18.21.1.2. Parametr v rozhraní ListIface

```
/**
 * Rozhrani, jez definuje operace, ktere musi splnit kazda implemen-
tace seznamu
 * @author
*/
public interface ListIface<ENTITY> {

    /**
     * Vlozi na konec seznamu prvek
     * @param i prvek k vlozeni
     */
    public void add(ENTITY i);

    /**
     * Vrati prvek na indexu i
     * @param i index prvku
     * @return prvek na indexu i
     */
    public ENTITY get(int i);

    /**
     * Smaze prvek na indexu i
     * @param i index mazaneho prvku
     */
    public void remove(int i);

    /**
     * Dotaz na delku seznamu
     * @return delka seznamu
     */
    public int size();
}
```

Zde jsme předali rozhraní seznamu z minulého dílu typový parametr *ENTITY*. Všimněme si, že nyní metoda *add* obsahuje generický parametr volání a metoda *get* vrací objekt totožného typu. Pokud nyní toto rozhraní implementujeme, tak máme jistotu, že námi vytvořená třída bude akceptovat (a vracet) objekty libovolné předem specifikované třídy.

18.21.1.3. Typový parametr třídy MyLinkedList

```
/**
 * Jednosmerne zretezeny spojovy seznam
 * @author
 * @param <ENTITY> typovy parametr specifikujici ukladany typ
 */
public class MyLinkedList<ENTITY> implements ListIface<ENTITY> {

    private Node first;
    private Node last;
    private int size;

    /**
     * Konstruktor spojoveho seznamu
     */
    public MyLinkedList() {
```

```
            this.size = 0;
        }

        /**
         * Vlozi na konec seznamu prvek
         * @param i prvek k vlozeni
         */
        public void add(ENTITY i) {
            Node n = new Node(i);
            if (size == 0) {
                this.first = n;
                this.last = n;
            } else {
                this.last.next = n;
                this.last = n;
            }
            size++;
        }

        /**
         * Vrati prvek na indexu i
         * @param i index prvku
         * @throws IndexOutOfBoundsException pokud je i vyssi nebo rovna
delce seznamu
         * @throws IllegalArgumentException pokud je i zaporne
         * @return prvek na indexu i
         */
        public ENTITY get(int i) {
            if (i >= size) {
                throw new IndexOutOfBoundsException("Mimo meze index:" +
i + ", size:" + this.size);
            }
            if (i < 0) {
                throw new IllegalArgumentException("Index mensi nez 0");
            }
            Node curr = first;
            for (int j = 0; j < i; j++) {
                curr = curr.next;
            }
            return curr.value;
        }

        /**
         * Smaze prvek na indexu i
         * @param i index mazaneho prvku
         * @throws IndexOutOfBoundsException pokud je i vyssi nebo rovna
delce seznamu
         * @throws IllegalArgumentException pokud je i zaporne
         */
        public void remove(int i) {
            if (i >= size) {
                throw new IndexOutOfBoundsException("Mimo meze index:" +
i + ", size:" + this.size);
            }
            if (i < 0) {
                throw new IllegalArgumentException("Index mensi nez 0");
            }
            if (i == 0) {
                first = first.next;
            } else {
                Node curr = first;
```

```
            for (int j = 0; j < i - 1; j++) { //najdeme predchozi
                curr = curr.next;
            }
            curr.next = curr.next.next; //a mazany prvek vynechame
            if (i == size - 1) { //pokud mazeme posledni
                last = curr;
            }
        }
        size--;
    }

    /**
     * Dotaz na delku seznamu
     * @return delka seznamu
     */
    public int size() {
        return this.size;
    }

    /**
     * Klasicka toString metoda, vraci textovou reprezentaci objektu
     * @return textova reprezentace objektu
     */
    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        Node curr = first;
        for (int i = 0; i < this.size; i++) {
            builder.append(curr.value);
            builder.append(" ");
            curr = curr.next;
        }
        return builder.toString();
    }

    /**
     * Vnitrni trida reprezentujici jeden uzel spojoveho seznamu
     */
    private class Node {

        private ENTITY value;
        private Node next;

        private Node(ENTITY value) {
            this.value = value;
        }
    }
}
```

V hlavičce třídy jsme stejně jako u rozhraní specifikovali typový parametr. Tento parametr jsme rovnou poslali o úroveň výš do rozhraní, abychom specifikovali také jeho generický typ.

Generikum dále využíváme v metodách *get* a *add* a ve vnitřní třídě *Node*. Oproti původní implementaci jsme jím pouze nahradili příslušné výskyty primitivního typu *int* (který byl původně ukládán).

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    ListIface<Integer> l = new MyLinkedList<Integer>();
```

```
        l.add(0);
        l.add(10);

        System.out.println(l.get(1)); //10

        ListIface<String> l2 = new MyLinkedList<String>();
        l2.add("pes");
        System.out.println(l2.get(0)); //pes

        //Neznamy (libovolny) typovy parametr rozsirujici tridu String
        ListIface<? extends String> l3 = l2;
        l3.get(0);

        //Nepouziti typoveho parametru
        ListIface l4 = new MyLinkedList();
        l4.add(new Object());
    }
```

V prvním volání vytváříme proměnnou typu *ListIface* s typovým parametrem *Integer* a přiřazujeme do ní odpovídající instanci seznamu. Tento seznam můžeme bezpečně používat jako kontejner na celá čísla. Další seznam (*l2*) funguje identicky, ale pro řetězce (*String*). Do seznamu *l3* můžeme ukládat pouze typy, které rozšiřují třídu *String* a *String* samotný (pomiňme pro jednoduchost příkladu, že je třída *String* finální). Poslední možností je nepoužít při volání generika vůbec.

18.21.2. Generika umí více

Generika jsou mocnou zbraní, která umí výrazně více, než jsme si ukázali. Lze vytvářet generické typy generických typů. Specifikovat nejen, že třída je potomkem jiné třídy, ale že také implementuje některá rozhraní. Dokonce můžeme říci, že se generický typ váže pouze k nadtypům nějaké třídy. V neposlední řadě můžeme dokonce typovat jednotlivá volání metod.

Tyto vlastnosti mají ale poměrně specifické použití, proto si je ukážeme až v případě, že na ně narazíme.

18.21.3. Co generika neumí

Generika ale bohužel nejsou všemocnou zbraní. Jejich omezení plyne ze skutečnosti, že jsou jakýmsi cukrem nad kompilátorem (v bytekódu neexistují).

Kompilátor pro nás emuluje v bytekódu generika stejným způsobem, jakým bychom jich docílili my, kdyby neexistovala (tj. používali bychom Javu 1.4 a starší). Místo generických typů dosadí třídu *Object* a na patřičných místech provádí přetypování na třídu, kterou vyžadujeme.

Z tohoto důvodu nemůžeme používat typové parametry za běhu programu (v daný okamžik již neexistují). Pokud bychom informaci o třídě typových parametrů přesto potřebovali, tak by bylo nejjednodušší dané typy předat konstruktoru (při konstrukci objektu je znát musíme) a tyto třídy si později vracet.

```
    /**
     * Demonstrace omezeni generik
     * @author
     */
    public class Foo<BAR> {
        public Class<?> getParameterClass(){
            return BAR.class; //toto nelze udelat, typovy parametr jiz
neexistuje
        }
    }
```

18.22. Iterátor

Z tohoto důvodu zavedeme do seznamu novou strukturu – iterátor. Iterátor si můžeme představit jako ukazovátko do seznamu, které si pamatuje pozici současného prvku. Iterátoru se můžeme zeptat, zda-li existuje v seznamu další prvek, a pokud ano, tak si jej můžeme vyžádat. Žádost o další prvek vyústí v přesun ukazovátka o jednu pozici dále. Tímto způsobem projde iterátor seznam v $n$krocích.

Implementace

Iterátor přidáme spojovému seznamu prostřednictvím implementace rozhraní *Iterable* (*dokumentace*), čímž získáme navíc jednu velmi zajímavou vlastnost – naši kolekci budeme moci procházet pomocí *for-each* cyklu.

```
public interface ListIface<ENTITY> extends Iterable<ENTITY> {
.
.
.
}
```

V hlavičce našeho rozhraní *ListIface* předáme rozhraní *Iterable* typový parametr *ENTITY*, který zajistí, že metoda iterator bude vracet iterátor odpovídajícího typu (iterujícího přes objekty typu *ENTITY*).

Třída *Iterator* má tři metody. Tou první je *hasNext* – dotaz na existenci dalšího prvku, druhou *next* – vracející další prvek a v neposlední řadě nepovinnou metodu *remove*, která odstraní současný prvek.

Iterátor, který vytvoříme, bude nejjednodušší možný – nebude implementovat metodu *remove* a nebude *fail-fast*.

Fail-fast implementace iterátoru by vyhodila výjimku v okamžiku, kdy by byl seznam změněn jinak než iterátorem (byl by např. smazán prvek, na který iterátor ukazuje – ten by pak totiž byl v nedefinovaném stavu).

```
/**
 * Jednosmerne zretezeny spojovy seznam
 * @author
 * @param <ENTITY> typovy parametr specifikujici ukladany typ
 */
public class MyLinkedList<ENTITY> implements ListIface<ENTITY> {
.
.
.
    @Override
    public Iterator iterator() {
        return new LinkedListIterator();
    }

    /**
     * Jednoduchy iterator pro spojovy seznam
     * Neresi situaci, kdy je iteratoru pod rukou pozmena kolekce
(tj. neni fail fast)
     */
    private class LinkedListIterator implements Iterator<ENTITY> {

        private Node currNode;
        private boolean start;

        public LinkedListIterator() {
            this.currNode = null;
            this.start = true;
        }
```

```
            @Override
            public boolean hasNext() {
                if (start) {
                    return first != null;
                }
                return currNode.next != null;
            }

            @Override
            public ENTITY next() {
                if (start) {
                    start = false;
                    currNode = first;
                } else {
                    currNode = currNode.next;
                }
                return currNode.value;
            }

            @Override
            public void remove() {
                throw new UnsupportedOperationException("Remove is not
supported");
            }
        }
```

Metoda *iterator*, jak jsme si již řekli, vrací objekt iterátoru. Technicky vzato bychom jej mohli vytvořit jako anonymní třídu, ale to je v tomto případě spíše záležitost vkusu.

Samotný iterátor (typovaný kvůli návratové hodnotě metody *next*) obsahuje referenci na aktuální prvek a příznak *start*.

Start využijeme na počátku průchodu, protože *for-each* cyklus prochází seznam následujícím způsobem – nejprve se zeptá na existenci dalšího prvku voláním *hasNext* a pokud existuje, tak na něj přejde. V okamžiku prvního volání se tedy musí iterátor nacházet na pozici -1, což simulujeme právě zmíněným příznakem.

Metoda *hasNext* vrátí *true*, pokud existuje další uzel spojového seznamu, *false* v opačném případě. *Next* přejde na další uzel. Metoda *remove* pouze vyhodí výjimku (v kontraktu – textovém popisu metody rozhraní – je napsáno, že je tato fukcionalita volitelná).

18.22.1. Spojový seznam s užitím generik a iterátoru

```
    /**
     * Jednosmerne zretezeny spojovy seznam
     * @author
     * @param <ENTITY> typovy parametr specifikujici ukladany typ
     */
    public class MyLinkedList<ENTITY> implements ListIface<ENTITY> {

        private Node first;
        private Node last;
        private int size;

        /**
         * Konstruktor spojoveho seznamu
         */
        public MyLinkedList() {
            this.size = 0;
        }

        /**
         * Vlozi na konec seznamu prvek
```

```
     * @param i prvek k vlozeni
     */
    @Override
    public void add(ENTITY i) {
        Node n = new Node(i);
        if (size == 0) {
            this.first = n;
            this.last = n;
        } else {
            this.last.next = n;
            this.last = n;
        }
        size++;
    }

    /**
     * Vrati prvek na indexu i
     * @param i index prvku
     * @throws IndexOutOfBoundsException pokud je i vyssi nebo rovna
delce seznamu
     * @throws IllegalArgumentException pokud je i zaporne
     * @return prvek na indexu i
     */
    @Override
    public ENTITY get(int i) {
        if (i >= size) {
            throw new IndexOutOfBoundsException("Mimo meze index:" +
i + ", size:" + this.size);
        }
        if (i < 0) {
            throw new IllegalArgumentException("Index mensi nez 0");
        }
        Node curr = first;
        for (int j = 0; j < i; j++) {
            curr = curr.next;
        }
        return curr.value;
    }

    /**
     * Smaze prvek na indexu i
     * @param i index mazaneho prvku
     * @throws IndexOutOfBoundsException pokud je i vyssi nebo rovna
delce seznamu
     * @throws IllegalArgumentException pokud je i zaporne
     */
    @Override
    public void remove(int i) {
        if (i >= size) {
            throw new IndexOutOfBoundsException("Mimo meze index:" +
i + ", size:" + this.size);
        }
        if (i < 0) {
            throw new IllegalArgumentException("Index mensi nez 0");
        }
        if (i == 0) {
            first = first.next;
        } else {
            Node curr = first;
            for (int j = 0; j < i - 1; j++) { //najdeme predchozi
                curr = curr.next;
```

```
                }
                curr.next = curr.next.next; //a mazany prvek vynechame
                if (i == size - 1) { //pokud mazeme posledni
                    last = curr;
                }
            }
            size--;
        }

        /**
         * Dotaz na delku seznamu
         * @return delka seznamu
         */
        @Override
        public int size() {
            return this.size;
        }

        /**
         * Klasicka toString metoda, vraci textovou reprezentaci objektu
         * @return textova reprezentace objektu
         */
        @Override
        public String toString() {
            StringBuilder builder = new StringBuilder();
            Node curr = first;
            for (int i = 0; i < this.size; i++) {
                builder.append(curr.value);
                builder.append(" ");
                curr = curr.next;
            }
            return builder.toString();
        }

        @Override
        public Iterator iterator() {
            return new LinkedListIterator();
        }

        /**
         * Jednoduchy iterator pro spojovy seznam
         * Neresi situaci, kdy je iteratoru pod rukou pozmena kolekce
(tj. neni fail fast)
         */
         private class LinkedListIterator implements Iterator<ENTITY> {

            private Node currNode;
            private boolean start;

            public LinkedListIterator() {
                this.currNode = null;
                this.start = true;
            }

            @Override
            public boolean hasNext() {
                if (start) {
                    return first != null;
                }
                return currNode.next != null;
            }
```

```
            @Override
            public ENTITY next() {
                if (start) {
                    start = false;
                    currNode = first;
                } else {
                    currNode = currNode.next;
                }
                return currNode.value;
            }

            @Override
            public void remove() {
                throw new UnsupportedOperationException("Remove is not
supported");
            }
        }

        /**
         * Vnitrni trida reprezentujici jeden uzel spojoveho seznamu
         */
        private class Node {

            private ENTITY value;
            private Node next;

            private Node(ENTITY value) {
                this.value = value;
            }
        }
    }

    /**
     * Rozhrani, jez definuje operace, ktere musi splnit kazda implemen-
tace seznamu
     * @author
     */
    public interface ListIface<ENTITY> extends Iterable<ENTITY> {

        /**
         * Vlozi na konec seznamu prvek
         * @param i prvek k vlozeni
         */
        public void add(ENTITY i);

        /**
         * Vrati prvek na indexu i
         * @param i index prvku
         * @return prvek na indexu i
         */
        public ENTITY get(int i);

        /**
         * Smaze prvek na indexu i
         * @param i index mazaneho prvku
         */
        public void remove(int i);

        /**
```

```
     * Dotaz na delku seznamu
     * @return delka seznamu
     */
    public int size();
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    ListIface<Integer> l = new MyLinkedList<Integer>();
    l.add(0);
    l.add(10);
    l.add(20);
    l.add(30);
    l.add(40);

    System.out.println(l); //toString

    for(Integer i : l){
        System.out.print(i + " ");
    }
}
```

Výstupem je:

```
0 10 20 30 40
0 10 20 30 40
```

18.23. Proudy a serializace

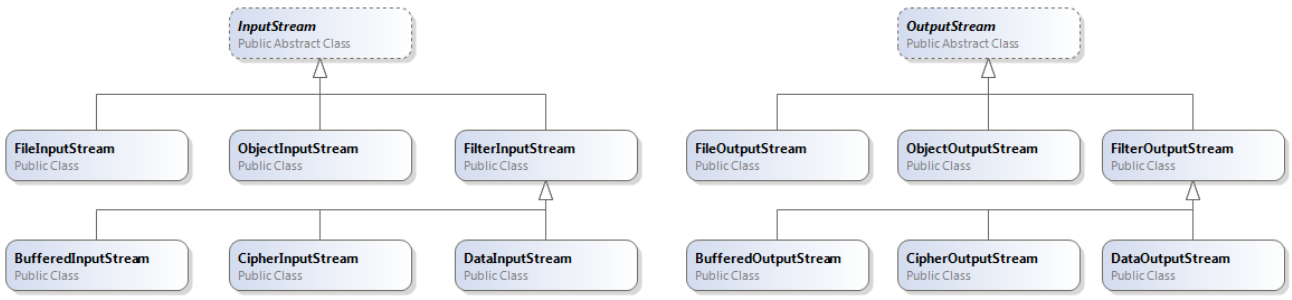
Doposud naše aplikace pracovaly vesměs s daty, která jsme jim interně zadali. V dnešním dílu si představíme datové proudy (streamy), pomocí kterých může naše aplikace komunikovat se svým okolím.

Proudy pro nás ale nejsou žádnou novinkou, již jsme se s nimi mnohokrát setkali. Pro výpis na konzoli jsme volali metody proudu *System.out*, a když jsme získávali číslo ze standardního vstupu, tak jsme jej četli z proudu *System.in*.

Bylo by ovšem mylné se domnívat, že vstupem a výstupem programů je pouze klávesnice a konzole. Proudy mohou stejně dobře pracovat s výstupními zařízeními (tiskárnou), obrázky, soubory a mnoha dalšími sériovými zdroji/cíly dat.

18.23.1. Typy proudů

Proudy můžeme dělit dle jejich směru na vstupní (přenášejí data do aplikace) a na výstupní (přenášejí data z aplikace). Proudy také rozlišujeme na binární a znakové. Jak již názvy napovídají, tak zatímco binární proudy využijeme pro libovolná binární data (tj. data vkládáme je po bajtech), tak znakové proudy jsou určeny pouze pro text (znaky).



18.23.2. Decorator

S proudy je v Javě neodmyslitelně spojen návrhový vzor *decorator* (*dekorátor*). Jak jsme si již naznačili, existuje velké množství proudů pro různé situace a zařízení. Tyto proudy ale neobsahují veškerou možnou funkcionalitu, primárně protože kombinace jednotlivých vlastností nemusí být kompatibilní, a ve většině případů s sebou nese i určitou režii.

Pokud bychom zároveň vytvářeli zvláštní třídu pro každý různě obohacený proud, tak bychom jich brzy měli tisíce. Příkladem by pak mohla být třída *Proud pro zápis do souboru s vyrovnávací pamětí, bez kontrolního součtu, šifrovaný, bez hashe (a pětatřiceti dalšími příznaky)*.

Základní myšlenka vzoru decorator je velmi jednoduchá – ať je proud obohacený jakýmkoliv způsobem, tak musí mít pořád stejné rozhraní.

18.23.2.1. Příklad

Pro případ vstupního (de)šifrujícího souborového proudu proto stačí vytvořit nejprve proud pro přístup k souboru (základní, nešifrovaný) a ten vložit jako zdroj do (de)šifrujícího proudu. Veškerá volání dešifrujícho proudu nejprve zpracujeme a poté delegujeme patřičným metodám obaleného proudu.

Pokud tedy budeme chtít použít metodu, která se šifrování nijak netýká – například uzavření proudu – tak (de)šifrující proud volání pouze přepošle. Pokud budeme chtít přečíst nějaká data, tak je (de)šifrující proud přečte z obaleného proudu a nám navrátí výstup interního dešifrovacího procesu.

```
	//Z tohoho souboru budeme cist
	InputStream is = new FileInputStream("/tmp/file.txt");
	//Ale soubor je sifrovan, predame (de)sifrovacimu proudu instanci
puvodniho vstupniho proudu
	//NullCipher nedela nic :-), je to identita
	CipherInputStream cis = new CipherInputStream(is, new NullCipher());
	//precteme jeden byte desifrovanych dat
	int i = cis.read();

	//uzavreme proud (uvolnime zdroje systemu)
	cis.close();
```

18.23.3. Binární proudy

Binární proudy umožňují přenést libovolná data. Vstupní proudy dědí z třídy *InputStream* (dokumentace), výstupní z *OutputStream* (dokumentace). Základní operací definovanou v *InputStream*u je metoda *read*, pomocí které můžeme z proudu přečíst jeden bajt. Analogicky výstupní proud definuje metodu *write*.

Čtení a zápis po jednotlivých bajtech je velmi pomalé, zvláště pokud uvážíme, že na druhé straně proudu může být disk – a každý dotaz může velmi snadno znamenat nutnost nového vystavení hlaviček.

18.23.3.1. BufferedStream

Třídy *BufferedInputStream* (dokumentace) a *BufferedOutputStream* (dokumentace) za nás tento nedostatek řeší. Tyto proudy obsahují pole, které slouží jako vyrovnávací paměť. V případě čtení z disku bufferovaný proud načte celý blok dat a uloží jej do pole, ze kterého data dále posílá našemu programu. V okamžiku, kdy se pole vyprázdní, učiní dotaz na další blok. Tímto způsobem dochází k eliminaci velkého množství zbytečných a drahých volání.

Obdobně pro zápis na disk použijeme *BufferedOutputStream*. Zde se ovšem můžeme dostat do situace, kdy zapíšeme do bufferu méně dat, než je potřebné k jeho vyprázdnění (zápisu do obaleného proudu). V tomto případě můžeme zavolat metodu *flush*, která vyprázdní

všechny buffery (tj. zajistí okamžitý zápis všech dat). Před uzavíráním proudu – uzavření dealokuje veškeré zdroje nutné pro udržení spojení – není nutné *flush* použít, metoda *close* to udělá sama.

18.23.3.2. Serializace

Velmi zajímavou možností využití proudů je *serializace* (ukládání objektů do formátu, který může být uložen nebo poslán po síti). K serializování objektů můžeme použít standardní mechanismus využívající třídy *ObjectOutputStream* (*dokumentace*) a rozhraní *Serializable* (*dokumentace*).

Pro serializaci objektu ovšem musí být splněno několik podmínek:

- Třída objektu implementuje rozhraní *Serializable* (nemá metody, slouží pouze jako marker).
- Každá nadtřída, která neimplementuje *Serializable* musí mít veřejný bezparametrický konstruktor. Serialiyovaná třída musí zajistit serializaci datových složek všech nadtříd, které nejsou serializovatelné.
- Všechny datové složky objektu musí být buď primitivního typu, serializovatelné nebo přechodné – klíčové slovo *transient* (značí, že daná položka nebude serializována).

První podmínku jsme schopni zaručit téměř vždy – stačí pouze objekt oddědit a implementovat rozhraní (výjimkou jsou konečné třídy (mají v hlavičce modifikátor *final*)). Zbylé výjimky můžeme obejít implementací rozhraní *Externalizable* (*dokumentace*), pomocí kterého si nadefinujeme vlastní serializační protokol.

Pokud chceme při serializaci a deserializaci pouze provést nějaké dodatečné akce, ale vystačíme si s automatickým mechanismem, tak musíme na serializovatelném objektu pouze implementovat metody *readObject* a *writeObject* (musí být soukromé).

```
/**
 * Demonstrace serializacnich mechanismu
 * @author
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws FileNotFoundExcep-
tion, IOException, ClassNotFoundException {
        ObjectOutputStream s = new ObjectOutputStream(new FileOutput-
Stream("c:/temp/object.txt"));
        A a = new A();
        a.setBar("bar");
        a.setFoo("foo");
        s.writeObject(a); //zapiseme a


        B b = new B();
        b.setFoo("fooB");
        b.setBar("barB");
        s.writeObject(b); //zapiseme b - vypise "Serializating"

        C c = new C();
        c.setFoo("fooC");
        c.setBar("barC");

        s.writeObject(c);

        s.close(); //zavreme vystupni stream
```

```
            ObjectInputStream inputStream = new ObjectInputStream(new Fi-
leInputStream("c:/temp/object.txt"));
            A readA = (A)inputStream.readObject();
            System.out.println(readA.getFoo()); //foo
            System.out.println(readA.getBar()); //null

            B readB = (B) inputStream.readObject(); //vypise "Deseriali-
zating"
            System.out.println(readB.getFoo()); //fooB
            System.out.println(readB.getBar()); //null

            C readC = (C) inputStream.readObject();
            System.out.println(readC.getFoo()); //fooC
            System.out.println(readC.getBar()); //barC

            inputStream.close(); //zavreme vstupni stream

        }
    }

    /**
     * Demonstrace automaticke serializace
     * @author Pavel Micka
     */
    class A implements Serializable {

        protected String foo;
        /**
         *
         */
        protected transient String bar;

        /**
         * @return the foo
         */
        public String getFoo() {
            return foo;
        }

        /**
         * @param foo the foo to set
         */
        public void setFoo(String foo) {
            this.foo = foo;
        }

        /**
         * @return the bar
         */
        public String getBar() {
            return bar;
        }

        /**
         * @param bar the bar to set
         */
        public void setBar(String bar) {
            this.bar = bar;
        }
    }
```

```
      /**
       * Demonstrace explicitniho pouziti metod readObject a writeObject
       * @author Pavel Micka
       */
      class B extends A {

          private void writeObject(ObjectOutputStream out) throws IOExcep-
tion {
              //zapise automaticky, co lze automaticky zapsat
              out.defaultWriteObject();
              System.out.println("Serializating");
          }

          private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException {
              // our "pseudo-constructor"
              in.defaultReadObject();
              System.out.println("Deserializating");

          }
      }

      /**
       * Pouziti rozhrani Externalizable
       * @author Pavel Micka
       */
      class C extends A implements Externalizable {

          public C() {
          }


          public void writeExternal(ObjectOutput out) throws IOException {
              out.writeObject(this.foo);
              out.writeObject(this.bar);
          }

          public void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException {
              this.foo = (String) in.readObject();
              this.bar = (String) in.readObject();
          }
      }
```

18.23.3.3. Metoda reset

Kdybychom chtěli do výstupního proudu ukládat jeden a ten samý objekt vícekrát (byť s pozměněnými datovými složkami), tak by nám k tomu prosté opakované volání *writeObject* nestačilo. Výstupní proud si totiž interně pamatuje, že daný objekt již byl zapsán a podruhé jej neuloží. Tuto vlastnost ale můžeme obejít vyčištěním interní paměti proudu pomocí volání metody *reset*.

18.23.3.4. Serial version UID

Java si v rámci serializovatelných tříd udržuje konstantu *serialVersionUID*, která udává verzi objektu. Pokud změníme jeho definici, tak se změní i zmíněná konstanta a původní objekt již nepůjde deserializovat.

Existují ale i operace, které uložený objekt nezneplatní – například přidání členské proměnné, její odmazání, přidání či odebrání metody objektu. V tento okamžik se hodí mít u původní i zastaralé (uložené) entity stejné, explicitně zadané, *serialVersionUID*, aby deserializace proběhla bez problémů (přidaná/odebraná proměnná byla ignorována).

Mezi změny, které nejsou kompatibilní, patří odebrání rozhraní *Serializable* a obecně jakákoliv změna hierarchie předků dané třídy.

```
class A implements Serializable {
    private static final long serialVersionUID = 10275539472837495L;
    .
    .
    .
}
```

18.23.4. Znakové proudy

Znakové proudy fungují stejným způsobem jako ty binární, pouze operují s textem (a jsou odvozeny od tříd *Reader* (dokumentace) a *Writer* (dokumentace)).

Poměrně důležitou poznámkou je, že Java interně uchovává řetězce ve znakové sadě *Unicode*. Z toho plyne, že při každém zápisu a čtení ze znakového proudu dochází k překódování daného řetězce (znaků).

18.23.4.1. Příklad

Abychom si demonstrovali použití znakových proudů, tak vytvoříme řádkový nástroj, který v zadaném textovém souboru spočítá počet řádků, slov a písmen.

```
/**
 * Utilita, ktera spocita pocet znaku, slov a radku v zadanem souboru
 * @author
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws FileNotFoundExcep-
tion, UnsupportedEncodingException, IOException {
        //dva parametry prikazove radky -> 1) cesta k souboru 2) ko-
dovani souboru
        if(args.length != 2){
            System.out.println("Usage: program file_name file_en-
coding");
            System.exit(1); //ukonci beh programu (virtualniho
stroje) s kodem 1, kod jiny nez 0 == chyba
        }
        String filePath = args[0];
        String fileEncoding = args[1];

        int wordCount = 0;
        int lineCount = 0;
        int characterCount = 0;

        //vytvorime novou bufferovanou ctecku s danym kodovanim z Fi-
leInputStreamu
        BufferedReader reader = new BufferedReader(new InputStreamRe-
ader(new FileInputStream(filePath), fileEncoding));
        String line = null;
        while((line = reader.readLine()) != null){ //dokud mame k
dispozici dalsi radek
            characterCount += line.length();
```

```
                lineCount++;

                //tokenizer rozlozi text na jednolive symboly (tokeny)
dle uvedenych oddelovacu (delimiter)
                StringTokenizer tokenizer = new StringTokenizer(line, "
\u0009"); //oddelovace jsou mezera a \u0009 == tabulator
                wordCount += tokenizer.countTokens();
            }

            reader.close(); //i Readery (a Writery) musime zavrit

            System.out.println("Pocet znaku: " + characterCount);
            System.out.println("Pocet slov: " + wordCount);
            System.out.println("Pocet radku: " + lineCount);
        }
    }
```

Abychom mohli program otestovat z vývojového prostředí *Netbeans*, musíme vybrat kontextovém menu projektu volbu *Properties*, zvolit položku *Run* a vyplnit pole *Arguments* (tj. umístit zde cestu ke zvolenému textovému souboru a název jeho kódování).

Také si můžeme zkusit program spustit z příkazové řádky. Nejprve vstupíme do složky projektu a podsložky *build/classes*. Zde program spustíme příkazem *"java balicek_tridy.jmeno_tridy parametry_prikazove_radky"*.

18.23.4.1.1. Zjištění kódování textového souboru

Pokud nevíte, jakým způsobem zjistit kódování textového souboru, tak si stáhněte textový editor *PSPad* (aktuálně ve verzi 4.5.4). Otevřete v něm zvolený soubor a podívejte se na dolní lištu (zde uvidíte napsáno například *Kódování UTF-8*). Pokud byste chtěli znakovou sadu změnit, tak můžete zvolit jinou v menu *Formát*.

18.24. Výčtový typ

Výčtový typ *enum*, který byl přidán do Javy ve verzi 5, nám umožňuje vytvářet vlastní datové typy, které mohou nabývat pouze určitého omezeného množství hodnot. Každé z těchto hodnot poté odpovídá právě jedna instance výčtu.

Příkladem mohou být měsíce v roce (*leden, únor...*), dny v týdnu (*pondělí, úterý...*) nebo světové strany (*sever, jih...*).

18.24.1. Výhody výčtového typu

Dokud v Javě nebyl výčtový typ implementován, tak byl programátory emulovám pomocí rozhraní obsahujícího odpovídající množinu (obvykle celočíselných) konstant. Tento postup měl několik nedostatků.

Prvním a nejvýznamnějším nedostatkem byla typová bezpečnost. Pokud chtěl programátor předat metodě konstantu z takto emulovaného výčtu, tak musel mít v hlavičce metody uveden celý její rozsah (tj. metoda přijímala například typ *int*). Takto definovaná metoda ovšem nebyla jakkoliv odolná vůči volání s nesprávnou/nedefinovanou hodnotou parametru (například při změně implementace).

Druhým nedostatkem je samotná neobjektovost tohoto přístupu. Konstanta definovaná v rámci rozhraní s sebou nemůže nést žádné chování, které tudíž musí být implementováno někde jinde, nejčastěji v metodě, která tuto konstantu přijímá ve svém parametru.

Další nevýhodou je nízká informační hodnota předané konstanty, která se projeví především při logování a debugování programu. Představme s situaci, kdy předáme konstantu *API*

vysoké úrovně, které ji postupně posílá do nižších vrstev aplikace. Pokud dojde k chybě v některém z těchto volání, tak se patrně pouze dozvíme, že došel chybný parametr "5", ale nikoliv, co vlastně znamená (pro převod bychom museli využít nějakého číselníku).

Výčtový typ *enum* všecny tyto nedostatky zcela eliminuje.

18.24.2. Deklarace

Deklarace výčtového typu je velmi podobná deklaraci třídy. Pouze v hlavičce nepoužijeme klíčové slovo *class*, ale slovo *enum*. Dále v rámci těla musíme nejprve vyjmenovat všechny možné hodnoty (instance), kterých může výčet nabývat. Ve zbytku těla smíme použít stejné konstrukty jako u libovolné třídy.

18.24.3. Omezení

Jak jsme si již uvedli, *enum* může nabývat pouze omezeného množství hodnot (reprezentovaných vyjmenovanými instancemi). Z toho plyne, že konstruktor *enum*u je vždy soukromý a *enum* jako takový je konečný (*final*) – což znamená, že z něj nelze odvozovat podtypy.

Jelikož jsou výčty serializovatelné, tak Java zaručuje, že v případě deserializace objektu výčtového typu nevznikne nová instance.

Z těchto omezení pro nás plyne také jedna úleva. Pro porovnávání hodnot výčtového typu můžeme bez obav používat operátor == (není třeba volat metodu *equals*).

18.24.4. Společný předek – třída Enum

Společným předkem všech výčtových typů je třída *Enum* (*dokumentace*), která definuje základní společné metody a implementuje rozhraní *Serializable* a *Comparable*.

Ze společných operací uveďme metody *values* a *ordidal*. Metoda *values* vrací pole všech instancí výčtu a je zajímavá tím, že ji přidává až překladač během kompilace. Metodu *ordinal* vrací pořadí, v němž byla daná instance v rámci výčtu nadefinována (indexováno od nuly).

18.24.4.1. Příklad

Vytvořme si výčtový typ pro měsíce v roce, který obsahovat metodu *getRandom*, která vrátí náhodný měsíc.

```
/**
 * Vyctovy typ pro mesice v roce
 * @author Pavel Micka
 */
public enum Month {
    JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEM-
BER, OCTOBER, NOVEMBER, DECEMBER;
    /**
     * Vrati nahodny mesic
     * @return nahodny mesic
     */
    public static Month getRandom(){
        Random r = new Random(); // pseudonahodny generator cisel
        int count = Month.values().length;
        return Month.values()[r.nextInt(count)];
    }
}
```

Instance jednotlivých měsíců nyní získáme následovně:

```
Month m1 = Month.JANUARY; // Ziskejme instanci odpovidajici lednu
Month m2 = Month.getRandom(); // Ziskejme nahodnou instanci
```

Rozšiřme nyní náš příklad a definujme konstruktor, jenž předá jednotlivým měsícům pranostiky, které se k nim vztahují.

```
/**
 * Vyctovy typ pro mesice v roce
 * @author
 */
public enum Month {
    JANUARY("Je-li teplo v lednu, sahá bída ke dnu."),
    FEBRUARY("Únor bílý, pole sílí."),
    MARCH("Březen - za kamna vlezem."),
    APRIL("Duben - ještě tam budem."),
    MAY("Máj - vyženem kozy v háj."),
    JUNE("Když kvete chrpa, za čtyři neděle chop se srpa."),
    JULY("Co červenec neuvaří - srpen nedopeče."),
    AUGUST("Když srpen z počátku hřeje, ledový vítr v zimě dlouho
věje."),
    SEPTEMBER("Bouřka v září - sníh v prosinci."),
    OCTOBER("Sněží-li v říjnu, bude měkká zima."),
    NOVEMBER("Když začátkem listopadu sněží, pak mívá sníh výšku
věží."),
    DECEMBER("Chodí-li Kateřina po ledě, chodí Štěpán po blátě.");

    private String weatherLore;

    private Month(String weatherLore) {
        this.weatherLore = weatherLore;
    }

    /**
     * Vrati nahodny mesic
     * @return nahodny mesic
     */
    public static Month getRandom(){
        Random r = new Random(); // pseudonahodny generator cisel
        int count = Month.values().length;
        return Month.values()[r.nextInt(count)];
    }

    /**
     * Vrati pranostiku na dany mesic
     * @return
     */
    public String getWeatherLore() {
        return weatherLore;
    }
}
```

Významnou vlastností výčtu je, že jej můžeme použít jako proměnnou v přepínači (switch).

```
Month m = Month.getRandom();

switch(m){
    case JUNE :
        System.out.println("Hura na prazdniny!");
        break;
    case SEPTEMBER:
        System.out.println("Skola je tu. :-(");
        break;
    case DECEMBER:
        System.out.println("Budou Vanoce!");
        break;
    default:
```

```
                System.out.println("V tomto mesici se nic zvlastniho nedeje.
:-)");
                System.out.println(m.name()); //vypiseme nazev mesice (nazev
instance)
                break;
        }
```

18.25. Vlákna

Programy, které jsme dosud tvořili, byly striktně sekvenční – žádné dvě jejich části nebyly vykonávány ve stejném čase. Nyní, jak vytvářet programy tak, aby některé úseky kódu probíhaly paralelně. Tohoto chování pak využijeme v grafickém uživatelském rozhraní, kdy budeme vyžadovat, aby se v jednu chvíli zároveň překreslovalo rozhraní a probíhal výpočet (složitého) úkolu.

Jelikož je však paralelní programování poměrně náročné na schopnosti vývojáře a programy se velmi špatně debugují, tak si ukážeme pouze jeho základní vlastnosti, možnosti, pasti a propasti.

18.25.1. Co je to vlákno?

Vlákno můžeme chápat jako abstrakci nad stavem programu – pozice zpracovávané instrukce v kódu a předané parametry jednotlivých volání metod. Každý běžící program obsahuje minimálně jedno vlákno, které začíná svou práci u javovských desktopových aplikací v metodě *main(String[] args)* a přestává existovat v okamžik, kdy tuto metodu opět opustí (tak dosud pracovaly všechny naše programy).

Je ale důležité říci, že vlákna můžeme také vytvářet a dávat jim ke zpracování určité části kódu. Pokud má náš počítač více jader procesorů, než je aktuálně spuštěných vláken, tak vlákna budou vykonávána souběžně. Pokud je volných jader méně než vláken, tak mezi nimi byde procesor ve velmi krátkých intervalech přepínat, čímž vytvoří zdání souběžnosti.

Program terminuje v okamžiku, kdy jej opustí poslední vlákno, které není označeno jako *daemon*. Takto označené vlákno je bráno jako pomocné a při terminaci programu na něj není brán ohled.

18.25.2. Procesy

Na podobném principu jako vlákna jsou založeny i procesy. Tyto jsou ovšem abstrakcí nad celým běžícím programem (proces může obsahovat mnoho vláken, avšak vždy minimálně jedno), jednotlivé procesy mají vlastní paměťový kontext (nikoliv pouze zásobník volání) a vlastní zámky nad soubory.

18.25.3. Vlákna vs. procesy

Z výše uvedených rozdílů plyne, že výměna vlákna na procesoru, stejně jako jeho vytvoření nebo zrušení, je oproti stejné akci s procesem výrazně levnější operací (není třeba měnit paměťový kontext). Dále je komunikace vláken výrazně snazší než komunikace procesů, protože stačí využít sdílených struktur (objektů a jejich fieldů), zatímco u procesů by bylo zapotřebí využít volání vzdálených metod, sdílet data skrze soubor namapovaný do paměti atp. (ani jedna z těchto technik se nedá označit za přívětivou).

Nevýhodou vláken pak je zejména obtížné ošetření jejich korektního souběhu – jedno vlákno nesmí druhému modifikovat data *pod rukama*, jelikož by tak mohlo dojít k ovlivnění výpočtu a navrácení zcela nesmyslného výsledku.

18.25.4. Vlákna a procesy v Javě

Z pohledu javovského programátora nás však tyto rozdíly moc trápit nemusí, protože budeme pracovat vždy pouze s vlákny. Procesy lze sice také využívat, ale zde se jedná o velmi speciální případy – například o volání externích programů.

18.25.4.1. Jednoduchá aplikace

```
/**
 * Demonstrace jednovlaknove aplikace
 * @author
 */
public class Main {
    /**
     * Do teto metody vstoupi hlavni vlakno aplikace
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();
        list.add(1);
        list.add(2);
        printList(list);
    }

    /**
     * Vypise obsah predaneho seznamu
     * @param l seznam
     */
    public static void printList(List l){
        for(Object o : l){
            System.out.println(o);
        }
    }
}
```

Ukažme si nejprve průchod vlákna skrz jednoduchou aplikaci, která pouze přidá do seznamu dvě čísla a posléze obsah seznamu vypíše.

Vlákno vstoupí do hlavní metody main na řádce 10, vytvoří seznam na haldě (společný paměťový prostor, ve kterém jsou vytvářeny všechny objekty), odkaz na něj je však uložen na zásobník volání (proměnná list je lokální). Nakonec je seznam předán metodě printList, která seznam vypíše. Po návratu z volání vlákno program na řádce 15 opustí, čímž jej terminuje.

V případě, že bychom volali metodu main z více vláken, tak by nemohlo dojít k jejich interakci, jelikož jsou reference na použité proměnné uložené na zásobníku, který má každé vlákno samo pro sebe. Stejně tak je při volání metody printList odkaz na předávaný seznam uložen na zásobníku daného vlákna. Jediným problematickým místem je samotný výpis, kde by mohlo hrozit, že budou vlákna interagovat při zápisu do výstupního proudu (uvnitř volání metody System.out.println). Zde je ale situace ošetřena již uvnitř zmíněného volání – jak ošetřit souběžnou modifikaci sdílené struktury si ukážeme později.

18.25.4.2. Vícevláknová aplikace

Nyní si ukažme, jak by ten vypadal obdobný kód vícevláknové aplikace. Pro tvorbu nových vláken využijeme třídu Thread (*dokumentace*), respektive její metodu *run*. Obsah metody *run* bude při spušnění nového vlákna (zavolání jeho metody *start*) vykonán asynchronně na volajícím vláknu.

```
/**
 * Demonstrace dvouvlaknove aplikace
 * @author
 */
public class Main {
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Thread t1 = new MyThread();
        Thread t2 = new MyThread();
        t1.start();
        t2.start();
    }
}

class MyThread extends Thread{

    @Override
    public void run() {
        List<Integer> list = new ArrayList<Integer>();

        for(int i = 0; i < 1000; i++){
            list.add(i);
        }

        printList(list);
    }

    /**
     * Vypise obsah predaneho seznamu
     * @param l seznam
     */
    public static void printList(List l) {
        for (Object o : l) {
            System.out.println(o);
        }
    }
}
```

Výstup je:

```
0
0
1
1
2
2
3
3
4
4
5
6
7
5
8
6
9
10
11
7
8
12
9
13
```

10
14
11
15
12
16
13
17
14
18
15
19
16
20
17
21
18
19
22
20
23
21
24
22
25
23
26
24
25
27
28
26
29
27
30
28
31
29
32
30
33
31
32
33
34
34
35
35
36
36
37
37
38
38
39
39
40
40
41
42
41

```
43
42
44
43
45
46
44
47
45
48
46
49
47
48
49
```

Ve výpisu na konzoli si všimněme, že jsou seznamy pomíchány, což je způsobeno tím, že je vnitřně chráněno pouze samotné volání *System.out.println*, nikoliv celé výpisy. Pokud by však nebylo chráněno ani samotné volání, tak bychom na výpis dostali *rozsypaný čaj*, jelikož by došlo k situaci, kdy by jedno vlákno zapsalo pouze část výstupního řetězce, a procesor by přepnul na druhé vlákno (jež by začalo vypisovat svůj výstup).

18.25.5. Synchronizace

V minulém případě jsme využívali toho, že metody byly tzv. *thread-safe* – vláknově bezpečné, jelikož žádným způsobem nemodifikovaly sdílený stav. Ukažme si, jak vypadá vláknově nebezpečný kód – kód, který modifikuje stav neatomickými operacemi (atomická operace je taková operace, která proběhne na procesoru v rámci jediného taktu – je nedělitelná).

```
/**
 * Demonstrace chyby v praci nad spolecnym zdrojem
 * Ctenari, se omlouvam za antiobjektovost tohoto prikladu :-)
 * @author
 */
public class NonAtomicExample {

    public volatile static int i = 0; //volatile == necachuj na pro-
cesoru

    public static void main(String[] args) throws InterruptedExcep-
tion {
        Thread t1 = new MyThread();
        Thread t2 = new MyThread();

        t1.start();
        t2.start();

        Thread.sleep(1000); //na vterinu uspime hlavni vlakno a ne-
chame dve pridana pracovat

        System.out.println(i); //melo by byt 20000
    }
}

class MyThread extends Thread {

    @Override
    public void run() {
        for(int i = 0; i < 10000; i++){
            NonAtomicExample.i++; // NonAtomicExample.i = NonAtomi-
cExample.i + 1 - dve operace: scitani a prirazeni
```

```
            }
        }
    }
```

Výstupem je: `19286`

Tento kousek kódu, inkrementující společnou proměnnou, ukazuje hned dvě pasti, do kterých ochotně spadne téměř každý programátor.

18.25.6. Neatomická inkrementace

První problém spočívá v inkrementaci společné proměnné, jelikož ji téměř každý považuje za jednu operaci. Ve skutečnosti se ale jedná o operace dvě: součet a přiřazení. Proto může dojít k situaci, kdy vlákno *A* přečte hodnotu proměnné (1000) a přičte k ní 1, v ten okamžik je přerušeno ve prospěch vlákna *B*, které opět přečte hodnotu (1000) a přičte k ní 1 a je přerušeno ve prospěch původního vlákna *A*. Nakonec obě vlákna postupně zapíší hodnotu 1001, ačkoliv došlo ke dvěma inkrementacím (má být 1002). Tato situace je ještě velmi příznivá, protože nikde není psáno, za jak dlouho budou vlákna vystřídána, stejně tak by mohlo dojít k situaci, že první vlákno přečte hodnotu (0) a je vystřídáno, druhé vlákno 10000x inkrementuje a je vystřídáno a vlákno původní přepíše hodnotu 10000 číslem 1.

Právě kvůli této chybě je výstupem programu 19286 místo 20000.

18.25.6.1. Volatile

Druhou věcí, na kterou je třeba myslet, je cacheování na procesoru. Pokud pracujeme stále s jednou proměnnou, tak je tato proměnná z důvodu optimalizace přiřazena do vyrovnávací paměti (jádra) procesoru (práce s ní je tak řádově rychlejší, než kdyby byla v RAM). Vyrovnávací paměť procesoru ovšem nemusí být sdílená mezi jednotlivými jádry. Proto pokud vlákno *A* na prvním jádře pracuje se svojí lokální kopií, tak o případných modifikacích (v našem případě inkrementacích) není nijak informováno vlákno *B* na druhém jádře pracující nad další lokální kopií proměnné. Obě jádra tak pracují nezávisle s neaktuálními daty a výsledek bude samozřejmě nekorektní.

Abychom tomuto předešli, tak u sdílených proměnných, jejichž hodnotu modifikujeme, uvádíme klíčové slovo *volatile*, které zajistí, že proměnná nebude nikdy cacheována (jádro si vždy vyžádá novou hodnotu a zápis provádí do RAM).

18.25.6.2. Synchronized

Abychom napravili výše zmíněnou chybu neatomické inkrementace, tak musíme zabránit tomu, aby se v kritické oblasti kódu v jeden okamžik vyskytovalo více než jedno vlákno. K tomuto využijeme synchronizovanou sekci, což je blok, před nímž je uvedeno klíčové slovo *synchronized* a v závorce objekt zámku.

Zámek je libovolný javovský objekt, který slouží jako pešek – může ho vlastnit pouze jedno vlákno, a právě toto vlákno smí vstoupit do kritické sekce a vykonat zde obsažené instrukce, zbylá vlákna čekají na okamžik, kdy je zámek opět dostupný. Po opuštění kritické sekce vlákno zámek automaticky uvolní.

```
    /**
     * Spravna implementace inkrementace spolecne promenne
     * @author
     */
    public class NonAtomicExample {

        public volatile static int i = 0; //volatile == necachuj na pro-
cesoru

        public static void main(String[] args) throws InterruptedExcep-
tion {
```

```
            Thread t1 = new MyThread();
            Thread t2 = new MyThread();

            t1.start();
            t2.start();

            Thread.sleep(1000); //na vterinu uspime hlavni vlakno a ne-
chame dve pridana pracovat

            System.out.println(i); //melo by byt 20000
        }
    }

    class MyThread extends Thread {

        @Override
        public void run() {
            for(int i = 0; i < 10000; i++){
                synchronized(NonAtomicExample.class){ //objekt tridy No-
nAtomicExample slouzi jako zamek
                    NonAtomicExample.i++; // NonAtomicExample.i = NonAto-
micExample.i + 1 - dve operace: scitani a prirazeni
                }
            }
        }
    }
```

Výstup: `20000`

18.25.7. Synchronizované metody

Kromě bloků lze také synchronizovat celé metody, stačí uvést klíčové slovo *synchronized* v jejich hlavičce. Je však třeba dávat pozor na to, co slouží jako zámek. V případě statických metod se jedná o objekt obalující třídy, u nestatických metod se jedná o instanci, nad níž je metoda volána.

```
    /**
     * Demonstrace dvouvlaknove aplikace nad sdilenym seznamem
     * @author
     */
    public class Main {

        /**
         * @param args the command line arguments
         */
        private List list = new ArrayList();

        public static void main(String[] args) throws InterruptedExcep-
tion {
            //promenne vyuzivane ve vnitrnich tridach musi byt konecne
(final)
            final Main m = new Main(); //vytvorime instanci objektu Main
(kvuli seznamu list)

            Thread t1 = new Thread() {

                @Override
                public void run() {
                    for (int i = 0; i < 50; i += 2) {
                        synchronized (m) {
                            m.list.add(i);
                        }
```

```
                    }
                }
            };
            Thread t2 = new Thread() {

                @Override
                public void run() {
                    for (int i = 1; i < 50; i += 2) {
                        synchronized (m) {
                            m.list.add(i);
                        }
                    }
                }
            };

            t1.start();
            t2.start();

            Thread.sleep(1000);
            Collections.sort(m.list);
            m.printList();
        }

        /**
         * Vypise obsah predaneho seznamu
         * Metoda je synchronizovana, u statickych metod je zamkem objekt
obalujici tridy
         * u nestatickych metod obalujici objekt (pozor na tento zasadni
rozdil)
         *
         * V tomto pripade se tedy jedna o synchronizaci nad objektem,
ktery referencujeme
         * v programu jako "m"
         */
        public synchronized void printList() {
            for (Object o : this.list) {
                System.out.println(o);
            }
        }
    }
```

Výstup:

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
```

V tomto příkladě přidáváme ve dvou vláknech čísla do sdíleného seznamu. Jedno vlákno přidává sudá čísla do 50, druhé lichá čísla do 50. Hlavní vlákno obě vlákna odstartuje a počká vteřinu (práce obou vláken by za vteřinu měla být velmi pohodlně hotova), seznam seřadí a vypíše.

Pro synchronizaci bloků používáme instanci třídy *Main*, nad kterou pracujeme. Stejný zámek je implicitně použit při volání instanční metody *printList*.

18.25.8. Deadlock

Další pastí, kterou jsme si dosud nezmínili, je *deadlock* (uváznutí). K deadlocku může dojít u vícevláknového programu, pokud synchronizujeme podle více zdrojů a zároveň nedodržujeme stálé pořadí zámků.

```
synchronized (pliers) {
    synchronized (hammer) {
        //do something
   }
}
.
.
.
synchronized (hammer) {
    synchronized (pliers) {
        //do something
   }
}
```

K demonstraci deadlocku se obvykle využívá problému dvou kovářů (zastupují vlákna). Každý z nich potřebuje ke své práci kleště a kladivo. První kovář bere do ruky nejprve kleště a teprve poté si vezme kladivo, druhý kovář volí opačný postup – nejprve kladivo, pak kleště.

Vše funguje perfektně dokud jsou oba kováři dostatečně rychlí a vždy jeden stihne sebrat oba kusy nářadí. V okamžiku, kdy první kovář sebere kleště a druhý kladivo, dojde k uváznutí. Oba kováři budou do konce věků čekat na uvolnění druhého nástroje (zámek může vlákno uvolnit až po opuštění kritické sekce).

18.25.9. Komunikace vláken

Výše uvedené příklady mají jednu nectnost, kterou jsme přecházeli bez povšimnutí – volání *Thread.sleep(time)* nad hlavním vlákem. V takto jednoduché situaci nám dané řešení zcela postačuje, ale častěji budeme potřebovat, aby vlákno čekalo na nějakou událost (např. *modifikace seznamu je hotova*), čímž se jednak vyhneme zbytečnému čekání, za druhé se pak nedostaneme do situace, kdy zpracování trvalo nečekaně déle a čekající vlákno tak začne zpracovávat nehotová data.

18.25.9.1. Wait, notify, notifyAll

Pro čekání na událost můžeme využít volání *monitor.wait()*, kde *monitor* je objekt zámku (volající vlákno musí být v synchronizované sekci – tj. musí zámek vlastnit). Vlákno, které narazí na tuto operaci se zablokuje (tj. odevzdá veškerý svůj procesorový čas a bude čekat na probuzení) a uvolní držený zámek.

Inverzní operací je volání *monitor.notify()*, které probudí některé z čekajících vláken. Metodu *notify()* musí vždy volat vlákno, které vlastní příslušný zámek. Probuzené vlákno pak čeká, než se mu podaří získat zámek pro sebe a poté pokračuje ve vykonávání zbytku kritické sekce.

Posledním voláním pro komunikaci mezi vlákny je metoda *notifyAll()*, která probudí všechna čekající vlákna.

Pro úplnost si uveďme, že všechny zmíněné metody má každý objekt, protože jsou zděděny od společného předka všech tříd – třídy *Object* (*dokumentace*).

18.25.9.2. Spurious wakeup

Aby situace nebyla příliš jednoduchá, tak na operačních systémech implementujících standard *POSIX* (např. *Linux, Unix*) může zřídka dojít k probuzení vlákna, které nebylo explicitně notifikováno. Toto chování není chyba (*bug*), ale vlastnost (*feature*) – ošetření situace, která toto chování způsobuje, by bylo na některých typech systémů příliš výpočetně drahé.

Kvůli *spurious wakeupu* by mělo vždy probuzené vlákno kontrolovat, že invariant jeho opětovného spuštění je skutečně splněn. Pokud tomu tak není, tak by mělo pokračovat ve spánku.

V reálném světě bychom toto stejně museli vždy kontrolovat, protože nemáme pod kontrolou rozvrhování na procesoru a probuzené vlákno může získat zámek buď téměř okamžitě, za milisekundu nebo také za týden, kdy by zmíněný invariant téměř jistě také neplatil.

```
      /**
       * Demonstrace dvouvlaknove aplikace nad sdilenym seznamem s notifi-
kaci vlaken
       * @author
       */
      public class Main {

          /**
           * @param args the command line arguments
           */
          private List list = new ArrayList();

          public static void main(String[] args) throws InterruptedExcep-
tion {
```

```
            //promenne vyuzivane ve vnitrnich tridach musi byt konecne
(final)
            final Main m = new Main(); //vytvorime instanci objektu Main
(kvuli seznamu list)

            Thread t1 = new Thread() {

                @Override
                public void run() {
                    for (int i = 0; i < 50; i += 2) {
                        synchronized (m) {
                            m.list.add(i);
                        }
                    }
                    if (m.list.size() == 50) {
                        synchronized (m) {
                            m.notify();
                        }
                    }
                }
            };
            Thread t2 = new Thread() {

                @Override
                public void run() {
                    for (int i = 1; i < 50; i += 2) {
                        synchronized (m) {
                            m.list.add(i);
                        }
                    }
                    if (m.list.size() == 50) {
                        synchronized (m) {
                            m.notify();
                        }
                    }
                }
            };

            t1.start();
            t2.start();

            synchronized (m) {
                while (m.list.size() != 50) { //spurious wakeup
                    m.wait(); //wait until notified
                }
            }

            Collections.sort(m.list);
            m.printList();
        }

        /**
         * Vypise obsah predaneho seznamu
         * Metoda je synchronizovana, u statickych metod je zamkem objekt
obalujici tridy
         * u nestatickych metod obalujici objekt (pozor na tento zasadni
rozdil)
         *
         * V tomto pripade se tedy jedna o synchronizaci nad objektem,
ktery referencujeme
         * v programu jako "m"
```

```
     */
    public synchronized void printList() {
        for (Object o : this.list) {
            System.out.println(o);
        }
    }
}
```

Výstupem jsou čísla od 0 do 49.

V kódu vidíme, že obě vlákna po přidání čísel zkontrolují, zda-li není již celá práce hotova (seznam obsahuje 50 položek). Pokud ano, tak probudí hlavní vlákno uspané na řádku 54. Samotné ověření podmínky notifikace nemusí být v synchronizované sekci, protože pokud v daný okamžik není podmínka splněna, tak ji splní později druhé vlákno. Pokud je splněna právě pro jedno vlákno, tak dojde k notifikaci. A pokud je splněna pro obě vlákna (z důvodu přepnutí vláken před podmínkou), tak druhá z notifikací neprobudí žádné čekající vlákno (což nijak nevadí).

Na řádku 53 si pak povšimněme cyklu, který zajistí, že se v případě spurious wakeupu hlavní vlákno znovu uspí.

18.26. Grafické rozhraní

18.26.1. Abstract Window Toolkit

Historicky první standardní knihovnou pro tvorbu grafického rozhraní v Javě bylo AWT (*Abstract Window Toolkit*). AWT je tzv. *heavyweight* knihovna, což znamená, že deleguje vykreslování jednotlivých grafických komponent na operační systém. Z toho vyplývá, že grafické rozhraní bude na každém operačním systému vypadat trochu jinak.

18.26.1.1. Swing

Druhou standardní knihovnou, jíž se budeme zabývat především, je Swing. Vnitřní fungování Swingu se od AWT velmi liší, jelikož se jedná o tzv. *lightweight* knihovnu, která provádí (téměř) veškeré kreslení komponent sama (tj. v Javě, nikoliv delegováním na systémové binárky).

Tento přístup s sebou nese řadu výhod – jednak dobrou přenositelnost jak samotné knihovny, tak programů v ní napsaných, za druhé pak širší škálu poskytovaných funkcí, jelikož se knihovna nemusí zabývat tím, zda-li daný systém konkrétní funkcionalitu podporuje.

Nevýhodou Swingu pak jsou vyšší náklady na vykreslování, kvůli kterým bylo javovské grafické rozhraní často kritizováno. Druhou stranou mince však je, že za většinu problémů s rychlostí rozhraní mohli programátoři, kteří ignorovali některé základní zákonitosti tvorby grafických aplikací – například, že vlákno, které kreslí rozhraní, nesmí dělat žádné další nákladné operace (protože rozhraní pak po dobu zpracování operace vymrzne...).

18.26.1.2. Společné vlastnosti AWT a Swingu

Ačkoliv Swingové komponenty z balíčku *java.awt* dědí, tak není vhodné tyto dvě knihovny kombinovat, jelikož komponenty samotné nejsou zcela kompatibilní (zejména na nižších verzích Javy).

Poslední zmínkou pak je, že obě knihovny jsou vícevláknové a ani jedna z nich není vláknově bezpečná (a ani nemůže být, protože synchronizace s sebou nese příliš vysoké náklady). Je tedy zapotřebí dávat velký pozor na to, abychom nemodifikovali stav jedné komponenty z více vláken, jelikož by to mohlo skončit havárií programu.

18.26.1.3. Zpracování událostí

Grafické rozhraní je založeno na zpracovávání událostí. Pod pojmem událost si můžeme představit jakoukoliv uživatelskou akci – zmáčknutí klávesy, přejetí kurzoru přes komponentu,

focus na komponentu, klik myši, držení tlačítka myši, jeho puštění a tak podobně. Ve všech těchto případech je upozorněna naše aplikace prostřednictvím k tomu určeného vlákna, jež se ve Swingu nazývá *event dispatching thread*, a které provede příslušnou reakci.

Všimněme si, že v tomto případě se již Swing nechová jako knihovna, ale jako aplikace, která volá náš program, jenž tak vystupuje v roli knihovny. Této situaci, která je charakteristická tzv *holywoodským principem* (*don't call us, we'll call you*), říkáme *inversion of control* (*inverze řízení*). Proto také nebudeme Swing označovat jako knihovnu, ale jako *rámec* (*framework*).

```
/**
 * Hello world - Swing framework
 * @author
 */
public class HelloWorld {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //Asynchronne pridej do fronty udalosti zadost o vykresleni
GUI
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createHelloWorld();
            }
        });
    }
    public static void createHelloWorld(){
        JFrame frame = new JFrame("Hello world frame"); //okno
        frame.add(new JLabel("Hello world"));
        frame.pack(); //vyskladej komponenty tak, aby byla respekto-
vana jejich vyska a sirka
        frame.setLocation(100, 100); //levy horni roh bude na sourad-
nici [100, 100]

        //Pri zmacknuti krizku zavri celou aplikaci (tj. vsechna
vlakna)
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true); //zobraz okno
    }
}
```

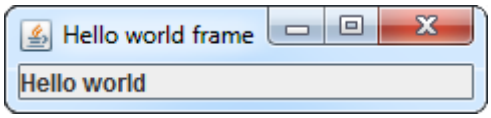
18.26.2. Volání

Grafická aplikace začíná stejně jako konzolová voláním metody *main(String[] args)*. Jak jsme si již uvedli, tak ve Swingu existuje privilegované vlákno – *event dispatching thread*, které se stará o obsluhu všech událostí. Jeho další vlastností pak je, že je to jediné vlákno, které smí modifikovat uživatelské rozhraní. Z tohoto důvodu nesmíme pro vykreslení okna s naším sdělením použít výchozí vlákno aplikace.

Pro popis akce, kterou má *event dispatching thread* vykonat, použijeme objekt typu *Runnable*, který obsahuje pouze metodu *run()* (zde je tento objekt instanciován pomocí anonymní třídy). Jedinou operací bude volání metody *createHelloWorld()*, která provádí vykreslení okna.

Samotné předání pak provedeme pomocí volání *SwingUtilities.invokeLater(Runnable run)*, které asynchronně uloží námi vytvořený objekt typu runnable do fronty událostí, jíž *event dispatching thread* zpracovává. Toto volání taktéž vytvoří všechna obslužná vlákna, která grafická aplikace potřebuje.

18.26.3. Tvorba rozhraní



Tvorba rozhraní v metodě *createHelloWorld()* je poměrně přímočará. Nejprve si vytvoříme objekt okna typu *JFrame* a okno pojmenujeme „*Hello world frame*“. Do oka pak vložíme nápis (*JLabel*) „*Hello world*“. Abychom zajistili, že okno bude mít správnou velikost, zavoláme metodu *pack()*, která zjistí preferovanou velikost všech dceřiných komponent a změní velikost okna, kterému pak určíme pozici levého horního rohu na 100px vodorovně a svisle od levého horního rohu monitoru pomocí metody *setPosition(int x, int y)*.

Poté vybereme operaci, která se provede při zavírání okna (např. kliknutí na křížek okna). Zde volíme zavření celé aplikace – tj. zničení všech oken a vláken. Mezi další volby patří například zrušení pouze daného okna (*DISPOSE_ON_CLOSE*), kde musíme mít na paměti, že aplikace terminuje až v okamžiku, kdy je zničeno poslední okno a doběhne poslední vlákno, které není označeno jako *daemon*.

Nakonec celé oko učiníme viditelným pro uživatele voláním *setVisible(true)*.

18.26.4. Přiřazování akcí komponentám

Každé uživatelské rozhraní je postaveno na interakci *uživatel-aplikace*. Uživatel klikne na tlačítko, aplikace zobrazí dialogové okno, uživatel vyplní data a odklepne okno. Otázkou však zůstává, jak tento proces implementovat.

Naivní řešení by bylo, aby se objekt dialogu nejprve periodicky ptal tlačítka, jestli už bylo zmáčknuto, a až by se dozvěděl, že ano, tak by se zobrazil. Pak by se periodicky ptal formuláře, jestli již byl vyplněn a odklepnut, a v okamžiku kdy ano, tak by se dialog opět zrušil. Toto řešení by však s sebou neslo řadu nevýhod – tou největší je stanovení intervalu, ve kterém se mají komponenty dotazovat na stav. Pokud bychom jej stanovili malý (třeba jednou za sekundu), tak by uživatel pozoroval velké prodlevy. Pokud bychom jej stanovili příliš malý (tisíckrát za vteřinu), tak by tyto dotazy spolehlivě zahltily celou aplikaci.

18.26.5. Návrhový vzor observer

Řešením této zapeklité situace je návrhový vzor *observer*, který říká, že pozorovatelné komponenty musí fungovat jako registr pozorovatelů. V okamžiku provedení uživatelské akce pak pozorovaná komponenta upozorní všechny registrované odběratele (ve *Swingu* se pozorovateli říká *listener* (*posluchač*)).

Pokud tento vzor vztáhneme na náš příklad s dialogovým oknem, tak se okno nejprve zaregistruje u tlačítka. Tlačítko v okamžiku kliku upozorní okno a toto se zobrazí, aniž by se muselo jakýmkoliv způsobem periodicky dotazovat na stav tlačítka.

18.26.5.1. Příklad v praxi

Ukažme si výše zmíněný příklad v praxi. Při jeho implementaci využijeme kromě již známé komponenty *JFrame* také komponentu *JButton* (tlačítko), *JOptionPane* (pomocná třída pro tvorbu dialogových oken) a konečně interface *ActionListener*, pomocí něhož implementujeme akci zobrazení dialogu.

Pro jednoduchost bude dialog obsahovat pouze zprávu obsahující text uvedený na zmáčknutém tlačítku.

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    //Asynchronne pridej do fronty udalosti zadost o vykresleni GUI
    SwingUtilities.invokeLater(new Runnable() {
```

```
            @Override
            public void run() {
                createAndShowGUI();
            }
        });
    }

    /**
     * Vytvor a zobraz uzivatelske rozhrani
     */
    public static void createAndShowGUI(){
        JFrame frame = new JFrame("JButton frame"); //okno

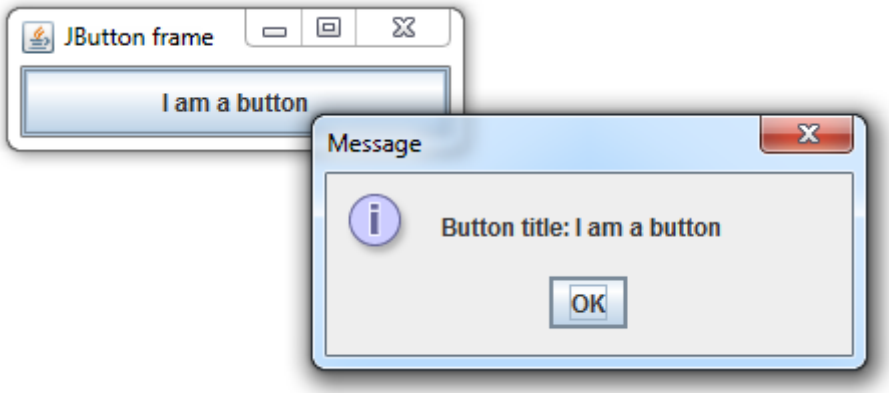
        final JButton button = new JButton("I am a button"); //tlacitku s
nazvem

        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(button, "Button title: " +
button.getText(), "Message", JOptionPane.INFORMATION_MESSAGE);
            }
        });

        frame.add(button);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Pri zma-
cknuti krizku zavri celou aplikaci (tj. pozabijej vsechna vlakna)
        frame.pack(); //prizpusob velikost okna
        frame.setLocation(100, 100); //levy horni roh bude na souradnici
[100, 100]

        frame.setVisible(true); //zobraz okno
    }
```



V metodě *main* asynchronně zavoláme metodu *createAndShowGUI*. Připomeňme si, že *Swing* není vláknově bezpečný a veškeré akce týkající se rozhraní musí být uskutečněny v *event dispatching threadu* (což zajišťuje právě výše uvedený kód).

V samotné metodě *createAndShowGUI* vytvoříme uživatelské rozhraní. Nejprve vytvoříme nové okno – *JFrame*. Poté do něj vložíme nové tlačítko, kterému napřed přiřadíme text při jeho konstrukci a akci voláním *addActionListener*. Na závěr pak provedeme povinné operace – nastavíme, že po kliknutí na křížek okna dojde k uzavření celé aplikace, přizpůsobíme velikost okna obsaženým komponentám (tlačítku), posuneme okno na pozici [100, 100] a samotné okno zobrazíme.

Implementace akce je velmi jednoduchá. Pouze zavoláme metodu na pomocné třídě *JOptionPane*. Tato pak zajistí zobrazení modálního okna s příslušným titulkem, textem, informačním piktogramem a tlačítkem na odklepnutí dialogu. Veškeré operace v listenerech jsou prováděny *event dispatching threadem* – můžeme zde proto bez obav provádět změny v uživatelském rozhraní.

18.26.6. Dědění vlastních komponent

Jedním ze základních způsobů, jak vytvořit vlastní komponentu, je ji oddědit od nějaké již existující. Pokud bychom htěli naše tlačítko použít vícekrát, tak bychom jej implementovali pomocí veřejné třídy dědící z *JButton*:

```
/**
 * Tlacitko zobrazujici pri akci dialog se svym popiskem (textem)
 * @author
 */
public class TitleJButton extends JButton{
    public TitleJButton(String text) {
        super(text);
        addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(TitleJButton.this, "Button title: " + TitleJButton.this.getText(), "Message", JOptionPane.INFORMATION_MESSAGE);
            }
        });
    }
}
```

V tomto případě si vystačíme s překrytím konstruktoru. Nejprve zavoláme konstruktor předka a předáme mu text, který má být vypsán na tlačítku – konstruktor předka bychom v případě *Swingu* měli volat (téměř) vždy, protože v předkovi obvykle dochází k inicializaci grafického kontextu a datového modelu. Poté přidáme akci, která se provede po kliknutí na tlačítko. Zde si všimněme konstrukce *TitleJButton.this*, která značí, že se odvoláváme na *this* kontext obalujícího objektu typu *TitleJButton*. Prosté volání *this* by nás odkázalo na kontext objektu listeneru.

18.25.7. JPanel

Všem třídám, které dědí z třídy *Container* (*dokumetace*), lze přiřadit layout manager (správce rozmístění). Správce rozmístění zajistí, že přidané podkomponenty budou vyskládány určitým způsobem.

Typickým příkladem kontejneru je *JPanel* (*dokumentace*), jehož využíváme primárně pro tvorbu složených komponent. *JPanel* sám o sobě nemá žádnou vizuální reprezentaci, stejně tak nepřidává žádné chování.

18.25.8. Správci rozmístění

18.25.8.1. FlowLayout

Nejjednodušším rozmisťovačem je *FlowLayout*, který umišťuje komponenty do řádku jednu za druhou v pořadí, v němž byly do přidány do daného kontejneru. V případě, že se komponenty nevejdou do jednoho řádku, tak zbylé přetečou do řádku nového.

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    //Asynchronne pridej do fronty udalosti zadost o vykresleni GUI
    SwingUtilities.invokeLater(new Runnable() {
```

```
            @Override
            public void run() {
                createAndShowGUI();
            }
        });
    }

    /**
     * Vytvor a zobraz uzivatelske rozhrani
     */
    public static void createAndShowGUI(){
        JFrame frame = new JFrame("Layout demonstration"); //okno

        frame.add(createLayoutPanel());

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Pri zma-
cknuti krizku zavri celou aplikaci (tj. pozabijej vsechna vlakna)
        frame.pack(); //prizpusob velikost okna
        frame.setLocation(100, 100); //pravy horni roh bude na souradnici
[100, 100]

        frame.setVisible(true); //zobraz okno
    }

    /**
     * Vytvor panel s danym rozmistovacem
     * @return panel
     */
    public static JPanel createLayoutPanel(){
        JPanel panel = new JPanel();

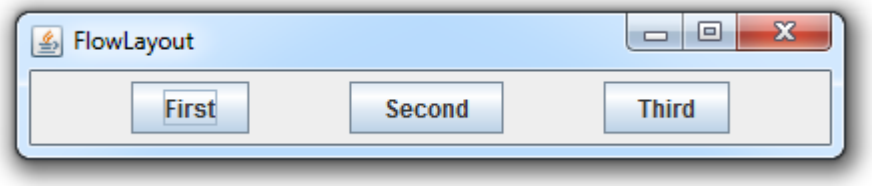
        FlowLayout l = new FlowLayout(FlowLayout.CENTER); //zarovnano na
stred
        l.setHgap(50); //horizontalni mezera mezi obsazenymi komponentami
v pixelech

        panel.setLayout(l);

        panel.add(new JButton("First"));
        panel.add(new JButton("Second"));
        panel.add(new JButton("Third"));

        return panel;
    }
```

Ve výše uvedeném kódu je pro nás zajímavá pouze metoda *createLayoutPanel*, jež vytváří *JPanel* s příslušným layoutem a několika obsaženými komponentami (tlačítky). V dalších příkladech správců rozmístění si již budeme uvádět pouze ji.



18.25.8.2. BorderLayout

BorderLayout zarovnává komponenty do světových stran a na střed (sever, jih, východ, západ, střed). BorderLayout je jedním z nejpoužitelnějších správců rozmístění, jelikož téměř každá aplikace má nějakou hlavičku (např. menu umístěné na severu),

obsah umístěný na středu a patičku na jihu. Na východě a západě se pak často nachází nějaký typ kontextové navigace (obvykle nějaká stromová komponenta).

Všechny umístěné komponenty jsou vždy roztaženy tak, aby zabraly veškeré dostupné místo.

```
public static JPanel createLayoutPanel() {
    JPanel panel = new JPanel();

    BorderLayout l = new BorderLayout();

    panel.setLayout(l);

    panel.add(new JButton("North"), BorderLayout.NORTH);
    panel.add(new JButton("South"), BorderLayout.SOUTH);
    panel.add(new JButton("East"), BorderLayout.EAST);
    panel.add(new JButton("West"), BorderLayout.WEST);
    panel.add(new JButton("Center"), BorderLayout.CENTER);

    return panel;
}
```



18.25.8.3. BoxLayout

BoxLayout umísťuje komponenty buď do jednoho řádku nebo sloupce podle nastavení, které předáme jeho konstruktoru. Na rozdíl od *FlowLayoutu* komponenty nikdy nepřetečou do dalšího řádku (sloupce), ale jsou vždy zmenšeny nebo oříznuty tak, aby se vešly do daného prostoru.

Pro tvorbu *BoxLayoutu* také můžeme použít třídu *Box*, která nám umožní jednak vytvářet své instance (*Box* je kontejner, který implicitně obsahuje *BoxLayout* jako správce rozmístění) a za druhé pak s jeho pomocí můžeme vytvářet neviditelné komponenty *tyčí* (*strut*) a *tmelu* (*glue*).

Horizontální a vertikální tyče jsou komponenty, které mají buď nulovou výšku, nebo nulovou šířku a zajišťují, že se předchozí a následující komponenty nepřiblíží více, než je délka tyče. Komponenta tmelu se pak snaží zabrat veškerý přebytečný prostor – slouží jako výplň.

```
public static JPanel createLayoutPanel() {
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout());

    Box left = Box.createVerticalBox();
    Box right = Box.createVerticalBox();

    panel.add(left, BorderLayout.WEST);
    panel.add(right, BorderLayout.CENTER);

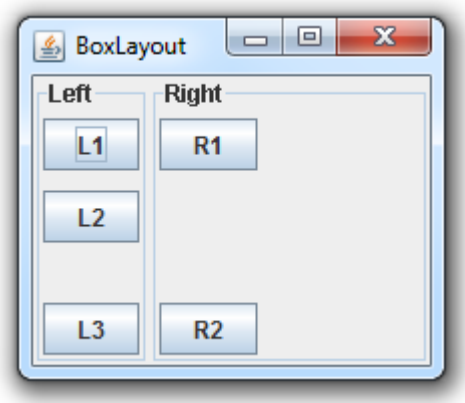
    left.add(new JButton("L1"));
```

```
        left.add(Box.createVerticalStrut(10));
        left.add(new JButton("L2"));
        left.add(Box.createVerticalStrut(30));
        left.add(new JButton("L3"));

        right.add(new JButton("R1"));
        right.add(Box.createVerticalGlue());
        right.add(new JButton("R2"));

        left.setBorder(BorderFactory.createTitledBorder("Left"));
        right.setBorder(BorderFactory.createTitledBorder("Right"));

        return panel;
    }
```



Tento příklad nám zároveň ukazuje, že je možné (a žádoucí) kombinovat více správců rozmístění. Zde pro rozmístění na nejvyšší úrovní používáme *BorderLayout*, který zajistí roztažení komponent na maximální možné rozměry (při zvětšování má vždy přednost komponenta na středu). V levém panelu pak umísťujeme do boxu tři tlačítka a ponecháváme mezi nimi mezery 10 a 30 pixelů. V pravém sloupci pak umísťujeme dvě tlačítka a dáváme mezi ně tmel, čímž zajistíme, že mezi nimi bude situován veškerý volný prostor – tj. jedno tlačítko bude nahoře a druhé dole. Nakonec oběma boxům přidáme ohraničení s popiskem.

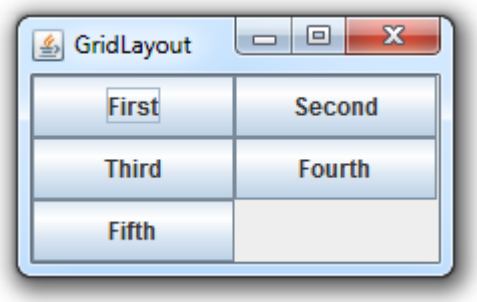
18.25.8.4. GridLayout

GridLayout, jak již název napovídá, umisťuje komponenty do pravoúhlé mřížky. Konstruktoru můžeme předat 2 parametry – počet komponent na šířku a na výšku – z nichž může maximálně jeden nulový. Nulový argument znamená, že se v daném rozměru bude mřížka automaticky natahovat.

```
    public static JPanel createLayoutPanel() {
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(0, 2)); //nedefinovany pocet
radku, 2 sloupce

        panel.add(new JButton("First"));
        panel.add(new JButton("Second"));
        panel.add(new JButton("Third"));
        panel.add(new JButton("Fourth"));
        panel.add(new JButton("Fifth"));

        return panel;
    }
```



18.25.8.5. GridBagLayout

Posledním layoutem, který si dnes představíme, je *GridBagLayout*. Jedná se o zobecnění *GridLayoutu*, ve němž může komponenta zabírat více buněk. *GridBagLayout* je ze všech uvedených správců nejsilnější a nejkomplexnější. Na druhou stranu je však mnohdy snazší (a přehlednější) použít složeninu několika jednodušších správců.

Pro každou komponentu umístěnou v *GridBagLayoutu* musíme vytvořit objekt typu *GridBagConstraints*, který specifikuje její pozici pomocí následujících vlastností:

- **gridx**, **gridy** – Pozice buňky, do níž bude komponenta umístěna.
- **gridwidth**, **gridheight** – Počet buněk na výšku a na šířku, které komponenta zabere.
- **weightx**, **weighty** – Poměr, v jakém bude přebytečné místo rozděleno mezi komponenty.
- **ipadx**, **ipady** – Určuje počet pixelů, o něž má být komponenta větší, než je její minimální velikost.
- **fill** – Určuje, jakým způsobem má být komponenta roztažena, aby vyplnila prostor, který jí byl vymezen (roztáhnout horizontálně, vertikálně, v obou dimenzích).
- **insets** – Specifikuje vnitřní okraje. Tj. kolik má být volného prostoru kolem komponenty.

```
public static JPanel createLayoutPanel() {
    JPanel panel = new JPanel();
    panel.setLayout(new GridBagLayout());

    GridBagConstraints gc1 = new GridBagConstraints();
    gc1.gridx = 0;
    gc1.gridy = 0;
    gc1.anchor = GridBagConstraints.WEST; //bunka zarovnana vlevo
    gc1.fill = GridBagConstraints.VERTICAL; //komponenta natazena na
vysku

    panel.add(new JButton("First"), gc1);

    GridBagConstraints gc2 = new GridBagConstraints();
    gc2.gridx = 0;
    gc2.gridy = 1;
    gc2.fill = GridBagConstraints.BOTH; //komponenta natazena do obou
rozmeru

    panel.add(new JButton("Second"), gc2);

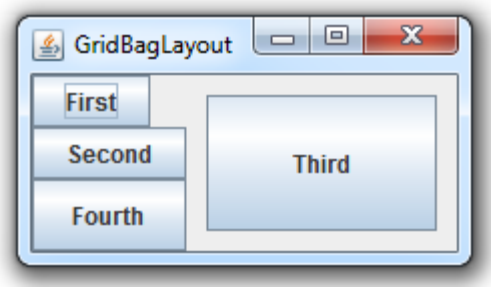
    GridBagConstraints gc3 = new GridBagConstraints();
    gc3.gridx = 2;
    gc3.gridy = 0;
    gc3.anchor = GridBagConstraints.CENTER; //bunka zarovnana na
stred
    gc3.fill = GridBagConstraints.BOTH; //komponenta natazena do obou
rozmeru
    gc3.gridheight = 3; //komponenta zabere na vysku 3 bunky
```

```
            gc3.insets = new Insets(10, 10, 10, 10); //vnitrni okraj bunky
10px na kazdou stranu
            gc3.weightx = 1; //komponenta zabere prebytecne misto na sirku
            panel.add(new JButton("Third"), gc3);


            GridBagConstraints gc4 = new GridBagConstraints();
            gc4.gridx = 0;
            gc4.gridy = 2;
            gc4.ipady = 10; //zvetsi komponentu o 10px na vysku
            gc4.anchor = GridBagConstraints.WEST; //bunka zarovnana vlevo
            gc4.fill = GridBagConstraints.BOTH; //komponenta natazena do obou
rozmeru

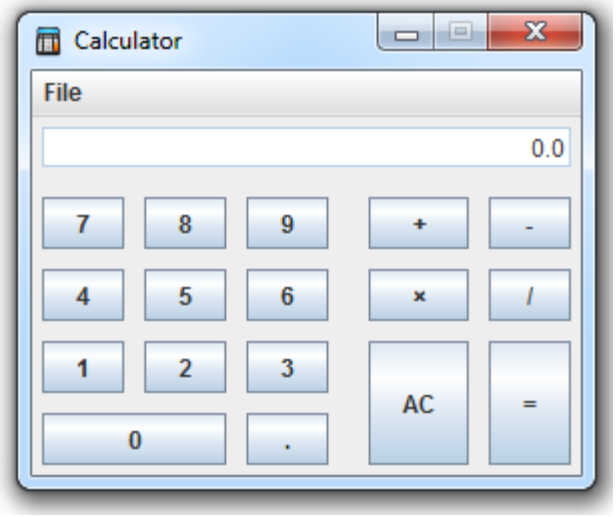
            panel.add(new JButton("Fourth"), gc4);

            return panel;
      }
```



18.26. Závěrem

Závěrem si ukážeme jeden program, který bude úplný včetně grafického prostředí. Bude se jednat o kalkulátor, který bude ve stylu *MS Windows Calculator* a bude podporovat operace sčítání, odčítání, násobení a dělení v plovoucí řádové čárce a infixové notaci. Kalkulačka bude mít operace reset a rovná se. Grafické rozraní půjde ovládat jak pomocí myši, tak pomocí klávesnice.



18.26.1. Architektura

Nyní, když již víme, co budeme programovat, tak si musíme rozmyslet, jakým způsobem budeme aplikaci vnitřně členit – tj. musíme si rozmyslet její architekturu (kterou posléze

rozvedeme do balíčků a konkrétních tříd – vizte hierarchie tříd výsledného projektu v rámečku vpravo).

Aplikace se bude skládat ze tří hlavních částí:

Model: Model reprezentuje stav aplikace – data, která kalkulačka potřebuje pro svoji práci. V našem případě se bude jednat o jednu třídu, která bude držet mezivýsledek operací, aktuálně používaný operátor (podporujeme infixovou notaci) a stav displaye kalkulačky (tj. uživatelský vstup).

Action: Druhou částí aplikace je její vlastní chování, respektive reakce na stisknutí jednotlivých tlačítek. V tomto případě si již nevystačíme pouze s jednou třídou, jelikož uživatelských akcí je mnoho typů – stisk tlačítka číslice pouze přidá příslušnou hodnotu na konec vstupu, binární operátory ovšem vykonávají zcela rozdílné akce (sčítání, odčítání...) a každému z nich proto musí odpovídat separátní akce (třída). To samé pak platí pro tlačítko resetu a rovnosti.

View: Poslední částí je view (pohled). View obsahuje veškeré třídy, které se týkají samotného uživatelského rozhraní (vytvářejí okno, layout atp.). Kdybychom chtěli být zcela korektní, tak bychom pod view zahrnuli i akce, jelikož se v našem podání bude jednat pouze o operace nad uživatelským rozhraním (třídy akcí nebudou zcela nezávislé na technologii rozhraní).

18.26.2. Proč aplikaci členit zrovna tímto způsobem?

Při objektovém návrhu je vhodné se držet tří základních principů: single responsibility, high cohesion, low coupling.

18.26.2.1. Single responsibility

Princip jedné odpovědnosti (*single responsibility principle*) nám říká, že každá třída by měla vykonávat jednu jasně definovanou odpovědnost. Pokud takto definované třídy zakryjeme rozhraním, tak je v případě potřeby snadno vyměníme v okamžiku, kdy se ukáže, že je daná funkcionalita implementována nevhodně nebo nedostatečně.

18.26.2.2. High cohesion a low coupling

High cohesion (*vysoká soudržnost*) a low coupling (*nízká provázanost*) jsou dva principy, které nás nutí vytvářet software modulárně. Náš kód je vysoce soudržný, pokud se každý z modulů (třída, balíček, fakticky jakákoliv logická část) zabývá jasně vymezenou množinou odpovědností. High cohesion můžeme chápat jako určité zobecnění principu jedné odpovědnosti.

Low coupling je princip, který vysokou soudržnost doplňuje – říká totiž, že moduly musí mít mezi sebou pouze minimum vazeb (abychom je mohli pohodlně zaměňovat). Nemůžeme proto v rámci principu vysoké soudržnosti vytvářet jenom třídy o jedné metodě s jedním řádkem kódu, protože by takto vytvořené třídy mezi sebou měly obrovské množství vazeb (a porušovaly by proto low coupling).

18.26.3. Implementace

18.26.3.1. Model

Model (paměť kalkulačky) bude reprezentován třídou *CalculatorModel* s atributy *storedResult* (mezivýsledek), *currentInput* (uživatelský vstup), *decimalPointPresent* (indikace přítomnosti desetinné čárky), *operator* (aktivní operátor) a *defaultOperator* (výchozí operátor).

Přítomnost řádové čárky bychom samozřejmě mohli také vždy znovu zjistit průchodem přes řetězec vstupu, ale to by bylo zbytečné mrhání procesorovým časem, zvlášť když tuto informaci dostáváme při stisku příslušného tlačítka kalkulačky.

Důvod, proč potřebujeme dva uložené operátory vychází z toho, jak budeme implementovat vnitřní logiku kalkulačky. V okamžiku, kdy uživatel zadá další operátor tj. 4*3- , tak se provede výpočet předchozích operace a na displayi se mu zobrazí aktuální mezivýsledek (12). Otázkou ovšem zůstává, jak se má kalkulačka zachovat při zadání prvního plus. Odpovědí je výchozí operátor, což v našem případě bude plus, které v kombinaci s uloženým mezivýsledkem (na začátku 0,0) zajistí korektní chování kalkulačky (tj. 0+4*3).

```
/**
 * Model (data) kalkulacky
 * @author
 */
public class CalculatorModel {

    private double storedResult;
    private String currentInput;
    private boolean decimalPointPresent;
    private OperatorActionIface operator;
    private OperatorActionIface defaultOperator;

    public CalculatorModel() {
        resetModel();
    }

    public final void resetModel() {
        currentInput = "";
        storedResult = 0;
        decimalPointPresent = false;
        operator = defaultOperator;
    }

    /**
     * @return the storedResult
     */
    public double getStoredResult() {
        return storedResult;
    }

    /**
     * @param storedResult the storedResult to set
     */
    public void setStoredResult(double storedResult) {
        this.storedResult = storedResult;
    }

    /**
     * @return the currentInput
     */
    public String getCurrentInput() {
        return currentInput;
    }

    /**
     * @param currentInput the currentInput to set
     */
    public void setCurrentInput(String currentInput) {
        this.currentInput = currentInput;
    }

    /**
     * @return the decimalPointPresent
```

```
         */
        public boolean isDecimalPointPresent() {
            return decimalPointPresent;
        }

        /**
         * @param decimalPointPresent the decimalPointPresent to set
         */
        public void setDecimalPointPresent(boolean decimalPointPresent) {
            this.decimalPointPresent = decimalPointPresent;
        }

        /**
         * @return the operator
         */
        public OperatorActionIface getOperator() {
            return operator;
        }

        /**
         * @param operator the operator to set
         */
        public void setOperator(OperatorActionIface operator) {
            this.operator = operator;
        }

        /**
         * Nastavi vychozi operator pro tento model. Pokud model dosud
nema
         * nastaven zadny aktivni operator, tak bude nastaven tento.
Stejne
         * tak v pripade resetu modelu dojde k prednastaveni tohoto ope-
ratoru.
         * @param defaultOperator the defaultOperator to set
         */
        public void setDefaultOperator(OperatorActionIface defaultOpera-
tor) {
            this.defaultOperator = defaultOperator;
            if (this.operator == null) {
                operator = defaultOperator;
            }
        }
    }
```

Kromě *getterů* a *setterů* obsahuje model operaci *reset()*, která jej vynuluje (vyčistí vstup, mezivýsledek, přítomnost desetinné čárky a nastaví výchozí operátor). Setter výchozího operátoru také nastaví aktivní operátor, není-li již nastaven.

18.26.3.2. CalculatorActionIface

Po kliku na libovolné tlačítko kalkulačky musí dojít k nějaké reakci. Specifikujme proto rozhraní, které zaváže implementující třídu tuto reakci poskytovat.

```
    /**
     * Interface specifikuje akci provedenou po stlaceni tlacitka
     * @author
     */
    public interface CalculatorActionIface {
        /**
         * Provede akci zpusobenou stlacenim tlacitka
         */
        public void performInputAction();
    }
```

18.26.3.3. AbstractCalculatorAction

Všechny akce kalkulačky budou nějakým způsobem interagovat s modelem a s displayem kalkukačky (který bude reprezentovám grafickou komponentou *JTextField*). Abstraktní třída *AbstractCalculatorAction* nám proto bude sloužit jako společný předek všech akcí.

```
/**
 * Spolecny predek pro akce na kalkulacce
 * @author
 */
public abstract class AbstractCalculatorAction implements CalculatorActionIface{

    protected JTextField resultField;
    protected CalculatorModel model;


    public AbstractCalculatorAction(JTextField resultField, CalculatorModel model) {
        this.resultField = resultField;
        this.model = model;
    }

    @Override
    public abstract void performInputAction();

    /**
     * @param resultField the resultField to set
     */
    public void setResultField(JTextField resultField) {
        this.resultField = resultField;
    }

    /**
     * @param model the model to set
     */
    public void setModel(CalculatorModel model) {
        this.model = model;
    }

    /**
     * @return the resultField
     */
    public JTextField getResultField() {
        return resultField;
    }

    /**
     * @return the model
     */
    public CalculatorModel getModel() {
        return model;
    }
}
```

Všimněme si, že k oběma polím budeme zvenčí přistupovat skrze *gettery* a *settery*, avšak podtřídy (akce samotné) k nim mohou přistupovat přímo díky modifikátoru přístupu *protected*. Jelikož je třída abstraktní, tak můžeme dále metodu *performInputAction()* ponechat nadále neimplementovanou (budou ji implementovat specifičtí potomci).

18.26.3.4. DecimalPointAction

```
/**
```

```
       * Akce provedena po stisku tlacitka desetinne carky
       * @author
       */
      public class DecimalPointAction extends AbstractCalculatorAction {

          public DecimalPointAction(JTextField resultField, CalculatorModel
model) {
              super(resultField, model);
          }

          @Override
          public void performInputAction() {
              if (!model.isDecimalPointPresent()) {
                  getModel().setDecimalPointPresent(true);
                  if (getModel().getCurrentInput().length() > 0) {
                      getModel().setCurrentInput(getModel().getCurren-
tInput() + ".");
                  } else {
                      getModel().setCurrentInput("0.");
                  }
                  getResultField().setText(getModel().getCurrentInput());
              }
          }
      }
```

Po stisku tlačítka desetinné čárky nejprve zkontrolujeme, že čárka již není přítomna. Pokud není, tak v modelu nastavíme příznak její přítomnosti (pokud je čárka přítomna, tak metoda terminuje). Pokud je uživatelský vstup neprázdný, tak tečku přidáme na jeho konec, v opačném případě před ni předřadíme nulu. Na závěr zaktualizujeme stav displaye kalkukačky.

18.26.3.5. NumberAction

Akce po stisku čísla je ještě jednodušší – pouze toto číslo přidáme na konec vstupu a zaktualizujeme display.

```
      /**
       * Akce provedena po stisku tlacitka hodnoty (cisla)
       * @author
       */
      public class NumberAction extends AbstractCalculatorAction {

          private String value;

          public NumberAction(JTextField resultField, CalculatorModel mo-
del, String value) {
              super(resultField, model);
              this.value = value;
          }

          @Override
          public void performInputAction() {
              getModel().setCurrentInput(getModel().getCurrentInput() + va-
lue);
              getResultField().setText(getModel().getCurrentInput());
          }
      }
```

18.26.3.6. ResetAction

Akce resetu kalkulačky vynuluje model a nastaví display na výchozí hodnotu.

```
      /**
       * Akce resetovaciho tlacitka
       * @author
```

```
     */
    public class ResetAction extends AbstractCalculatorAction {

        public ResetAction(JTextField resultField, CalculatorModel model)
{
            super(resultField, model);
        }

        @Override
        public void performInputAction() {
            getModel().resetModel();
            getResultField().setText(String.valueOf(getModel().getSto-
redResult()));
        }

    }
```

18.26.3.7. ResultAction

Po stisku tlačítka rovnosti se nejprve vykoná poslední uložený operátor a poté se na displayi zobrazí výsledek.

```
    /**
     * Akce tlacitka rovnost
     * @author
     */
    public class ResultAction extends AbstractCalculatorAction{

        public ResultAction(JTextField resultField, CalculatorModel mo-
del) {
            super(resultField, model);
        }

        @Override
        public void performInputAction() {
            getModel().getOperator().performOperatorAction();
            getResultField().setText(String.valueOf(getModel().getSto-
redResult()));
        }

    }
```

18.26.3.8. OperatorActionIface

Operátory (sčítání, odčítání, násobení, dělení) jako takové reagují jak na stisk odpovídajícího tlačítka (uloží se do paměti jakožto aktivní operátory), tak dále musí poskytovat metodu, která vykoná samotnou operaci v okamžiku, kdy jsou již známy oba operandy. Tuto metodu definuje rozhraní *OperatorActionIface*.

```
    /**
     * Interface specifikujici akci operatoru
     * @author
     */
    public interface OperatorActionIface {
        /**
         * Provede akci operatoru
         */
        public void performOperatorAction();
    }
```

18.26.3.9. AbstractOperatorAction

Analogicky s tlačítky vstupu vytvoříme i tlačítkům operátorů abstraktního předka, který bude implementovat obecnou funkcionalitu. Její první část můžeme přímo oddědit z třídy *AbstractCalculatorAction* (abstraktní předek všech tlačítek). Dále pak implementujeme rozhraní *OperatorActionIface*.

```
/**
 * Abstraktni predek akci operatoru
 * @author
 */
public abstract class AbstractOperatorAction extends AbstractCalcula-
torAction implements OperatorActionIface {

    public AbstractOperatorAction(JTextField resultField, Calculator-
Model model) {
        super(resultField, model);
    }

    @Override
    public void performInputAction() {
        getModel().getOperator().performOperatorAction();
        getModel().setOperator(this);
    }

    @Override
    public void performOperatorAction() {
        if (getModel().getCurrentInput().length() > 0) {
            performActualOperatorAction();
            getModel().setCurrentInput("");
            getResultField().setText(String.valueOf(getModel().get-
StoredResult()));
        }
    }

    /**
     * Vykona samotnou akci operatoru
     */
    abstract protected void performActualOperatorAction();
}
```

Kód metody *performInputAction()* není příliš zajímavý – pouze vykonáme operaci předchozího operátoru a uložíme tento operátor do modelu.

Metoda *performOperatorAction()* ale obsahuje jeden zajímavý trik. Všechny akce operátorů musí nejprve zkontrolovat, jestli mají k dispozici druhý operand (pokud ne, tak není co počítat). Pokud jej mají, tak mohou vykonat daný výpočet/akci. Po provedení akce vyčistí uživatelský vstup a zobrazí na displayi výsledek. Abychom nemuseli tento kód stále opakovat ve všech specifických potomcích, tak vytvoříme novou abstraktní metodu *performActualOperatorAction()*, jejímž prostřednictvím budou potomci implementovat specifické chování daného operátoru. Tuto metodu opatříme modifikátorem přístupu *protected*. Tímto pro vnější svět vytvoříme iluzi, že veškerou práci dělá metoda specifikovaná v rozhraní *performOperatorAction()*, která však slouží pouze jako šablona (tzv. *template method)*.

18.26.3.10. PlusAction, MinusAction, MultiplicationAction, DivisionAction

Samotné operace sčítání, odčítání, násobení a dělení jsou již nyní naprosto triviální.

```
/**
 * Operace scitani
 * @author
 */
public class PlusAction extends AbstractOperatorAction {
```

```
        public PlusAction(JTextField resultField, CalculatorModel model)
{
            super(resultField, model);
        }

        @Override
        public void performActualOperatorAction() {
            getModel().setStoredResult(getModel().getStoredResult() +
Double.valueOf(getModel().getCurrentInput()));
        }

    }

    /**
     * Operace odecitani
     * @author
     */
    public class MinusAction extends AbstractOperatorAction {

        public MinusAction(JTextField resultField, CalculatorModel model)
{
            super(resultField, model);
        }

        @Override
        public void performActualOperatorAction() {
            getModel().setStoredResult(getModel().getStoredResult() -
Double.valueOf(getModel().getCurrentInput()));
        }
    }

    /**
     * Operace nasobeni
     * @author
    */
    public class MultiplicationAction extends AbstractOperatorAction{

        public MultiplicationAction(JTextField resultField, CalculatorMo-
del model) {
            super(resultField, model);
        }


        @Override
        public void performActualOperatorAction() {
            getModel().setStoredResult(getModel().getStoredResult() *
Double.valueOf(getModel().getCurrentInput()));
        }

    }

    /**
     * Operace deleni
     * @author
     */
    public class DivisionAction extends AbstractOperatorAction{

        public DivisionAction(JTextField resultField, CalculatorModel mo-
del) {
```

```
            super(resultField, model);
        }

        @Override
        protected void performActualOperatorAction() {
            getModel().setStoredResult(getModel().getStoredResult() /
Double.valueOf(getModel().getCurrentInput()));
        }

    }
```

18.26.4. View

Poslední částí naší aplikace je náhled (view), který vytvoříme pomocí frameworku *Swing*. Ještě než začneme se samotnou tvorbou rozhraní, tak si napíšeme dvě pomocné třídy – první z nich bude třída tlačítka, které může být ovládáno také pomocí klávesnice, druhou pak potomek třídy *GridBagConstraints*, který nám nabídne dodatečné konstruktory, jež nám v další fázi ušetří hodně psaní.

```
    /**
     * Trida reprezentujici tlacitko kalkulacky
     * @author
     */
    public class JCalculatorButton extends JButton {

        public JCalculatorButton(String text, int key, final CalculatorA-
ctionIface action) {
            super(text);
            ActionListener al = new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    action.performInputAction();
                }
            };
            this.addActionListener(al);
            this.registerKeyboardAction(al, KeyStroke.getKeyStroke(key,
0), JComponent.WHEN_IN_FOCUSED_WINDOW);
        }
    }
```

Tlačítko kalkulačky má pouze vlastní konstruktor, který přijme popis tlačítka, kód asociovaného tlačítka klávesnice a samozřejmě akci samotnou. Text nejprve předáme konstruktoru předka, poté vytvoříme vnitřní třídu listeneru tlačítka, která na při jeho stisku zavolá námi definovanou akci a nakonec pomocí volání *registerKeyboardAction()* akci přiřadíme taktéž ke stisku příslušné klávesy v aktivním okně kalkulačky (bez jakékoliv masky (např. *alt* nebo *shift*)).

Druhá pomocná třída ButtonGridBagConstraints jak již bylo předesláno pouze nabízí dodatečné konstruktory, jež budeme v aplikaci využívat – třída GridbagConstraints obsahuje pouze jeden všemocný konstruktor, který bychom museli stále dokola vyplňovat téměř totožnými hodnotami.

```
    /**
     * Trida pro zjednoduseni konstrukce GridBagConstraints
     * @author
     */
    public class ButtonGridBagConstraints extends GridBagConstraints {

        /**
         * Vychozi sirka vnitrnich okraju
         */
```

```
        public static final int DEFAULT_INSETS = 5;

        /**
         * Zkonstruuje GridBagConstraints pro umisteni komponenty na da-
nych souradnicich
         * s vychozimi okraji (na vsechny strany <span class="bold">DE-
FAULT_INSETS</span>) a sirkou a vyskou
         * pres jednu bunku. Anchor na stredu, roztazeni do obou stran,
vaha 1.
         * @param x souradnice x
         * @param y souradnice y
         */
        public ButtonGridBagConstraints(int x, int y) {
            this(x, y, 1, 1);
        }

        /**
         * Zkonstruuje GridBagConstraints pro umisteni komponenty na da-
nych souradnicich,
         * sirkou a vyskou pres jednu bunku. Anchor na stredu, roztazeni
do obou stran, vaha 1.
         * @param x souradnice x
         * @param y souradnice y
         * @param insets vnitrni okraje
         */
        public ButtonGridBagConstraints(int x, int y, Insets insets) {
            this(x, y, 1, 1, insets);
        }

        /**
         * Zkonstruuje GridBagConstraints pro umisteni komponenty na da-
nych souradnicich
         * s vychozimi okraji (na vsechny strany <span class="bold">DE-
FAULT_INSETS</span>). Anchor na
         * stredu, roztazeni do obou stran, vaha 1.
         * @param x souradnice x
         * @param y souradnice y
         * @param gridWidth pocet bunek, ktere na sirku zabere dana kom-
ponenta
         * @param gridHeight pocet bunek, ktere na vysku zabere dana kom-
ponenta
         */
        public ButtonGridBagConstraints(int x, int y, int gridWidth, int
gridHeight) {
            this(x, y, gridWidth, gridHeight, new Insets(DEFAULT_INSETS,
DEFAULT_INSETS, DEFAULT_INSETS, DEFAULT_INSETS));
        }

        /**
         * Zkonstruuje GridBagConstraints pro umisteni komponenty na da-
nych souradnicich.
         * Anchor na stredu, roztazeni do obou stran, vaha 1.
         * @param x souradnice x
         * @param y souradnice y
         * @param gridWidth pocet bunek, ktere na sirku zabere dana kom-
ponenta
         * @param gridHeight pocet bunek, ktere na vysku zabere dana kom-
ponenta
         * @param insets vnitrni okraje
         */
```

```
        public ButtonGridBagConstraints(int x, int y, int gridWidth, int
gridHeight, Insets insets) {
            super(x, y, gridWidth, gridHeight, 1d, 1d, GridBagConstra-
ints.CENTER, GridBagConstraints.BOTH, insets, 0, 0);
        }
    }
```

Konstruktor předka voláme pouze v jednom nejobecnějším konstruktoru, čímž opět docílíme toho, že se případné změny dotknou pouze minima příkazů.

Nyní již máme vše připraveno ke konstrukci samotného grafického rozhraní.

```
    /**
     * Graficke rozhrani kalkukacky
     * @author
     */
    public class JCalculatorFrame extends JFrame {

        private CalculatorModel model;

        public JCalculatorFrame() {
            this.model = new CalculatorModel();
            createAndShowGUI();
        }

        private void createAndShowGUI() {
            this.setTitle("Calculator");
            this.setLayout(new BorderLayout());
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            try {
                //ikonka (freeware) byla ziskana zde: http://www.iconar-
chive.com/show/mobile-icons-by-webiconset/calculator-icon.html
                this.setIconImage(ImageIO.read(this.getClass().getRe-
source("calculator-icon.png")));
            } catch (IOException ex) {
                //nepovedlo se, nebudeme z toho delat vedu :-), uzivatel
proste neuvidi ikonku, jenom vypiseme stacktrace do konzole
                ex.printStackTrace();
            }

            createMenu();
            createCalculator();

            adjustAndDisplayWindow();
        }

        /**
         * Zarovna okno a zobrazi jej uzivateli
         * @param frame hlavni ramec aplikace
         */
        private void adjustAndDisplayWindow() {
            this.setResizable(false);
            this.pack(); //Nechame spocitat (a zmenit) velikost okna dle
obsazenych komponent

            Toolkit t = Toolkit.getDefaultToolkit(); //z tridy toolkit
muzeme ziskat mnoho uzitecnych informaci o displayi
            //okno zarovname na prostredek displaye
            this.setLocation(t.getScreenSize().width / 2 - this.ge-
tWidth() / 2, t.getScreenSize().height / 2 - this.getHeight() / 2);

            this.setVisible(true); //a nakonec okno ukazeme uzivateli
        }
```

```
        /**
         * Vytvori menu aplikace
         * @param frame hlavni ramec aplikace
         */
        private void createMenu() {
            JMenuBar menuBar = new JMenuBar(); //prouzek menu
            JMenu fileMenu = new JMenu("File"); //jednotliva nabidka
            JMenuItem exit = new JMenuItem("Exit"); //polozka nabidky

            //Hot-key alt+q zpusobi zavreni aplikace (stisk menuitem
exit)
            exit.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q,
ActionEvent.ALT_MASK));

            exit.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent e) {
                    JCalculatorFrame.this.dispose(); //zrus okno, rozpus-
tenim vsech oken muze dojit k ukonceni aplikace
                }
            });

            fileMenu.add(exit); //polozku vlozime do nabidky
            menuBar.add(fileMenu); //nabidku do prouzku
            this.add(menuBar, BorderLayout.NORTH); //prouzek na sever
hlavniho okna (ramce)
        }

        /**
         * Vytvori vlastni kalkulacku
         * @param frame hlavni ramec aplikace
         */
        private void createCalculator() {
            JPanel panel = new JPanel();
            GridBagLayout layout = new GridBagLayout();
            panel.setLayout(layout);

            JTextField resultField = new JTextField(String.valueOf(mo-
del.getStoredResult())); //pole pro vysledek
            resultField.setHorizontalAlignment(JTextField.TRAILING);
//vysledek bude zarovnan na pravou stranu
            resultField.setEditable(false); //nenechame do nej uzivatele
sahat
            resultField.setBackground(Color.WHITE); //editable false za-
barvi pozadi do sede barvy
            panel.add(resultField, new GridBagConstraints(0, 0, 5, 1, 1,
1, GridBagConstraints.CENTER, GridBagConstraints.BOTH, new Insets(5, 5, 10,
5), 0, 0));
            model.setDefaultOperator(new PlusAction(resultField, model));

            final int inset = ButtonGridBagConstraints.DEFAULT_INSETS;
            final int bigInset = inset * 3;

            panel.add(new JCalculatorButton("7", KeyEvent.VK_NUMPAD7, new
NumberAction(resultField, model, "7")), new ButtonGridBagConstraints(0,
1));
            panel.add(new JCalculatorButton("8", KeyEvent.VK_NUMPAD8, new
NumberAction(resultField, model, "8")), new ButtonGridBagConstraints(1,
1));
```

```
            panel.add(new JCalculatorButton("9", KeyEvent.VK_NUMPAD9, new
NumberAction(resultField, model, "9")), new ButtonGridBagConstraints(2,
1));
            panel.add(new JCalculatorButton("4", KeyEvent.VK_NUMPAD4, new
NumberAction(resultField, model, "4")), new ButtonGridBagConstraints(0,
2));
            panel.add(new JCalculatorButton("5", KeyEvent.VK_NUMPAD5, new
NumberAction(resultField, model, "5")), new ButtonGridBagConstraints(1,
2));
            panel.add(new JCalculatorButton("6", KeyEvent.VK_NUMPAD6, new
NumberAction(resultField, model, "6")), new ButtonGridBagConstraints(2,
2));
            panel.add(new JCalculatorButton("1", KeyEvent.VK_NUMPAD1, new
NumberAction(resultField, model, "1")), new ButtonGridBagConstraints(0,
3));
            panel.add(new JCalculatorButton("2", KeyEvent.VK_NUMPAD2, new
NumberAction(resultField, model, "2")), new ButtonGridBagConstraints(1,
3));
            panel.add(new JCalculatorButton("3", KeyEvent.VK_NUMPAD3, new
NumberAction(resultField, model, "3")), new ButtonGridBagConstraints(2,
3));
            panel.add(new JCalculatorButton("0", KeyEvent.VK_NUMPAD0, new
NumberAction(resultField, model, "0")), new ButtonGridBagConstraints(0, 4,
2, 1));
            panel.add(new JCalculatorButton(".", KeyEvent.VK_DECIMAL, new
DecimalPointAction(resultField, model)), new ButtonGridBagConstraints(2,
4));


            panel.add(new JCalculatorButton("+", KeyEvent.VK_ADD, new
PlusAction(resultField, model)), new ButtonGridBagConstraints(3, 1, new In-
sets(inset, bigInset, inset, inset)));
            panel.add(new JCalculatorButton("-", KeyEvent.VK_SUBTRACT,
new MinusAction(resultField, model)), new ButtonGridBagConstraints(4, 1));
            panel.add(new JCalculatorButton("\u00D7", KeyEvent.VK_MULTI-
PLY, new MultiplicationAction(resultField, model)), new ButtonGridBagCon-
straints(3, 2, new Insets(inset, bigInset, inset, inset))); //\u00D7 je
UTF8 kod pro znamenko multiplikace
            panel.add(new JCalculatorButton("/", KeyEvent.VK_DIVIDE, new
DivisionAction(resultField, model)), new ButtonGridBagConstraints(4, 2));

            panel.add(new JCalculatorButton("AC", KeyEvent.VK_DELETE, new
ResetAction(resultField, model)), new ButtonGridBagConstraints(3, 3, 1, 2,
new Insets(inset, bigInset, inset, inset)));
            panel.add(new JCalculatorButton("=", KeyEvent.VK_ENTER, new
ResultAction(resultField, model)), new ButtonGridBagConstraints(4, 3, 1,
2));
            this.add(panel, BorderLayout.CENTER);

        }
    }
```

Třídu rámce (okna) oddědíme od třídy JFrame. V jejím konstruktoru pak nejprve vytvoříme model kalkulačky a poté zavoláme metodu *createAndShowGUI()*, v níž vyskládáme celé rozhraní.

Metoda createAndShowGUI()

V metodě createAndShowGUI() musíme kromě vytváření samotného rozhraní (*createMenu()* a *createCalculator()*) také udělat několik *povinných* operací. Nejprve okno pojmenu-

jeme, aby uživatel viděl, kterou aplikaci používá a onu samotnému nastavíme správce rozmístění – *BorderLayout*. Dále nastavíme, že po stisknutí zavíracího tlačítka (obvykle křížek v pravém horním rohu) dojde k vypnutí příslušné **JVM** a tím pádem i k terminaci programu.

Poté programu přiřadíme i ikonku – což zajistíme voláním *setIconImage()* jež přijímá vstupní proud (více v *18. dílu* tohoto seriálu). Pro vytvoření proudu zavoláme metodu *read()* pomocné třídy *ImageIO*, která obsahuje mnoho užitečných metod pro manipulaci s obrázky. Zde použijeme menší trik a načteme obrázek, který je umístěn na *classpath* ve stejném balíčku jako je námi vytvářená třída *JCalculatorFrame*. V obecném případě nelze doporučit umísťování obrázků a obecně jakýchkoliv zdrojů na *classpath*, ale pokud se jedná pouze o jeden (málo) zdrojů, tak to může zjednodušit distribuci programu (můžeme jej distribuovat v binární podobě jako jeden soubor).

Poslední operací v rámci metody *createAndShowGUI()* je volání *adjustAndDisplayWindow()*, které zajistí správné umístění okna a jeho zobrazení uživateli.

Metoda adjustAndDisplayWindow()

Metoda *adjustAndDisplayWindow()* nejprve nastaví oknu neměnnou velikost (zvětšování kalkulačky nedává příliš velkou logiku a bylo by i poměrně ošklivé). Poté nechá systém přepočítat a nastavit velikost obsažených komponent voláním *pack()*. Při ponechání výchozího nastavení by se okno zobrazilo v levém horním rohu, což je z uživatelského hlediska nešikovné. Proto za pomoci třídy *Toolkit*, jež poskytuje informace o displayi, okno přemístíme na prostředek obrazovky (střed na střed). Nyní již nám zbývá pouze okno zobrazit.

Metoda createMenu()

Vraťme se nyní zhruba doprostřed metody *createAndShowGUI()*, zde voláme dvě metody, které vytvářejí hlavní část grafického rozhraní. První z nich – metoda *createMenu()* – jak již název napovídá, konstruuje menu aplikace. Ve *Swingu* menu vytváříme pomocí tří tříd – třídy proužku menu (*JMenuBar*), třídy nabídky (*JMenu*) a konečně třídy položky (*JMenuItem*). V našem případě máme poměrně snadnou pozici, protože menu bude mít jedinou položku **Exit**, která bude sloužit k opuštění aplikace.

Položce *Exit* proto přiřadíme *ActionListener*, který při svém vykonání zničí okno kalkulačky, čímž programu umožní terminovat (program terminuje v okamžiku, kdy již neobsahuje žádné běžící vlákno (jež není *daemon*) ani okno). Jelikož chceme, aby aplikace šla zavřít i notoricky známou zkratkou **alt+Q**, tak přiřadíme položce i klávesovou zkratku (*accelerator*).

Nakonec do sebe celé menu řádně zanoříme a umístíme jej do severní části okna (vzpomeňme si, že jsme oknu nastavili *BorderLayout* v metodě *createAndShowGUI()*).

Metoda createCalculator()

Nyní se konečně dostáváme k metodě *createCalculator()*, která kreslí jádro naší aplikace – kalkukačku samotnou.

Nejprve vytvoříme *JPanel*, do nějž budeme komponenty umísťovat. Tomuto panelu přiřadíme *GridBagLayout* jako správce rozmístění, jenž je pro daný účel nejvhodnější – tlačítka tvoří mřížku a některá z nich zabírají více řádků, display naopak zabírá pouze jeden řádek, ale jde přes všechny sloupce.

Display kalkulačky bude reprezentován textovým polem (*JTextField*), jež nebude editovatelné, bude mít bílé pozadí a text v něm bude zarovnán na jeho pravou stranu. Po přidání komponenty na správné místo mřížky můžeme konečně vytvořit výchozí operátor, který umístíme do modelu (doposud nám k jeho vytvoření chyběla právě komponenta reprezentující display).

Rozmístění samotných tlačítek je již pouze otrocká práce, kdy pro každé z nich musíme vytvořit novou instanci *JCalculatorButton*, jíž při konstrukci předáme popisek, kód asociovaného tlačítka klávesnice a akci, jež se po stisku daného tlačítka vykoná. Toto tlačítko pak umístíme do mřížky za využití námi vytvořené třídy *ButtonGridBagConstraints*. Po vytvoření a přidání všech tlačítek umístíme panel kalkulačky na střed *BoderLayoutu* okna aplikace.

18.26.5. Calculator

Poslední částí dnešního příkladu je vstupní bod aplikace – třída *Calculator*. Tento kód jsme již v rámci tohoto seriálu několikrát v různých obměnách viděli a vždy dělal to samé – asynchronně inicializoval grafické rozhraní aplikace.

```
/**
 * Java pro zacatecniky 25 - kalkulacka
 * @author Pavel Micka
 */
public class Calculator {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {

            @Override
            public void run() {
                new JCalculatorFrame(); //referenci drzi interne JVM
            }
        });
    }
}
```

19. Jazyk C++

V naší poslední kapitole se budeme věnovat malému naťuknutí jednoho z nejuniversálnějších programovacích jazyků současnosti, konkrétně tedy jazyku C++. Jde o jazyk, ve kterém je možné programovat jak strukturovaně, tak objektově a také využívat šablon. Další velkou výhodou je, že jazyk C++ umí využít v jednom programu jak strukturovaný přístup, tak objekty. Další velkou a nespornou výhodou jazyka C++ je jeho universálnost, umožňuje psát opravdu pro všechny platformy a rozhraní, od Windows Desktop, přes Linux, Android, OS X, Windows Phone, mikropočítače až po Windows Metro. Zde bych chtěl zdůraznit, že jde opravdu pouze o naťuknutí, protože sám neovládám všechny možnosti, které jazyk nabízí, píši v něm relativně krádkou dobu, ale hlavně účelem je, dát vám do ruky nástroj, který umožní napsat jednoduché zlepšováky na školní či jiné úlohy. Na rozdíl od Javy zůstaneme v tomto případě u konzolových apliací.

C++ začíná, podobně jako C. svůj život v Bell Labs, kde začátkem 80tých let vyvinul tento jazyk Bjarne Stroustrup. Podle jeho vlastních slov, „C++ byl původně navržen tak, abychom (mí přátelé a já) nemuseli programovat v assembleru nebo jiných moderních vyšších jazycích. Jeho hlavním účelem bylo udělat pro jednotlivého programátora jednodušší a příjemnější psaní dobrých programů“ (Bjarne Stroustrup, Tfie C++ Programming Lan- guage. Třetí vydání. Záznam MA: Addison-Wesley Publishing Company, 1997). Stroustrup se více soustředil na vytvoření užitečného jazyka, než na prosazení určitých programovacích postupů nebo stylů. V určení jazykových rysů jsou skutečné programovací potřeby důležitější než teoretická čistota. Stroustrup založil C++ na stručnosti C, jeho vhodnosti pro systémové programování, široce rozšířené dostupnosti a jeho úzké vazby na operační systém UNIX. Hledisko OOP v C++ bylo inspirováno počítačovým simulačním jazykem, který se nazýval Simula67. Stroustrup přidal do C rysy OOP, aniž by významně změnil složku C. C++ je tedy nadstavba C, znamenající, že každý platný program v C je také platným programem v C++ (Neplatí zcela, dle posledních standardů ISO, upozorníme v dalším textu). Existuje několik méně důležitých nesrovnalostí, ale nic rozhodujícího. Programy v C++ mohou používat softwarové knihovny C. Knihovny jsou kolekce programových modulů, které můžete volat z programu. Poskytují prověřená řešení mnoha programovacích problémů, proto vám šetří mnoho času a úsilí. To pomohlo rozšíření C++. Jméno C++ pochází z přírůstkového operátoru ++, který přičítá k hodnotě proměnné 1. Jméno C++ přesně připomíná rozšířenou verzi C. Zatímco stránka OOP dává jazyku C++ schopnost uvést do spojitosti představy, které jsou zahrnuty v problému, C část C++ dává jazyku schopnost, aby se dostal blíže k hardwaru. Toto spojení schopností rozšířilo C++. Možná také zahrne duševní posun v rychlosti, když přejdete z jednoho pohledu na program na jiný. Tedy, protože C++ roubuje OOP na C, můžete ignorovat objektově orientované rysy C++. Ale mnoho zmeškáte, když to bude všechno, co uděláte. Jakmile C++ dosáhl jistého úspěchu, Stroustrup dodal šablony, umožňující programování pomocí šablon. Ale až teprve tehdy, když se začaly používat a rozšiřovat, začalo být zřejmé, že snad byly tak důležitým dodatkem jako OOP, dokonce, jak by někteří tvrdili, důležitějším. Skutečnost, že C++ zahrnuje jak OOP, tak programování pomocí šablon, ukazuje, že C++ zdůrazňuje užitkovost před ideovým přístupem, což je jedním z důvodů úspěchu jazyka.

19.1. Přenositelnost a standardy

Napsali jste v práci praktický program C++ pro Starší 286 PC AT, když se vedení rozhodlo nahradit tento stroj pracovní stanicí Sun, počítačem, který používá jiný procesor a jiný operační systém. Můžete spustit svůj program na nové platformě? Samozřejmě musíte překompilovat program za použití kompilátoru, který je navržen pro novou platformu. Ale budete muset udělat nějaké změny do kódu, který jste napsali. Pokud můžete přeložit program bez

provedení změn, a ten běží bez nesnází, říkáme, že je program přenositelný. Pro přenositelnost existuje množství překážek, první z nich je hardware. Program, který je hardwarově závislý, pravděpodobně nebude přenositelný. Ten, kdo chce například přímo ovládat video kartu IBM PC VGA, tak si nebude vědět rady, pokud se to bude týkat Sunu. Druhou překážkou je jazyková odlišnost. Zajisté může nastat problém s mluveným jazykem. Popis denních událostí člověkem z Yorkshiru nebude přenositelný do Brooklynu, i když se v obou oblastech mluví anglicky. Počítačové jazyky také mohou rozvíjet dialekty. Je implementace C++ na IBM PC stejná jako implementace na Sunu? Ačkoli by většina implementátoni ráda vytvořila své verze C++ kompatibilní s jinými, je to obtížné udělat bez zveřejnění standardů, které přesně popisují, jak jazyk pracuje. Proto Americký národní institut pro standardy (ANSI) vytvořil v roce 1990 výbor (ANSI X3J16), aby pro C++ vyvinul standard. (ANSI již vyvinulo standardy pro C.)

Mezinárodní organizace pro standardy (ISO) brzy spojila tento proces se svým vlastním výborem (ISO-WG-21), takže na rozvoj standardů C++ existuje spojené úsilí ANSI/ISO. Tyto výbory se setkávají společně třikrát do roka a my je jednoduše celosvětově spojujeme do výboru ANS1/ISO. Rozhodnutí ANSI ISO vytvořit standardy zdůrazňuje, že C++ je důležitým a široce rozšířeným jazykem. Také to indikuje, že C++ dosáhl jisté úrovně dospělosti, navzdory tomu, že není produktivní zavádět standardy, když se jazyk tak rychle vyvíjí. Přesto C++ prodělal významné změny od té doby, co výbor začal svou práci. Práce na standardech ANSI ISO C++ začala v roce 1990. Výbor vydal v následujících letech několik prozatímních pracovních podkladů. V roce 1995 koncept výboru (CD-Committee Draft) k veřejnému připomínkování. V prosinci 1996 uvolnil druhou verzi (CD2) pro další veřejné posouzení. Tyto dokumenty nejen vypilovaly popis existujících rysů C++, ale také rozšířily jazyk o výjimky, ŘITI, šablony a standardní knihovnu šablon (STL – Standard Template Library). Na prozatímní pracovní podklady a koncepty výboru se často odkazovalo jako na „standardní koncept" („standard draft“), ale první dokument, který dal právo k tomuto označení, je Final Draft International Standard (FDIS) vydaný v listopadu 1997. V době psaní této knihy se očekává, že má být v březnu 1998 schválen konečný Mezinárodní Standard (IS – International Standard). Tato kniha je primárně založena na CD2, který je velmi blízký FDIS. Rozdíly mezi FDIS a IS by mohly naproti tomu být v povaze korekcí, nikoli v dodatcích o změně rysu. Standard ANSI ISO C++ dodatečně dotahuje standard C, protože se předpokládá, dokud je to možné, že C++ je nadstavbou C. To znamená, jakýkoli platný program C by měl být platným programem C++. Mezi ANSI C a odpovídajícími pravidly C++ existuje několik rozdílů, ale jsou méně důležité. ANSI C zahrnuje některé rysy, které byly nejprve zavedeny do C++, jako jsou vytváření prototypu funkcí a kvalifikátor typu const. Standard ANSI C nejen že definuje jazyk C, ale také definuje standardní knihovnu C, kterou musí implementace ANSI C podporovat. C++ tuto knihovnu také používá; tato kniha se na ni bude odkazovat jako na standardní knihovnu. Navíc výbor ANSI/ISO C++ poskytl standardní knihovnu tříd C++. Než začal výbor ANSI ISO pracovat, mnoho lidí přijalo nejnovější verzi standardu C++ od Bell Labs. Například kompilátor se sám mohl charakterizovat jako kompatibilní s vydáním C++ 2.0 nelx> 3 0.

19.2. Postup psaní programu v C++

Pokud si ve Visual studiu vytvoříme projekt a v něm soubor s příponou cpp, tak se můžeme dát do vlastního psaní. Krom norem je potřeba myslet na několik nepsaných pravidel, které usnadní orientaci v programu a předcházejí vzniku chyb.

První řádky by měly obsahovat komentář s názvem programu, jménem autora, verzí programu a stručným popisem (výstup, vstup, stručně činnost, opravdu na 1 až 5 řádků). Úvodní komentář je následován direktivou include, která připojí iostream, fstream a hlavičkové soubory. Za direktivou include následuje definování jmenných prostorů (v našich aplikacích vždy std). Po jmenných prostorech musíme definovat prototypy funkcí (pokud máme více

funkcí v kódu). Následuje definice globálních konstant a globálních proměnných. Pokud užíváme globální struktury, tak je definujeme hned za globálními proměnnými a konečně se můžeme pustit do hlavní funkce programu, tedy funkce main().

Funkce main() je hlavní funkcí programu, bez ní není možné spustit program a slouží jako spojení dílčích částí kódů. Tato funkce nemá žádné parametry, proto má za svým názvem prázdné kulaté závorky. Tělo této funkce, stejně jako všech ostatních, je uzavřeno ve složených závorkách. Tělo funkce main musí obsahovat příkaz výstupu do konzole, aby vůbec došlo k jejímu vykreslení, tyto příkazy máme v základu 2, cout a fprint. Posledním příkazem této funkce je příkaz return 0; sloužící operačnímu systému, nebo i vývojovému prostředí k rozpoznání, že program skončil svou činnost bez chyb. U jednoduchých kódů obsahuje funkce main celý kód (pokud nevyužívá rekurze), ale u složitějších programů obsahuje volání jednotlivých funkcí a udává strukturu celého programu, proto by měla být psána jako první funkce v programu.

Příkazy se píší každý na samostatný řádek a jsou ukončovány středníkem. Pokud je příkaz součástí cyklu (tedy je schován v těle jiného příkazu), tak je odsazen proti jeho nadřazenému příkazu (Visual studio odsazuje samo). Cykly, podmínky a další podobně stěžejní příkazy je dobré opatřit komentářem. Středníkem se **neukončuje** direktiva include, deklarace funkce, příkazy větvení a cyklů (mají místo středníku tělo ze složených závorek), krom příkazu do while, kdy se za podmínku u while středník píše.

```
1     #include <iostream>
2     using namespace std;
3     int main(void)
4     {
5         cout << "velikost datového typu short: " << sizeof(short) << "\n";
6         cout << "velikost datového typu int: " << sizeof(int) << "\n";
7         cout << "velikost datového typu long: " << sizeof(long) << "\n";
8         cout << "velikost datového typu float: " << sizeof(float) << "\n";
9         cout << "velikost datového typu double: " << sizeof(double) << "\n";
10        cout << "velikost datového typu long double: " << sizeof(long double) << "\n";
11        return 0;
12    }
```

Pro jednoduchost budeme mít v naší knize zlého skřítka, který bude uvedeným kódům pojídat úvodní komentář, takže uvedené kódy budou funkční, ale úvodní komentář si vždy budeme muset dopsat sami. Název programu bude nejčastěji uveden v textu. Ukázkový kód slouží ke zjištění velikostí jednotlivých typů proměnných přímo na našem překladači.

19.3. Jmenné prostory

Každé jméno objektu, proměnné atd. musí být jedinečné. Aby bylo možné jména opakovat, tak se rozdělují pomocí jmenných prostorů. Není to nic, co bychom neznali z běžného života, přirovnat se to dá k „systému“ jmen lidí, každé jméno je v rámci nějaké oblasti jedinečné, některé více, jiné méně. Ve světě programování musí být každé jméno plně jedinečné, když to převedu na lidský svět, tak název objektu je křestní jméno a název jmenného prostoru plní funkci příjmení.

Jmenný prosto máme možnost zapisovat pro každý objekt samostatně, pak tedy píšeme jmenný prostor, čtyřtečku a název objektu, tedy např. std::cout. Asi se ptáte, proč jsem toho nevyužil v uvedeném příkladu, protože jmenný prostor je povinnou součástí názvu. Odpověď je jednoduchá, užil jsem druhou možnost, která se vytváří pomocí příkazu using namespace, který říká, že všechna jména uvedená v programu budou z daného jmenného prostoru, pokud nebude pomocí první možnosti zápisu uvedeno jinak. Příklad zápisu, který je nejčastější je: using namespace std;. Tento příklad říká, že všechna jména uvedena v programu jsou jmenného prostoru standartní knihovny. Příkazem using namespace je možné předdefinovat více

jmenných prostorů, ale potom názvy objektů musí být jedinečné v rámci všech předdefinovaných jmenných prostorů, proto doporučuji definovat jmenný prostor std a v případě potřeby jeden vlastní, více ne. Pro naše ukázkové programy bude vždy užito jmenného prostoru std a mohu slíbit, že v žádném kódu nebudu mást tělem užitím přiřazení jmenného prostoru pomocí čtyřtečky.

19.4. Paměť

Když napíšeme program, tak při jeho spuštění je vždy požádáno OS o přidělení operační paměti, kam se ukládají zpracovaná data a také program jako takový. Aby bylo přidělení paměti co nejvíce efektivní a systém hned mohl rozpoznat možnost spuštění a zabránit kolapsu z nedostatku paměti, tak programátor při tvorbě kódu definuje množství paměti, potřebné k provedení algoritmu. Všechna data se ukládají na paměťová místa, do tzv. proměnných. Při spouštění systém přidělí konkrétní adresy paměti, ale aby věděl, jaké adresy mají mít jednotlivé proměnné, musí vědět, co v nich bude, čímž se dozví, jak velké konkrétní paměťové místo bude.

Programátoři mají k dispozici 3 základní typy proměnných, číselné, textové a logické (nejmenší). V každém datovém typu máme k dispozici různě velké varianty proměnných. Pozor, ti kdo si zkoušeli něco v php si speciálně musí dávat pozor na typ proměnné, C++ je podobně jako Java přísně typový jazyk.

19.4.1. Celočíselné datové typy

Celočíselné datové typy jsou určeny k uložení hodnot reprezentovaných celým číslem, jako např.: počet dokončených cyklů. Tyto hodnoty jsou kladné nebo i záporné. Pokud víme, že v proměnné bude jen kladné číslo (např.: počet zakoupených obědů), tak užijeme předznamenání unsignet, které řekne, že nejmenší číslo je nula, pokud zkusíme vložit mínus, tak dojde k ukončení programu chybou.

Přehled typů		
typ	Velikost v bajtech	Rozsah hodnot
short	2	-32768 až 32767
unsigned short	2	0 až 65535
int	4	-2147483648 až 2147483647
unsigned int	4	0 až 4294967295
long	4	-2147483648 až 2147483647
unsigned long	4	0 až 4294967295

19.4.2. Datový typ pro reálná čísla

Tato skupina datových typů je určená pro čísla vybavená plovoucí desetinnou čárkou. Desetinná část může být kdekoli v čísle, je jedno, kolik cifer je před a kolik za desetinnou čárkou. Reálná čísla v informatice neodpovídají reálným číslům v matematice!! Každopádně tyto datové typy musí ukládat výsledky dělení, odmocnění a podobně.

Přehled datových typů		
Datový typ	Velikost v bajtech	Rozsah
float	4	3,4E-38 až 3,4E38
double	8	1,7E-308 až 1,7E308
Long double	10	3,4E-4932 až 3,4E4932

19.4.3. Textové datové typy

Rozpoznáváme dvojici základních typů. prvním je char, který umožní uložit jeden libovolný znak z klávesnice nebo ascii tabulky. Druhým je string, který umožňuje uložit posloupnost znaků a je součástí knihovny základní knihovny. String je přidán do C++, v jazyce

C se museli programátoři spokojit s polem typu char. Pokud potřebujeme konkrétní znak z datového typu string je možné jej získat.

19.4.4. Logické datové typy

Logický datový typ máme jeden, konkrétně jde o typ bool. Tento datový typ je nejstriktnějším datovým typem, může obsahovat pouze dvojici hodnot, a to true a false. Tohoto datového typu užíváme při podmíněném větvení, podmíněném ukončení cyklu atd. ve vyšších jazycích jako je C++ není příliš užíván, více je rozšířen u programování PLC.

Upozorňuji, že nad tímto datovým typem není možné provádět matematické operace, tento datový typ odporuje jen operace logické, tedy and, nand, or, nor, xor, xnor a to jen za předpokladu, že obě proměnné jsou typu bool a výsledek je přiřazen rovněž do proměnné typu bool.

19.4.5. Deklarace proměnných

Deklarace proměnné je její zrození. Každou proměnnou je nutné deklarovat před jejím použitím, ale již při deklaraci ji může použít tím, že do ni přiřadíme hodnotu. Deklarace proměnné je dvojího typu, buď globální, deklaruje se pod příkazem using namespace či direktivou include. Druhým způsobem je deklarace uvnitř funkce, taková proměnná je pak dostupná pouze z funkce, kde byla deklarována, z jiných míst programu není dostupná.

Proměnnou deklarujeme velmi lehce, uvedeme datový typ a její jméno, tak třeba např.: int obedu;. Pokud deklarujeme více proměnných stejného datového typu, tak je můžeme deklarovat jedním jediným příkazem, pak jejich jména oddělujeme čárkou, takže např.: int snidani, obedu, veceri;. Pokud ovšem není datový typ pro všechny zamýšlené proměnné stejný, tak je musíme deklarovat po skupinách dle datového typu, nebo jednotlivě.

Jméno proměnné musí vždy začínat písmenem (velkým či malým, tady je to na rozdíl od Javy na vůli programátora), nebo je možné na první místo dát podtržítko. Název proměnné může obsahovat čísla, ale číslice nesmí být prvním znakem jména proměnné. Název nesmí obsahovat mezeru, tečku, čárku, vykřičník otazník a další interpunkční znaménka!! Proměnné je možné pojmenovávat s diakritikou, ale ne všechny překladače to zvládají bez problému, a hlavně překladače neodliší od sebe c a č apod., všeobecně doporučuji pojmenovávat proměnné bez diakritiky a speciálních znaků. Jména proměnných se musí vyhýbat chráněným slovům jazyka, nesmí být stejné jméno proměnné jako příkaz! A jméno proměnné by mělo být jedinečné v celém programu. Pokud je proměnná deklarována uvnitř funkce, tak její jméno musí být jedinečné v rámci dané funkce, ale lépe v celém programu. Doporučuji pojmenovávat proměnné tak, aby bylo jasné, co obsahují, název by měl vystihovat jejich funkci. Je jedno nepsané pravidlo, proměnná jménem ***i*** se používá jako pomocná proměnná pro počítání a další podmínky v cyklech. Také doporučuji, aby název proměnné začínal malým písmenem, je to další nepsaný zvyk, který usnadňuje programátorům orientaci ve více jazycích.

19.4.6. Přiřazení hodnoty do proměnné

Hodnotu do proměnné můžeme přiřadit pomocí operátoru = přímo v kódu a to tak, že zadáme název proměnné = hodnota;, takže např.: vyska = 175;. Další možností je, že k přiřazení hodnoty do proměnné dojde vyhodnocením nějakého výrazu, takže např.: o = a+b;. Poslední, hojně využívanou metodou je načtení hodnoty z klávesnice, k čemuž slouží objekt **cin** a operátor ≫. Objekt cin je součástí takzvaného iostream, který je nutný připojit direktivitou include. Objekt cin čte do proměnné po první bílý znak, tedy mezeru či enter. Pokud chceme, aby jedním příkazem došlo k načtení hodnot do více proměnných, tak uživatel musí tyto hodnoty oddělit mezerou a my musíme jednotlivé proměnné spojit operátorem ≫. Myslím si, že je na čase si ukázat, jak takové načtení objektem cin vypadá na příkladu: cin ≫ vyska;

Speciálním případem je, když proměnnou definujeme a hned v definici ji přiřadíme nějakou hodnotu. Takovému to případu se inicializace a provádí se např.: int vyska = 175; Pokud máme kdykoli v kódu přiřazenou hodnotu přímo do proměnné, tak je nutné, aby u proměnných pro reálná čísla byla čísla v kódu zadána s desetinnou tečkou, uživatel, pokud zadává pomocí cin, se řídí nastavením svého OS.

19.4.7. Zobrazení hodnoty proměnné na konzoli

Výsledek výpočtu uložený v proměnný je krásná věc, ale bez zobrazení uživateli je úplně marný. Aby program mohl uživateli sdělit, co má zadat do proměnné, zobrazit doprovodné texty, a hlavně obsah námi zvolených proměnných, tak je potřeba mít nějaký výstup na monitor. V našem případě bude výstup probíhat na konzoli, což je podobné okno jako má příkazový řádek. Pro zobrazení slouží objekt **cout** a operátor **≪**, nebo funkce **fprint()**, která má v závorkách uvedený textový řetězec a názvy proměnných.

Objekt cout je obdobně jako cin součástí iostream. Pokud chceme pomocí tohoto objektu zobrazit nějaký konkrétní text, tak se to provede tak, že užijeme objekt cout a operátor ≪ za který napíšeme uvozovky a do nich vložíme text, který chceme uživateli zobrazit. Proměnnou zobrazíme ještě jednodušeji, stačí nám objekt cout, operátor ≪ a jméno proměnné. Pokud chceme zobrazit nějaký text a za něj obsah proměnné, tak to provedeme tak, že za uvozovky obsahující tex vložíme znovu operátor ≪ a napíšeme název proměnné. Pojďme si to všechno ukázat postupně na příkladech. Zobrazení textu by mohlo vypadat takto: cout ≪ „Zadej počet dosažených bodů: „; Zobrazení obsahu proměnné třeba takto: cout ≪ vyska; a zobrazení kombinace textu a proměnné: cout ≪ „Vaše výška je: „ ≪ vyska; nebo cout ≪ jmeno ≪ „ je vysoký „ ≪ vyska;

Aby se nám vše nepsalo do nekonečně dlouhého nepřehledného řádku, je potřeba někdy také oddělit řádky jejich zalomením, podobně jako když ve Wordu zadáme enter. V C++ k tomu slouží dost obdobně jako v Javě esc sekvence, která je konkrétně pro náš případ \n a zadává se přímo v textu bez jakýchkoli mezer. Pokud potřebujeme zalomit za obsahem proměnné, tak musíme přidat textový řetězec s \n nebo dát samostatný příkaz s tímto řetězcem. Další základní možnosti při vypisování pomocí esc sekvence jsou \t což je tabulátor, \\ což je zobrazení znaku \, \‘ což je zobrazení apostrofu a \“ což zobrazí uvozovky.

Pokud zobrazujeme pomocí funkce fprint(), tak je v závorkách v uvozovkách uveden text a formátovací masky stejné jako v Javě, za uvozovkami následují čárkami oddělené proměnné, jejichž obsah je zobrazován. Osobně preferuji objekt cout, takže se s tímto výpisem setkáme až v pozdějších příkladech, kde funkci formátovacích masek poznáte z proměnných, takže je zbytečné se zde o formátovacích maskách více rozepisovat, jen pro připomenutí z minulé kapitoly, formátovací maska začíná znakem %.

Nyní si ukážeme jeden příklad, na kterém jsou vidět užití vstupů a výstupů.

```
#include <iostream>
#include <string>
 using namespace std;
int main(void)
{
    int vaha, vyska;
    string jmeno;
    cout << "Zadejte sve jmeno: ";
    cin >> jmeno;
    cout >> "Zadejte vahu v kilogramech: ";
    cin >> vaha;
    cout << "Zadejte vyšku v cm: ";
    cin >> vyska;
    cout << "Jmenujete se " << jmeno << "\n";
    cout << "Vážíte " << vaha << "kg a měříte " << vyska << "cm.";
    return 0;
}
```

Žlutá čára před příkazy znamená, že kód není uložen a byl přímo vytvořen, nikoli otevřen z uloženého souboru bez jakýchkoli změn.

Dalším příkladem může být:

```
#include <iostream>
using namespace std;
int den, mesic, rok, body, konec;
int main(void)
{
        cout << "Zadejte cisli den, mesic a rok v tomto poradi a cisla oddelte
mezerou. ";
        cin >> den >> mesic >> rok;
        cout << "Zadejte pocet spravnych odpovedi: ";
        cin >> body;
        cout << "\n";
        cout << "Dne: "<<den<<"."<<mesic<<"."<<rok<<" jste dosahli: " << body<<
" bodu. \n";
        cout << "\n";
        cout << "Pro ukoncení programu zadejte cislo 2. ";
        cin >> konec;
        return 0;
}
```

19.4.8. Přetečení a podtečení

Programátory v souvislosti s proměnnými trápí jedna velká skutečnost, kterou je limit nejvyšší hodnoty, kterou lze do proměnné přiřadit. O velikosti jednotlivých proměnných jsme se bavili již u datových typů, jenže co se vlastně stane s obsahem proměnné, když se pokusíme přiřadit hodnotu, která je vyšší nežli maximum, které je schopná proměnná daného typu pojmout? Odpověď mám poskytnou ručičkové hodiny, tedy pro ideální případ. Prostě se uloží hodnota naprosto nesmyslná, i když ne zcela. Dojde k tomu, že se přiřazovaná hodnota poníží o celistvý násobek maximálního rozsahu a zbytek se přiřadí. Teorie je hezká, ale v praxi se zobrazují opravdu nahodile nesmyslné hodnoty. Tomuto stavu se u se říká přetečení.

Podtečení je to samé jako přetečení, ale nastává, když je hodnota menší nežli nejmenší možná, tedy je zápornější, nežli je přípustné záporné maximum. Oba tyto stavy jsou nežádoucí, a hlavně u reálných datových typů si musíme dávat pozor, aby třeba výsledek dělení nebyl příliš malý. V programech, které budeme postupně společně psát, tyto stavy nehrozí, nicméně je nutné již při vymýšlení algoritmu s těmito stavy počítat a využívat třeba exponenciálního

zápisu pomocí dvojice proměnných, nebo hodnotu rozložit do více proměnných v poli, či ji převést na řetězec.

19.5. Aritmetické operátory

Aritmetické operátory realizují matematické operace tak, jak jsme zvyklí z matematiky, a to včetně předností operací. Většina základních aritmetických operátorů, kromě odmocniny, mocniny a modula je binárních, tedy vyžadují dva operandy a přiřazení. Všechny aritmetické operátory mají větší prioritu nežli operátor přiřazení, což znamená, že se nejdříve vyhodnotí výraz a následně se výsledek přiřadí.

Jen jedna malá odbočka, pokud potřebujeme stanovit podmínku, že se něco musí rovnat něčemu, tak klasické rovná se se v jazyce C++ značí obdobně jako v Javě symbolem == .

19.5.1. Operátor sčítání

Operátor sčítání je obdobně jako v matematice +. Jediné, na co si musíme dávat pozor, je, aby nedošlo k přetečení. Ještě není nejlepší přiřazovat výsledek do unsignet proměnné, aby nedošlo ke kolizi, protože i sčítání může vrátit záporný výsledek, pokud je jedno z čísel hodně záporné. Specialitou tohoto operátoru je možnost jeho použití na řetězce. Pokud sečtu dvojici řetězců, tak druhý řetězec se připojí na konec prvního.

19.5.2. Operátor odčítání

Tento operátor je opět takový, jaký jej známe z matematiky. Na rozdíl od sčítání se nedá použít na řetězce a také se musí dávat pozor na podtečení.

19.5.3. Priority operátorů

Obdobně jako v naší hodně oblíbené matematice ovlivňují pořadí prováděných úkonů ve výrazu priority operátorů. Nyní si vypíšeme priority od nejvyšší po nejnižší, nevyšší je záporná hodnota, násobení i dělení a modulu, sčítání a odčítání, přiřazení.

19.5.4. Operátor dělení

Pokud je dělenec i dělitel celočíselného datového typu, tak je provedeno celočíselné dělení, pokud ovšem je alespoň jeden z operandů reálného datového typu, tak je provedeno klasické dělení. Je nutné si uvědomit jednu věc, pokud je výsledek dělení přiřazen do celočíselné datové proměnné, tak nedojde k zaokrouhlení výsledku, ale k odříznutí desetinné části výsledku. Dělení nulou není definováno, takže pokud na něj narazí překladač, tak vrátí chybu a pokud vznikne zadáním nuly od uživatele, tak dojde k ukončení programu a operačnímu systému je vrácen chybový kód.

19.5.5. Operátor modulo

Modulo zní každý i ten, kdo nečetl kapitolu o Javě již z nějaké druhé či třetí třídy základní školy. Modulo je zbytek, který zůstane po celočíselném dělení, ale opravdu jen ta zbytková část výsledku. Obdobně jako v jazyce Java jej označujeme symbolem % a je definován pouze nad celočíselnými datovými typy.

19.5.6. Mocnina

Na umocňování neexistuje žádný operátor. Umocňujeme pomocí funkce pow, která má dvojici parametrů, základ a exponent. Tato funkce je součástí knihovny cmath, kterou je nutné připojit pomocí direktivity include. Funkce pow vrací vždy výsledek datového typu double. Na závěr povídání o mocnění si jen ukážeme jeden příklad, takže pokud napíší pow(2,3); tak mám hodnotu 2^3.

19.5.7. Zkrácená forma zápisu

Pokud upravujeme hodnotu proměnné, tedy provádíme příkaz typu i = i +1, tak můžeme využít zkrácené formy zápisu. V následující tabulce si ukážeme, jak jednotlivé zkrácené formy vypadají.

Plná forma	Zkrácená forma
I = i + 1	I++
I = i - 1	i--
I = i + 2	I +=2
I = i - 2	I -= 2
I = i / 2	I /= 2
I = i * 2	I *= 2
I = i % 2	I %= 2

19.5.8. Příklad na výpočet školného

Nyní máme za sebou opět kus teorie a to ne malý. Jak už to v pořádné učebnici ICT musí být, tak následuje příklad, na kterém si vše zopakujeme.

```
1   #include <iostream>
2   using namespace std;
3   int registrovano, pridano, zruseno;
4   char konec;
5   float prumerneSkolne;
6   int main(void)
7   {
8       cout << "Program na evidenci studentu "<<"\n";
9       cout << "Zadejte pocet registrovanych studentu: ";
10      cin >> registrovano;
11      cout << "Zadejte pocet studentu, kteri se registrovali na prednasku dodatecne: ";
12      cin >> pridano;
13      registrovano += pridano; // je totéž jako: registrovano = registrovano + pridano
14      cout << "Zadejte pocet studentu, kteri se odhlasili: ";
15      cin >> zruseno;
16      registrovano -= zruseno;
17      cout << "Celkem registrovanych studentu: " << registrovano << "\n";
18      cout << "Skolne celkem: " << (registrovano + zruseno) * 72 << " Kc \n";
19      prumerneSkolne = (float)((registrovano + zruseno) * 72) / registrovano;
20      cout << "Prumerne skolne na zapsaneho studenta: " << prumerneSkolne;
21      cout << "\n" << "Pro ukonceni programu zadejte Y: ";
22      cin >> konec;
23  }
```

19.6. Píšeme román, aneb větvení pomocí if

Příkaz if slouží k tomu, aby se kód mohl rozdělit do dvou větví, nebo aby bylo možné oddělit úsek kódu, který se má provést jen v případě, že je splněna nějaká podmínka. Za klíčovém slovem if následuje v kulatých závorkách uvedená podmínka, takže zápis může vypadat například takto: *if(počet_piv == 10)* . Jako podmínka je většinou reálný výraz, již méně časté, ale také správné je uvedení proměnné typu bool do kulaté závorky, čímž je nahrazena podmínka, protože aby byla podmínka splněna, tak je potřeba mít hodnotu v bool nastavenou na true.

V základní definici je možné za if, mimochodem bez středníku za uzavírací závorkou podmínky, uvést jeden jediný příkaz, pokud je potřeba podmíněně plnit více příkazů, je nutné je uzavřít do složených závorek. Pokud využijete služeb našeptávání ve Visual Studiu 2013, tak se dostanete na to, složené závorky jsou doplněny implicitně.

Pokud potřebujeme opravdu dvě větve kódu, tak je potřeba za složené závorky, které obepínají posloupnost podmíněných příkazů, umístit příkaz else, za kterým opět bez středníku následují složené závorky, kam přijde větev, která se má provést při nesplnění podmínky, která je vyhodnocena v if. Pokud je potřeba vyšetřit postupně více podmínek, tak můžeme použít kombinovaného příkazu else if(), kde pak následuje opět další podmíněný příkaz.

Pro jednoduché příkazové konstrukce lze příkaz if nahradit ternálmím výrazem, který má syntaxi: podmínka ? příkaz pro splněnou podmínku : příkaz pro nesplnění podmínky;

Opět si z guláše udělejme pořádek pomocí jednoho jednoduchého příkladu, kdy zjistíme, zda je číslo sudé.

```
1   #include<iostream>;
2   using namespace std;
3   int cislo;
4   int main(void)
5   {
6       cout << "Program na zjisteni sudosti cisel \n";
7       cout << "Zadejte cislo: ";
8       cin >> cislo;
9       cout << "\n";
10      if (cislo % 2 == 0)
11          cout << "Cislo " << cislo << " je sude.";
12      else
13          cout << "Cislo " << cislo << " je liche.";
14      return 0;
15  }
```

19.7. Vícepolohový přepínač swith

Tento příkaz dělí kód na n větví, kdy n je rovno 2 až nekonečno. Rozhodování mezi jednotlivými větvemi probíhá na základě obsahu řídící proměnné, kdy se její obsah porovnává s možnostmi pro jednotlivé větve kódu. Toto rozhodování je definováno pro čísla a znaky. Znaky se porovnávají na základě srovnání jejich kódů v ascii tabulce.

Každou větev kódu je potřeba zakončit příkazem break;, jinak dojde k tomu, že budou postupně vyhodnocovány i další větve, dokud se nenarazí na příkaz break nebo na konec příkazu swith.

Konstanta, na jejímž základě se rozhodne o prováděné větvi kódu, musí být vždy celočíselná nebo jeden znak, nesmí být zadán číselný rozsah! Syntaxi příkazu si ukážeme rovnou na konkrétním příkladu, protože je relativně složitá a opět může dost napomoci vysual studio se svým našeptáváním.

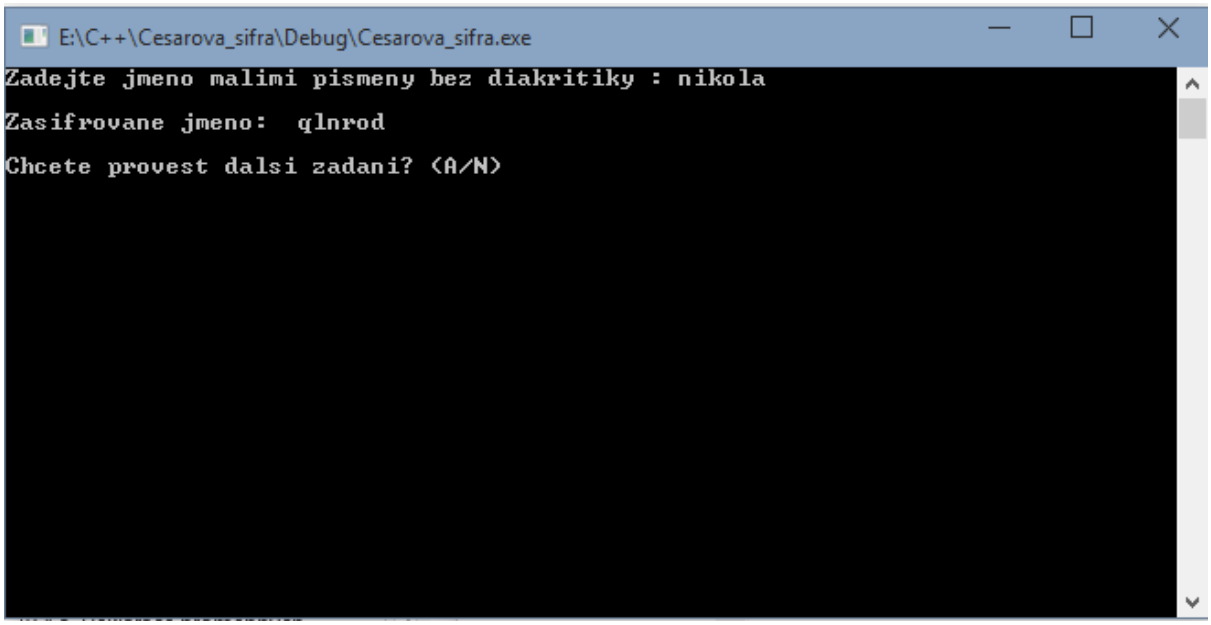
```
#include<iostream>;
using namespace std;
int znamka;
int main(void)
{
    cout << "Zadejte znamku: ";
    cin >> znamka;
    switch (znamka)
    {
    case '1':
        cout << "Pocet ziskanych bodu: 90 az 100.\n";
        break;
    case '2':
        cout << "Pocet ziskanych bodu: 80 az 89.\n";
        break;
    case '3':
        cout << "Poct ziskanych bodu: 70 az 79.\n";
        break;
    case '4':
        cout << "Pocet ziskanych bodu: 65 az 69.\n";
        break;
    default:
        cout << "Neuspel, mene jak 65 bodu.";
        break;
    }
    return 0;
}
```

Nebo jako příklad 2 zde uvádím nástroj pro tvorbu šifrovaných vzkazů pomocí Caesarovy šifry.

```cpp
#include <iostream>
using namespace std;
const int MAX = 80;
int main()
{
    bool konec;
    do
    {
        char vysledky[MAX];
        char volba;
        cout << "Zadejte jmeno malimi pismeny bez diakritiky " << ": ";
        cin >> vysledky;
        char pomoc;
        for (int i = 0; i < MAX; i++)
        {
            pomoc = vysledky[i];
            switch (pomoc)
            {
            case 'a':
                vysledky[i] = 'd';
                break;
            case 'b':
                vysledky[i] = 'e';
                break;
            case 'c':
                vysledky[i] = 'f';
                break;
            case 'd':
                vysledky[i] = 'g';
                break;
            case 'e':
                vysledky[i] = 'h';
                break;
            case 'f':
                vysledky[i] = 'i';
                break;
```

```
            case 'g':
                vysledky[i] = 'j';
                break;
            case 'h':
                vysledky[i] = 'k';
                break;
            case 'i':
                vysledky[i] = 'l';
                break;
            case 'j':
                vysledky[i] = 'm';
                break;
            case 'k':
                vysledky[i] = 'n';
                break;
            case 'l':
                vysledky[i] = 'o';
                break;
            case 'm':
                vysledky[i] = 'p';
                break;
            case 'n':
                vysledky[i] = 'q';
                break;
            case 'o':
                vysledky[i] = 'r';
                break;
            case 'p':
                vysledky[i] = 's';
                break;
            case 'q':
                vysledky[i] = 't';
                break;
            case 'r':
                vysledky[i] = 'u';
                break;
```

```
73              case 's':
74                  vysledky[i] = 'v';
75                  break;
76              case 't':
77                  vysledky[i] = 'w';
78                  break;
79              case 'u':
80                  vysledky[i] = 'x';
81                  break;
82              case 'v':
83                  vysledky[i] = 'y';
84                  break;
85              case 'w':
86                  vysledky[i] = 'z';
87                  break;
88              case 'x':
89                  vysledky[i] = 'a';
90                  break;
91              case 'y':
92                  vysledky[i] = 'b';
93                  break;
94              case 'z':
95                  vysledky[i] = 'c';
96                  break;
97              }
98          }
99          cout << "\n";
100         cout << "Zasifrovane jmeno:  " << vysledky << "\n";
101         cout << "\n";
102         cout << "Chcete provest dalsi zadani? (A/N) ";
103         cin >> volba;
104         if (volba != 'A')
105             konec = false;
106         cout << "\n";
107     } while (konec);
108     return 0;
109 }
```



19.8. Vnořené podmínky

Obdobně jako jsme si uváděli možnost konstrukce else if, tak lze zařadit další podmínku i do kladné větve. Takto podmínka se uvádí normálně do těla a je vyhodnocena při splnění vnější podmínky. Umístit podmínku do těla můžeme i do záporné větve. Zde je nutné přemýšlet, protože pokud do kladné větve vložíme další if, tak se nám původní elese přiřadí k němu. Abychom takovému to karambolu zabránili, tak v kladné větvi užíváme úplné větvení, kdy do vnořeného elese vložíme na samostatný řádek středník, což je dle syntaxe prázdný příkaz, nicméně původní else nám zůstane spojeno s if, kde jej chceme.

19.9. Logické operátory, aneb slučuje podmínky

Postupně se dopracujeme do stádia, kdy potřebujeme provést nějaký kus kódu, jen když platí 2 nebo 36 podmínky zároveň, nebo když platí jedna ze dvou podmínek a podobně. V takovém případě užijeme logických operátorů, se kterými se normálně pracuje nad boolovskými proměnnými. V úplném prazákladu máme k dispozici trojici logických operátorů:

&& slouží ke spojení podmínek, které mají platit zároveň, tady je pravdivý, pokud jsou obě podmínky pravdivé.

|| slouží ke spojení dvojice podmínek, kdy je pravdivý za předpokladu, že je pravdivá jedna, nebo obě podmínky.

! tento symbol značí negaci, tedy obrátí hodnotu podmínky, pokud se uvede před podmínku, která je pravdivá, tak její výsledek je false.

Nyní si uvedeme příklad, který se nesnažme pochopit celý, protože obsahuje konstrukce, na jejichž vysvětlení teprve dojde, ale všimněme si, jak se využívají logické operátory v takové triviální úloze jako je hledání největšího společného dělitele.

```
#include <iostream>
using namespace std;
int a, b, k;
void delitel ();
int main()
{
    cout << "Zadejte prvni cislo: ";
    cin >> a;
    cout << "\n";
    cout << "Zadejte druhe cislo: ";
    cin >> b;
    delitel();
    cout << "\nNejvetsi spolecny delitel je:
    cout << "\nPro ukonceni zadejte libovoln
    cin >> k;
    return 0;
}
void delitel()
{
    if ((a <= 0) || (b <= 0))
    {
        cout << "Chybne zadani";
    }
    while (a>0)
    {
        if (a > b)
            a -= b;
        else
        {
            b -= a;
        }
        if (b == 0)
            break;
    }

}
```

19.10. Cykly

Nyní se dostáváme již do problematiky, kde potřebujeme, aby se nám nějaká část kódu provedla opakovaně. Cykly nejsou nekonečné, nicméně dokud je platná podmínka, tak se jejich tělo provádí. Každému proběhnutí těla cyklu říkáme iterace. Těla cyklů se uzavírají opět do složených závorek. Pokud je potřeba podmínečně ukončit předčasně cyklus, tak to provedeme sepsáním této podmínky a pomocí if, kam do kladné větve umístíme příkaz break, který posune provádění zdrojového kódu za cyklus. Cykly lze vnořovat, kdy s každou iterací vnějšího cyklu se provedou všechny iterace vnitřního cyklu.

19.10.1. Cyklus for

Cyklus for je určen pro předem známý počet opakování těla. Jeho tělo se provede vždy minimálně jednou, i když je podmínka umístěna na začátku cyklu. Za klíčovým slovem for následují kulaté závorky, v niž je pomocná proměnná (může být i inicializována), za středníkem následuje podmínka, která musí platit, aby se cyklus provedl, a opět za středníkem je aktualizace pomocné proměnné, která se provede vždy na konci těla. Pojďme si ukázat, jak tento zběsilý popis vypadá v praxi: for(int i; i ≤ 50; i++)

Pojďme si ukázat, jak takový cyklus for můžeme použít například k výpočtu faktoriálu nějakého čísla, jen to číslo nesmí být nějak velké, protože jinak program skončí přetečením a chybou.

```
#include <iostream>
using namespace std;
int cislo, vysledek = 1;
int main(void)
{
    cout << "Zadejte cislo pro vypocet faktorialu: ";
    cin >> cislo;
    for (int citac = 1; citac <= cislo; citac++)
        vysledek = vysledek * citac;
    cout << "Faktorial cisla " << cislo << " je: " << vysledek << "\n";
    cin >> cislo;
    return 0;
}
```

19.10.2. Cyklus While

Cyklus while používáme tehdy, kdy neznáme počt iterací cyklu. Jen pro připomenutí, iterace znamená vykonání těla cyklu. Podmínka je v tomto případě testována na začátku, takže nemusí být provedeno tělo cyklu. Opět pro připomenutí, tělo cyklu obsahující více nežli jeden příkaz musí být uzavřeno do složených závorek. Tímto cyklem lze vyrobit nekonečnou smyčku a to tak, že na místo podmínky uvedeme klíčové slovo true. Cyklus je vykonáván, doku platí podmínka, pokud podmínka přestane platit, tak se cyklus ukončí. Cyklus je možné také ukončit pomocí příkazu break, který je většinou uveden v nějaké podmínce, pokud se vyskytuje. Obdobně jako u for se v případě v noření dalšího cyklu provedou s jednou iterací vnějšího cyklu všechny iterace vnitřního cyklu.

Jako příklad si ukážeme výpočet konkrétního členu fibonačiho posloupnosti, se kterou se setají, či setkali, ti, kdož mají seminář z matematiky, nebo se účastní matematické olympiády. Principem je výpočet n. členu posloupnosti, kdy n neznáme, protože nám jej zadává uživatel.

```
#include <iostream>
using namespace std;
int f1 = 0;
int f2 = 1;
int nove, i, n, k;
int main()
{
	cout << "Vypocet fibonaciho cisla\n";
	cout << "**************************\n";
	cout << "Zadejte, kolikate fibonacoho cislo chcete zobrazit: ";
	cin >> n;
	cout << "\n";
	//Zde začíná teprve vlastní algoritmus
	if (n <= 1)
	{
		cout << "Pozadovane fibonaciho cislo je " << n << "\n";
	}
	else
	{
		i = 2;
```

```
            while (i<n)
            {
                nove = f1 + f2;
                f1 = f2; //poslední vypočtené se stane přeposledním
                f2 = nove; //posledním je nově vypočtené
                i++;
            }
            cout << "Pozadovane fibonaciho cislo je " << f2 << "\n";
        }
        cout << "Pro ukonceni stisknete libovolné cislo: ";
        cin >> k;
}
```

19.10.3. Cyklud do while

Cyklus do while je cyklus s testem na konci, což znamená, že vždy (bez ohledu na podmínku) se provede minimálně jedna iterace. Tělo cyklu začíná za klíčový slovem do, za kterým následují složené závorky a v nich uzavřené tělo cyklu. Za uzavírací složenou závorkou (zde je vyjímka úpravy, příkaz následuje bezprostředně za závorkou, ne na novém řádku pod ní) následuje klíčové slovo while(pomínka);

Jinak o tomto cyklu platí, že je pro předem neznámý počet iterací, fungují v něm příkazy break a continue, obdobně jako v cyklu while.

Velmi užitečný je pro ukončení programu určitou klávesou, jinak dojde k jinému zadání, nebo pro ukočení klávesou z nabídky Y/N. V tomto případě může nahradit zastaralou strukturu goto, což by při objektově orientovaném kódu nedělao dobrotu. Postup je jednoduchý, do funkce main() napíšeme jako první příkaz do a do těla uzavřeme celý program zakončný tiskem nabídky Y/N a načtením hotnoty typu char (například do roměnné konec). Za tělem bude následovat wihle(konec == ´Y‘); Pokud bude proměnná konec obsahovat hodnotu jinou, nežli Y, tak dojde k ukončení cyklu a provede akorád příkaz return0 a ukončí se program, ovšem pokud bude hodnota Y, tak dojde k tomu, že se celý program včetně inicializace a načtení dat provede znova. V těle je důležité mít i inicializaci, nebo by tělo muselo po načtení hodnoty do proměnné konec obsahovat příkazy na vynulování roměných, či nastavení počátečních hodnot, aby program nepočítal se starými daty.

19.11. Funkce

Funkce je velmi mocný nástroj programátora, protože funkce umožňují skládat kód z toho, co již vytvořil a také umožňují, a to je dalece důležitější, rozdělit zdrojový kód na dílčí problémy, které se řeší naprosto samostatně v jednotlivých funkcích, které se propojí ve správném pořadí ve funkci main. Obecně by se dalo říct, že funkce v programování řeší nějaký dílčí úkol, nebo si ji můžeme představit jako podprográmek, který si zavoláme, když jej potřebujeme. Funkce se vyplatí psát hlavně pro opakované úkony jako je například načítání dat, nebo zobrazení výstupů, popřípadě vložení dílčího (nějakého cizího) algoritmu do našeho.

Každá funkce musí být deklarovaná dříve, nežli je použita, takže jsou dvě možnosti, buď deklarujeme prototyp funkce a pak program může začínat funkcí main(), nebo deklarujeme funkci rovnou a pak spojující funkce main je až na konci programu, což nedoporučuji, protože se hůře hledají chyby v kódu.

Každá funkce se deklaruje hlavičkou, která se nezakončuje středníkem. Hlavička začíná návrhovým typem (např. int, který je nejčastější), pak je uvedeno jméno funkce a v závorkách seznam parametrů. Výjimku tvoří funkce s návratovou hodnotou void, což je funkce main a

většina přetížených funkcí, kdy se do závorek uvádí klíčové slovo void, nebo se nechávají závorky prázdné. Příkladem hlavičky může být: int soucin(a,b,c) a jako zástupce funkcí s parametrem void můžeme uvést hlavičku: int main() .

Tělo funkce, stejně jako u cyklů, uzavíráme do složených závorek (tady na rozdíl od cyklů vždy!) a může obsahovat libovolný počet příkazů, poznámek, větvení, cyklů a dalších struktur. Každá funkce, krom funkcí s návratovou hodnotou void, má návratovou hodnotu!, která se vrací pomocí příkazu *return*.

Hlavička spolu s tělem tvoří takzvanou definici funkce. Nesmíme zapomenout, že volat funkci můžeme jen tehdy, pokud již byla definována minimálně prototypem. Jedinou výjimkou je funkce main(), která se provede vždy implicitně.

Volání funkce provedeme tak, že napíšeme její jméno následované závorkami s parametry (či prázdnými pro void) a to vše je umístěno na samostatném řádku a zakončeno středníkem.

Prototyp funkce je hlavička funkce zakončená středníkem. Zvykem je tyto prototypy uvádět mezi příkazy using namespace a deklarace globálních proměnných. Při překladu pak dojde k tomu, že v té binární podobě se funkce „přesune“ na toto místo. Pokud pak chceme definovat funkci, tak musíme celou hlavičku napsat znova a tentokrát tak jak bylo popsáno výše, tedy bez středníků a s tělem.

Jako příklad použití funkce v programu si ukážeme jednoduchý kalkulátor, kde funkce main obstarává komunikaci s uživatelem a řízení algoritmu, vlastní algoritmus je umístěn v samostatné funkci.

```
#include <iostream>
using namespace std;
void vypocet(); //prototyp funkce
int menu;
double a, b, vysledek;
char konec;
bool esc;
int main(void)
{
	do
	{
	cout << "Menu programu kalkulacka:\n";
	cout << "(1) Scitani \n";
	cout << "(2) Odcitani \n";
	cout << "(3) Nasobeni \n";
	cout << "(4) Deleni \n";
	cout << "Zadejte cislo ukonu z menu: ";
	cin >> menu;
	cout << "\n";
	cout << "Zadejte první cislo: ";
	cin >> a;
	cout << "\n";
	cout << "Zadejte druhé cislo: ";
	cin >> b;
	cout << "\n";
	vypocet(); //volání funkce výpočet
	cout << "Chcete provést nové zadání? (A/N) ";
	cin >> konec;
	if (konec == 'N')
		esc = false;
	else
	{
		esc = true;
	}
	cout << "\n";
	} while (esc);
```

```
	return 0;
}

void vypocet() //vlastní funkce
{
	switch (menu)
	{
	case 1:
		vysledek = a + b;
		break;
	case 2:
		vysledek = a - b;
		break;
	case 3:
		vysledek = a * b;
		break;
	case 4:
		vysledek = a / b;
		break;
	}
	cout << "Vysledek vypoctu je: " << vysledek << "\n";
}
```

19.11.1. Předávání parametrů

Parametry funkce se udávají v závorkách a slouží pro upravení chování funkce. Parametry je nutné uvádět i v prototypu funkce, aby byla funkce správně zadefinována. Když dojde na volání funkce, tak na místo parametru vložíme název proměnné, ve které je parametr uložen. Pokud funkce obsahuje více parametrů, tak se parametry oddělují čárkami, nicméně datový typ parametru je nutné uvést u každého i když se jedná o parametry stejného datového typu. Pokud má funce více parametrů, tak musíme dbát na to, aby parametry byly uvedeny ve správném pořadí, nicméně parametrem může být i konstanta, takže dát pozor, na správný zápis.

Pokud chceme, aby bylo možné měnit hodnotu parametru, tak jej předáváme odkazem, což se dělá tak že za datový typ napíšeme znak & a následuje název parametru. Znak & nesmíme opomenout ani v prototypu funkce. Odkazem lze předávat jen proměnné ☺. Pokud máme více parametrů, tak si můžeme zvolit, který budeme předávat odkazem a který ne, jestli všechny či žádný, v tomto žádná omezení nejsou.

19.11.2. Návratová hodnota funkce

Pokud funkce vrací nějakou hodnotu, tak se typ této hodnoty píše před název funkce jak v prototypu, tak v hlavičce funkce. Ve volající funkci pak návratovou hodnotu používáme na pravé straně operátoru přiřazení, na jehož levé straně je proměnná typově odpovídající návratové hodnoty a na pravé straně stojí název funkce volané. Ve vlastní funkci se pak o vrácené hodnoty stará příkaz return, za nějž přijde proměnná, kde je hodnota uložená, nebo výraz, který tuto hodnotu vytvoří. Pokud je návratová hodnota zobrazována, tak není nutné ji přiřazovat, lze ji vložit přímo do datového proudu objektu cout. Návratová hodnota je vždy jen jedna (jeden int, či double, char, string…), pokud potřebujeme z nějakého důvodu vrátit více hodnot, tak je potřeba použít pole, o kterém bude řeč v následující kapitolce.

Příklad uvedu triviální, v praxi asi ne příliš použitelný, nicméně je na mě vidět, jak se navrací pole hodnot a také ukazuje, jak se tedy s takovou funkcí pracuje.

```
#include <iostream>
using namespace std;
char* precti_jmeno();
int main()
{
	char* jmeno = precti_jmeno();
	cout << jmeno << "\n";
	delete[]jmeno;
	return 0;
}
char* precti_jmeno()
{
	char* jmeno = new char[80];
	cout << "Zadej jmeno: ";
	cin.getline (jmeno, 80);// načte řadu znaků po enter nebo dosažení 79
zadaných znaků
	return jmeno;
}
```

19.12. Pole

Pole je již složenou datovou strukturou, která není implementována v jazyce, ale musíme si ji vytvořit. Jen pro připomenutí z minulé kapitoly, pole slouží pro uložení více hodnot stejného datového typu a významu. K jednotlivým hodnotám v poli přistupujeme pomocí indexu, přičemž první hodnota v poli má index 0, kdy index je kladné celé číslo. Poslední položka nacházející se v poli má index počet prvků v poli – 1. Pole může obecně být n rozměrné, ale používají se jedno až čtyřrozměrná pole, protože každý rozměr znamená jednu vnořenou smyčku pro procházení navíc, což zpomaluje průběh programu. Velikost pole je omezena velikostí operační paměti, a to platí pro klasickou deklaraci, tak také pro dynamickou deklaraci. Znakové pole původně v základním C nahrazuje proměnnou typu string, aby se dalo co nejblíže se přiblížit vlastnostem sringu, tak se užívá dynamické alokování pole.

Co je jiné, nežli v Javě, tak je to, že u základního pole je nutné znát jeho velikost už v průběhu překladu, jinak nedojde k přeložení aplikace, nicméně je zde možné alokovat pole dynamicky, tedy podobně jako v Javě, ale tato deklarace pracuje s tzv. haldou, pole není přímo v paměti vyhrazené pro náš program, ale z našeho programu je na pole pouze ukázáno.

19.12.1. Deklarace

Pole je deklarováno obdobně jako obyčejná proměnná, tato deklarace je jen rozšířena o počet hodnot v hranatých závorkách za názvem proměnné. Pokud není jednoznačně možné určit velikost pole, je lepší deklarovat trošku větší pole, nežli mít nedostatek místa na data. Deklarace pole není sama o sobě nijak obtížnou záležitostí, jen je velmi zásadní pro použití programu, takže vyžaduje hodně přemýšlení od programátora.

Vlastní základní deklarace má zhruba následující strukturu: datový typ jméno[rozměr]; takže například: int výsledky[5];

Další možností jak deklarovat pole v základním provedení je žití tzv. výčtu, čímž provedeme rovnou i inicializaci pole. Pomocí výčtu se pole deklaruje následovně:

Datový typ jméno[] = {hodnota, hodnota,};

Tedy například: int poradi[] = {1,2,3,4,5,6,7,8,9,10};

Pokud bychom pole chtěli deklarovat dynamicky, tak to provedeme následovně:

datový typ* jméno = new datový typ[řídící proměnná];

takže například: int* vysledky = new int[pocet];

Další odlišností od Javy je potřeba mazat pole před ukončením programu a to jak ta dynamická, tak ta, která jsou deklarována přímo v programu. O ta dynamická se v případě opomenutí postará OS, ale zbytečně zabírají paměť. K mazání slouží příkaz delete[];, takže např.: `delete[] vysledky;`.

19.12.2. Konstanty

Konstanta je jméno, které reprezentuje jednu neměnnou hodnotu v programu. Konstanta může být libovolného primitivního datového typu (celočíselné, reálné a znakové datové typy). Deklaruje se klíčovým slovem const následovanou deklarací shodnou s proměnnou a přiřazením hodnoty. Konstantu nelze deklarovat prázdnou a následně do ní později přiřadit hodnotu. Název konstanty se píše velkými písmeny. Často se deklarují globálně, ale i ty lokální zanikají v paměti až s ukončením programu. Konstanty se často užívají k určení rozměru pole, obzvlášť pokud potřebujeme více polí o shodném rozměru.

Příklad vytvoření konstanty: `const int MAX = 3;`

19.12.3. Inicializace

Hodnoty se uvádí za přiřazovacím příkazem ve složených závorkách a vždy v přesném pořadí. Nemusí se inicializovat všechny proměnné, ale neinicializované proměnné jsou vždy umístěny na konci pole. Počet hodnot nesmí překročit počet prvků pole. Pokud je implicitní velikost inicializace, tak necháme hranaté závorky prázdné a za přiřazovací příkaz napíšeme jen požadované hodnoty (lze jen u inicializace!!) a pole se tak zaplní úplně celé.

Neinicializované prvky se inicializují na výchozí hodnotu, tedy číselné na 0 (res. 0,0) a znakové na znak null, proto na rozdíl od Javy není nutné přiřazovat základní hodnoty a pole, či proměnná, může zůstat nevyužito a nevede to k chybám v programu. Při inicializaci znakového pole char se užívá přiřazení znakového řetězce končícího nulovým znakem, nebo znaky zapíšu do uvozovek jako řetězec. Příklad inicializace znakového pole: char jmeno [] = „Niki“ //po provedení cout by bylo vypsáno: Niki

Pole se dá použít jako konstanta (celé pole, tedy jednotlivé prvky se automaticky stanou konstantami, ale uložení konstant do pole se nepoužívá!)

19.12.4. Načtení hodnot do pole a výpis pole na konzoli

Jak operace výpisu celého, nebo části pole, tak načtení celého, nebo části, která má více jak 2 prvky, nebo nevíme, jak velká část pole to bude, se provádí pomocí cyklu. Nejčastěji se pro práce s polem využívá for cyklus, protože využitím jiného cyklu bychom vlastně stejně došli k funkci for cyklu. Pokud má pole jeden rozměr, tak se nejedná o nic komplikovaného, prostě se pomocí jednoho for cyklu projde pole, či jeho část, a cyklicky se provede operace načtení hodnoty z klávesnice, či výpis hodnoty na konzoli.

V případě, že má pole více rozměrů, tak se nám situace mírně komplikuje, protože je potřeba užít více for cyklů vnořených do sebe ve správné hierarchii, je zde nutné si uvědomit, jak takové více rozměrné pole vypadá (nemyslím fyzicky v paměti, ale logicky). Ukážeme si pro jednoduchost načtení a výstup pro jednorozměrné pole, je to v praxi nejčastější případ.

```
#include <iostream>
using namespace std;
void napln(int[], int);
void vypis(int[], int);
const int MAX = 3;
int main()
{
	bool konec;
	do
	{
		int vysledky[MAX];
```

```
		char volba;
		napln(vysledky, MAX);
		vypis(vysledky, MAX);
		cout << "\n";
		cout << "Chcete provest dalsi zadani? (A/N) ";
		cin >> volba;
		if (volba != 'A')
			konec = false;
		cout << "\n";
	} while (konec);
	return 0;
}
void napln(int body[], int velikost)
{
	for (int i = 0; i < velikost; i++)
	{
		cout << "Zadejte vysledek " << i + 1 << ": ";
		cin >> body[i];
	}
	cout << "\n";
}
void vypis(int body[], int velikost)
{
	for (int i = 0; i < velikost; i++)
	{
		cout << "Vysledek cislo " << i + 1 << ": " << body[i] << "\n";
	}
}
```

Jak můžeme vidět, tak tento program ve skutečnosti nic neudělá, v jedné funkci načte do pole hodnoty a druhou funkcí tyto hodnoty vypíše bez jakéhokoli zpracování. Celý program lez volat opakovaně díky uzavření do cyklu do while. V praxi naprosto zbytečný program, ale jde o ukázku načtení a vypsání jednorozměrného pole. Nyní tento program budeme modifikovat pro dvourozměrné pole, takže řekněme pro jednoduchost pro čtvercovou matici 3x3.

```
#include <iostream>
using namespace std;
void napln(int[][], int);
void vypis(int[][], int);
const int MAX = 3;
int main()
{
	bool konec;
	do
	{
		int vysledky[MAX][MAX];
		char volba;
		napln(vysledky, MAX);
		vypis(vysledky, MAX);
		cout << "\n";
		cout << "Chcete provest dalsi zadani? (A/N) ";
		cin >> volba;
		if (volba != 'A')
			konec = false;
		cout << "\n";
	} while (konec);
	return 0;
}
```

```
void napln(int body[][], int velikost)
{
	for (int i = 0; i < velikost; i++)
	{
		for (int j = 0; i < velikost; i++)
		{
			cout << "Zadejte vysledek " << i + 1 << ": ";
			cin >> body[i];
		}
	}
	cout << "\n";
}
void vypis(int body[][], int velikost)
{
	for (int i = 0; i < velikost; i++)
	{
		for (int j = 0; i < velikost; i++)
		{
			cout <<"Vysledek cislo "<< i + 1 <<": " << body[i] <<"\n";
		}
	}
}
```

Do takto upraveného dvojrozměrného pole se nevejdou 3 hodnoty, jako do jednorozměrné varianty, ale 9 hodnot, i při zachování konstanty, která udává rozměr pole. Je to dáno tím, že toto pole je vlastně 3 prvkové pole, kdy každý prvek tohoto pole je opět tříprvkové pole. Když bychom toto pole upravili do 3 rozměrů, tak by postup byl obdobný jako u dvourozměrného pole, ale dočkali bychom se při zachování konstanty kapacity 27 hodnot.

Pokud bychom něco takového chtěli provést s dynamicky alokovaným polem, tak by to mohlo vypadat nějak následovně:

```
#include <iostream>
using namespace std;
int main()
{
	int pocet;
	cout << "Zadejte pocet testu: ";
	cin >> pocet;
	int* vysledky = new int[pocet];
	for (int i = 0; i < pocet; i++)
	{
		cout << "Zadejte " << i + 1 << ". vysledek: ";
		cin >> vysledky[i];
	}
	for (int i = 0; i < pocet; i++)
	{
		cout << i + 1 << ". vysledek :" << vysledky[i] << "\n";
	}
	delete[] vysledky;
	return 0;
}
```

Úprava na vícerozměrné provedení by byla obdobná jako v předchozím případě.

19.13. Abychom se neztratili v paměti, aneb ukazatele

Odkazují na hodnotu proměnné nebo konstanty, která je nejčastěji uložena na hromadě, tedy mimo paměť, která je přímo vyhrazena programu. Jak je vidět, tak se jedná hlavně o problematiku dynamických struktur. Ukazatel může postupně ukazovat na různé proměnné. Pole jsou konstantním ukazatelem, adresa pole v paměti je ve skutečnosti adresou první proměnné, na kterou navazují další jako nepřetržitý blok. S hodnotami v ukazatelích lze pracovat stejně jako s celými čísly (ukazatele se totiž chovají nejčastěji jako proměnná typu int). Pole mohu předat jako ukazatel, pokud místo hranatých závorek napíšu do prototypu hvězdičku, pak hvězdičku musím uvést i v hlavičce, tímto předáním předávám adresu pole, nikoli jeho kopii. Pro předání klasické proměnné mohu ukazatel použít místo odkazu, ale pokud upravuji hodnotu předanou ukazatelem, tak ji musím dereferencovat.

19.13.1. Deklarace

Deklaruje se podobně jako proměnná, tedy za typem (bez mezer) následuje hvězdička a až za ní název (např.: int* vyska;). Ukazatele se nedají deklarovat ve větším množství jedním příkazem, jde to, pokud hvězdičku uvedeme před každý název proměnné, ale není to přehledné. Datový typ ukazatele musí být shodný s datovým typem proměnné, na kterou ukazuje, i když jeho obsahem je vždy adresa v proměnné v paměti v podobě hexačísla.

Při práci s ukazateli ale musíme vědět, co děláme, abychom se nepokusili ukazatelem vstoupit do bloku paměti, který je vyhrazen pro OS či jiný program, poté by došlo k tomu, že když bychom se pokusili načíst hodnotu z cizího bloku nebo ji změnit, tak OS okamžitě chybovým stavem ukončí činnost našeho programu. Tuto chybu většinou neodhalí překladač, ale projeví se hned při prvním spuštění programu, o které se stará překladač. Po deklarování před přiřazením je výchozí hodnotou globálního ukazatele nula, tedy začátek paměti, což je prostor vyhrazený pro OS, konkrétně pro jeho jádro. Proto je zvykem, aby byl ukazatel okamžitě inicializován na adresu příslušné proměnné, neprovádíme samotné deklarace ukazatelů.

Nastavení ukazatele na hodnotu proměnné probíhá pomocí přiřazení a operátoru & uvedeným před jménem proměnné, tedy na příklad: int* ukazatel = &číslo;

Na příkladu jednoduchého kódu si ukážeme, jak se nastavují ukazatele, abychom se vyvarovali chyby při pohybu v paměti.

```
#include <iostream>
using namespace std;
int main()
{
	int cislo = 5;
	int* ukazatel = &cislo;
	cout << "Adresa promene pomoci operatoru: " << &cislo << "\n";
	cout << "Adresa promene pomoci ukazatele: " << ukazatel << "\n";
	return 0;
}
```

19.13.2. Dereferencování ukazatelů

Umožňuje změnit obsah proměnné, na kterou ukazatel ukazuje. Provádí se tak, že se před jméno ukazatele napíše hvězdička a pak se provede přiřazení, příklad: *ukazatel = 15;

Ve skutečnosti dojde k tomu, že se přiřazená hodnota uloží přímo na paměťové místo, jehož adresa je pojmenována proměnnou, takže se to jeví jako změna hodnoty proměnné, ale v procesor nehledá danou proměnnou a neprovádí přiřazení, prostě dojde rovnou v záměně na dané adrese v programu, tedy k přiřazení jako když bychom pracovali přímo s proměnnou a nikoli s ukazatelem na ní.

Obdobně jako lze dereferencovat hodnotu ukazatele, tak můžeme ukazatel použít jako návratovou hodnotu z funkce. Zde je zbytečné se rozepisovat, uveďme si příklad na nějaké funkci.

```
#include <iostream>
using namespace std;
char* precti_jmeno();
int main()
{
	char* jmeno = precti_jmeno();
	cout << jmeno << "\n";
	delete[]jmeno;
	return 0;
}
char* precti_jmeno()
{
	char* jmeno = new char[80];
	cout << "Zadej jmeno: ";
	cin.getline (jmeno, 80);// načte řadu znaků po enter nebo dosažení 79
zadaných znaků
	return jmeno;
}
```

19.13.3. Dynamické alokování paměti

Umožňuje zvolit velikost pole až za běhu programu, nebo vytvářet složitější datové struktury jako jsou spojové seznamy či stromy. Celá struktura se alokuje až za běhu programu, tudíž překladač nevyhodí chybu, pokud nemá jeho velikost struktury (konkrétně u polí). Dynamicky alokovaná paměť není v zásobníku jako proměnné alokované při překladu, ale je na haldě. Dynamicky lze alokovat libovolnou proměnnou, nejčastěji se dynamicky alokují pole, složitější datové struktury a třídy. Alokaci vytvoříme přiřazením (např. pro pole: int* výsledky = new int[pocet]). Dynamické struktury zanikají až s koncem programu, nebo je můžeme odstranit pomocí příkazu delete.

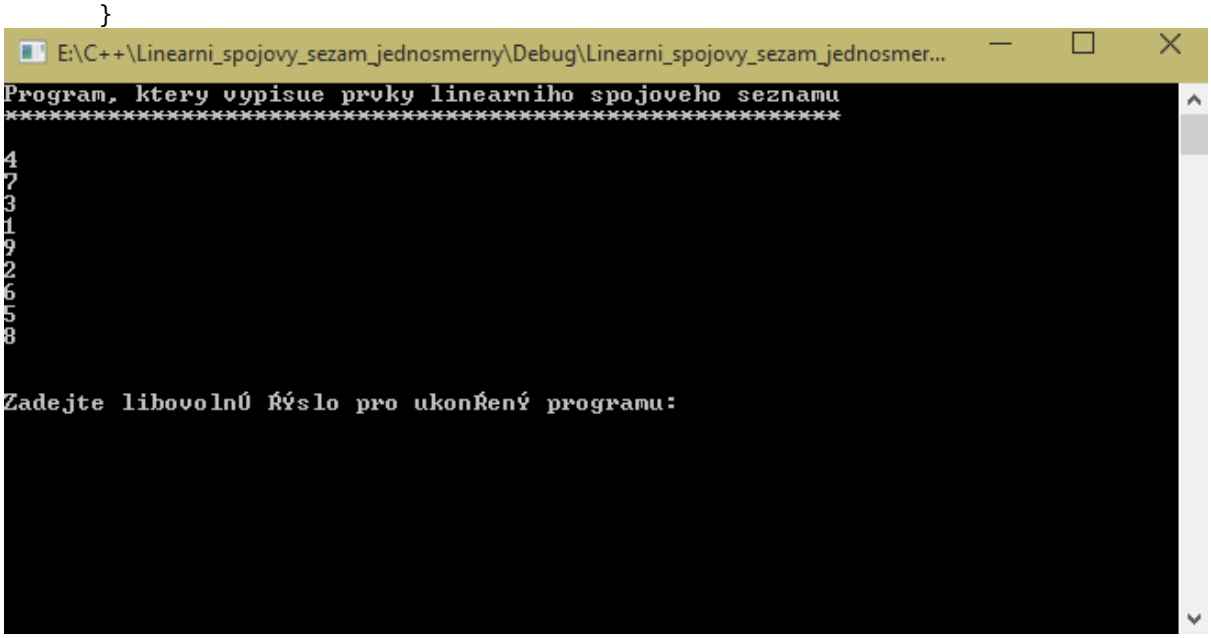
Problém nastává, pokud zanikne ukazatel, pak paměť je alokována, ale je nepřístupná a je na OS, aby po ukončení běhu programu provedl odstranění této alokace, nicméně hlavně u starších verzí OS je tato paměť uvolněna až při nedostatku paměti pro OS. Standardně návrat alokované paměti systému se provádí pomocí operátoru delete za nímž následuje jméno ukazatele (pokud jde o pole, tak se před jméno píší hranaté závorky). Efektivně užíváme dynamickou paměť pro přenos proměnných mezi funkcemi, kdy se ukazatel vrátí jako návratová hodnota funkce.

Ukázku dynamicky alokovaného pole jsme si ukázali již v kapitole o polích, proto zde uvádím příklad další možné datové struktury, což je zřetězený seznam, který propojí jednotlivé hodnoty sám v sobě právě použitím ukazatelů.

```
#include <iostream>
#include <conio.h>
using namespace std;
struct rekord
{
	struct rekord *next;
	int prvek;
};
int n;
typedef struct rekord zaznam;
zaznam *p;
zaznam *q;
int Create(int a[], int n);
void pruchod(zaznam *p);
void relace(zaznam *p);
```

```
int main()
{
	cout << "Program, ktery vypisue prvky linearniho spojoveho seznamu\n";
	cout << "*************************************************\n\n";
	int a[9] = { 8, 5, 6, 2, 9, 1, 3, 7, 4 };
	int n = 9;
	p = NULL;
	if (Create(a,n) != 1)
	{
		pruchod(q);
	}
	relace(q);
	return 0;
}

int Create(int a[], int n) // vytvoří lineární spojový záznam s hodnotami pře-
vzatými z pole a
{
	int i;
	p = NULL;  //p ukazuje na začátekk seznamu
	for (i = 0; i < n; i++)
	{
		q = (zaznam*)malloc(sizeof(zaznam)); //přidělení paměti vrácený
ukazatel přetypujeme
		if (q != NULL) //Test úspěšnosti
		{
			q->prvek = a[i];
			q->next = p;
			p = q;
		}
		else
		{
			return 1; //přidělení paměti nebylo úspěšné
		}
	}
	return 0;
}
```



19.14. Znaky a třída string, aneb naučme počítač číst

Na úvod se ohlédněme malinko zpět, když se vrátíme k primitivním datovým typům, tak vidíme, že pro znaky je určen typ char. V jazyce C, ze kterého C++ vychází, se textové řetězce ukládaly do pole typu char, nicméně pro načítané hodnoty to má jedno značné omezené, je pevně dán počet znaků, které je možné načíst (včetně mezer, teček, čárek, speciálních znaků a číslic). Proměnná typu char je vhodná použití na vstupu, aby program nezkolaboval při zadání jiného znaku nežli čísla, ale pozor, pokud má číslo více jak jednu cifru, tak je nutné užít pole.

C++ nám ulehčuje práci tím, že pro znakové řetězce má speciální proměnnou typu string, která nahrazuje pole typu char. Pokud využíváme proměnné typu string, tak je potřeba připojit knihovnu pomocí `#include <cstring>`. Výhodou užití stringu je, že lze načíst předem neznámou délku řetězce, tedy předem neznámý počet znaků. Pro potřeby výstupu má každý primitivní datový typ k mání implicitní konverzi na string. Proměnná typu string je skritě indexována jako pole, takže můžeme přistupovat k jednotlivým znakům a měnit jejich pořadí, nebo si vyzvedávat jen ty, které potřebujeme. String je také možné užít vhodněji nežli pole char na ošetření vstupu, aby bylo možné zkontrolovat, zda zadaná hodnota je opravdu číslem, pokud ano, tak je možné provést konverzi na typ int nebo double. Ukázkou ošetření vstupu, aby byl opravdu číslem, je následující kód, který využívá standartu C.

```
#include <iostream>
#include <cstring>
using namespace std;
bool zkontroluj(char*);
int main()
{
	char vstup[80];
	cout << "Zadejte cele cislo: ";
	cin >> vstup;
	if (zkontroluj(vstup))
	{
		int cislo = atoi(vstup);
		cout << "Bylo zadano: " << cislo << "\n";
	}
	else
	{
		cout << "Nezadali jste cele cislo";
	}
	return 0;
}
bool zkontroluj(char* vstup)
{
	for (int i = 0; i < strlen(vstup); i++)
	{
		if (i = 0 && vstup[i] == '-' && strlen(vstup) > 1)
			continue;
		if (!isdigit(vstup[i]))
			return false;
	}
	return true;
}
```

Ve třídě string lze s řetězci pracovat, dají se přiřazovat a spojovat. Přiřazují se operátorem = a spojují se pomocí +=.

19.14.1. Čtení z klávesnice do stingu

Ke čtení jednoho slova můžeme použít základní cin >>, ale pokud potřebujeme více slov, tak potřebujme vracet i netisknutelné znaky, což operátor >> neumí. Pro tento případ máme k dispozici členskou funkci objektu cin a tou je funkce get.

Obecně členské funkce objektu voláme tak, že zadáme jméno objektu.jmeno funkce;, tedy pro náší funkci get, by byl příkaz následovný: cin.get(znak); Pokud bychom chtěli číst víceslovné zadání, tak užijeme funkci getline, tedy přesně cin.getline(promena,80); Zadání končí načtením znaku konce řádku. Funkce getline je žravá, ale funkce get není žravá.

19.14.1.1. Čištění Bufferu, aneb cin.ingnore

Tato funkce slouží k odstranění starých enterů z bufferu klávesnice, aby nedošlo k chybnému načtení hodnoty po funkcích, které nejsou žravé. Pokud je volaná bez parametrů, tak načte znak a zahodí jej, přesně v tomto provedení se používá pro čištění bufferu, protože poslední znak je enkter. Pro odstranění enterů je potřeba ji volat před funkcí get, která není žravá, takže by „snědla“ enter a nikam dále by nepokračovala. Jedním parametrem lze nastavit vyšší počet zahozených znaků, dvěma parametry lze kromě počtu zahozených znaků nastavit i znak pro předčasné ukončení zahazování. Pokud potřebujeme jen odstranit poslední enter, tak můžeme tuto funkci nahradit funkcí get bez parametrů.

Pokud je třeba číst znaky po čtení pomocí operátoru >>, tak je nutné vždy volat za operátorem >> funkcí cin.ignore, aby nedošlo k načtení enteru a tím i ukončení čtení. Naopak ji nevoláme po funkci cin.getline, ta je žravá. Pokud se načítá pomocí funkce cin.get, tak musí být spuštění cin.ignore podmíněné.

19.14.2. Porovnávání řetězců

Céčkové řetězce (tedy pole) se nedají standardně porovnat pomocí relačního operátoru. Pro porovnání céčkových řetězců se užívá funkce strcmp, která vrátí nulu v případě schody řetězců, a nula je logické false. Funkce strcmp porovnává řetězce na základě srovnání ASCII kódů jednotlivých znaků, dokud nenarazí na rozdíl nebo na konec obou řetězců. Řetězce třídy sring se porovnávají běžnými relačními operátory.

19.14.3. Převod řetězce na číslo

Funkce pro převod céčkových řetězců na číslo jsou definovány ve standartním knihovním souboru sctdlib. Základní funkcí pro převod z řetězce na číslo je funkce atoi, která má jeden parametr (céčkový řetězec), příklad užití funkce atoi: int číslo = atoi(„7578“);

Pokud načítáme do řetězce string, tak pro užití atoi se musí nejprve převést string na céčko a pak provést atoi, tedy příklad: atoi(vstup.c_str()); Jak vidíme, tak převod stringu na céčkový řetězec je také knihovní záležitostí a leze provést přímo v parametru funkce atoi, takže se jedná jen o jediné, nesmíme na to zapomenout, ale práce s tím žádní není.

19.15. Soubory, neboli zachraňme data

Rozlišujeme stejně jako u Javy dva typy souborů, binární a textové. Binární datové soubory umožňují formátovat, ukládat objekty, ukládat pomocné struktury, ale jsou poměrně složité, takže se jimi nebudeme zabývat. Pro studium práce s těmito soubory mohu doporučit knihu Mistrovství v C++ od nakladatelství Computer Press, nicméně bych před čtením této velmi dobře napsané knihy doporučil si sehnat nějakou učebnici jazyka C++, aby ti, kdo se tím chtějí zaobírat se dokázali dostat na vyšší úroveň, nežli se dostaneme v této knize, přeci jen tato kniha není učebnicí programování.

Soubory nám obecně umožňují vybraná data uchovat na vnějším paměťovém médiu jako je HDD, SSD, SHD či Flash nebo CD/DVD. Malým lákadlem může být to, že si můžeme vymýšlt vlastní koncovky souborů, které nativně přečte naše aplikace, ale u textových je vždy otevře i poznámkový blok.

19.15.1. Souborové datové proudy

Obdobně jako máme vtupněvýstupní datové proudy, které jsou ošetřeny v knihovně iostream, tak i pro práci se soubory máme datové proudy, které jsou ošetřeny v knihovně fstream. Ofstream je určen pro výstup do souborů, dokáže kromě zápisu i vytvořit soubor, pokud se pomocí parametrů neurčí jinak, tak v adresáři aplikace, tedy projektu, a u toho v našem základním kurzu zůstaneme. Ifstream načítá obsah souborů, pokud soubor neexistuje, tak vyvolá chybu. Fstream umožňuje jak zápis, tak čtení souborů, jde o obousměrný datový proud. Všechny názvy datových proudů píšeme s malý písmenem, takže správně: ofstream, ifstream, fstream.

19.15.1.1. Životní cyklus práce se souborem

1. otevřením souboru vznikne komunikační cesta mezi souborem a obslužným objektem
2. probíhají operace čtení a zápisu do souboru
3. soubor se uzavře, aby se uvolnily prostředky nutné pro udržení komunikační cesty mezi souborem a obslužným objektem

19.15.2. Otevření souboru pro zápis

Otevření souboru pro zápis, nebo v případě jeho neexistence je proveditelné objekty typu ofstream a fstream. Oba typy objektů umožňují otevření souboru dvěma způsoby, buď členskou funkcí open nebo konstruktorem.

19.15.2.1. Členská funkce open

Funkce open má dva parametry (cestu k souboru a režim práce se souborem). Pokud soubor neexistuje, tak se otevřením na dané cestě vytvoří. Relativní cesta funguje v umístění programu – soubor.open(„studenti.dat), tedy v adresáši projektu, pokud s programem pracujeme ještě v rámci vývojového prostedí, v případě instalace na OS se poté soubor vytvoří neprakticky v adresáři aplikace. Druhou možností je užití absolutní cesty začínající vždy od kořenového adresáře – soubor.open(„C\\skola\studenti.dat). Pro zadání cesty je možné užít i proměnnou, která obsahuje buď relativní, nebo absolutní cestu, takže se jedná o proměnnou typu string. Je lepší dávat přednost relativním cestám, nebo cestám, teré se pohybují přímo na diskové jednotce OS, rozhodně se vyhýbejme, alespoň na naší úrovni, adresáři user atd.

Pokud užijeme funkci fstream, tak se uvádí i druhý parametr, který může být následující:

ios::app = připisování na konec souboru, stávající obsah souboru zůstane bezezměny
ios::ate = po otevření souboru přejde na jeho konec
ios::Binary = binární režim, informace jsou ukládány v binárním souboru
ios::in = otevřít pro čtení, nexistující soubor se nevytvoří
ios::out = otevření pro zápis, pokud soubor neexistuje, tak se vytvoří

ios::trunc = pokud soubor existuje, tak se zahodí jeho obsah
Příznaky pro práci se souborem lze kombinovat tzv. bitovým součtem, tedy znakem: |

19.15.2.2. Konstruktor

Vytvoří soubor podobně, jako se deklaruje proměnná. Vytvoření souboru kontruktorem: ofstream soubor(„studenti.dat“);
Po vytovření souboru konstruktorem se volá funkce open.

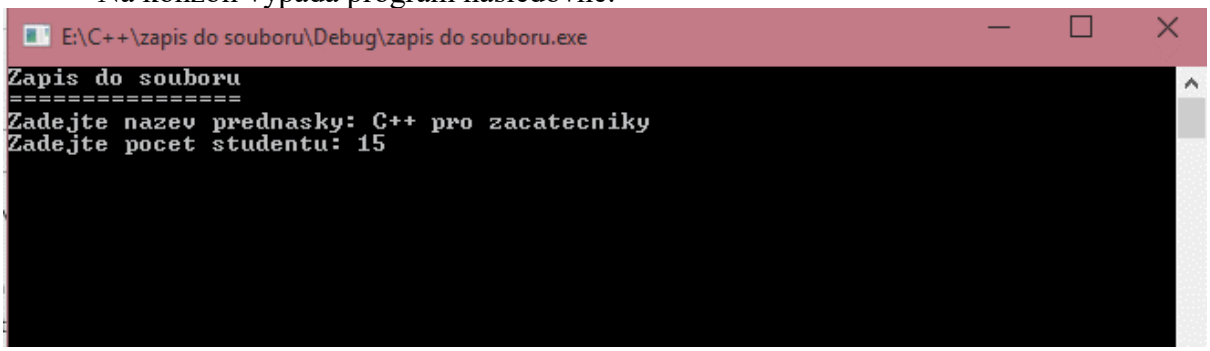
19.15.2.3. Zápis do souboru

Užívá se operátor << stejně jako u cout, přeci jen se jedná o objekty z podobného těsta. Místo cout se zadává proměnná objektu fstream nebo ifstream. Pokud využijeme fstram je nutné dopsat druhý parametr, který udává zvolenou činnost se souborem. U zadávání nemají mezery vliv, pokud zapisujeme uživatelský vstup, tak je dobré číst funkcí getline.

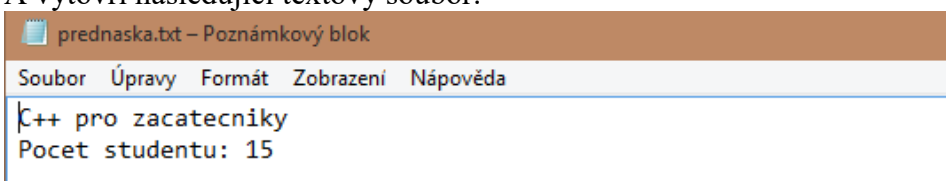
Nyní si ukážeme zcela jednoduchý kód pro zápis do textového souboru, abychom viděli, že to není taková věda, jak to vypadá.

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int main()
{
	char vstup[80];
	ofstream soubor;
	soubor.open("prednaska.txt");
	cout << "Zapis do souboru\n";
	cout << "================\n";
	cout << "Zadejte nazev prednasky: ";
	cin.getline(vstup, 80);
	soubor << vstup << "\n";
	soubor << "Pocet studentu: ";
	cout << "Zadejte pocet studentu: ";
	cin.getline(vstup, 5);
	soubor << vstup << "\n";
	soubor.close();
	return 0;
}
```

Na konzoli vypadá program následovně:



A vytovří následující textový soubor:



19.15.3. Otevření souboru pro čtení

Od zápisu se moc neliší, opět se používají souborové datové proudy, tentokrát ifstream a fstream. příklad: fstream soubor(„studenti.dat“, ios::in);

Využívá se operátor >> stejně jako u cin, ono totiž jde také o vstup, tak proč v tom hledat vědu. Místo cin se užívá objketu fstream nebo ostream. Pokud využijeme fstream je nutné dopsat druhý parametr, který určí typ činnosti se souborem. Pro čtení s mezerami je nutné užít funkci getline, jinak dojde k ignoraci netisknutelných znaků. Čteme v cyklu while, který umořní zjistit konec souboru pomocí funkce fail. Objekt ofstream je vybaven funkcí eof, která nemá parametry a vrátí true, když dojde na konec souboru (funguje velmi spolehlivě i u binárních souborů).

Rozšiřme si kód, který jsme si ukázali v minulé podkapitolece o zápisu do souboru o část, který nám pro kontrolu náš soubor načte a zobrazí na konzoli.

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
int main()
{
	char vstup[80];
	fstream soubor;
	//zápis
	soubor.open("prednaska.txt", ios::out);
	cout << "Zapis do souboru\n";
	cout << "================\n";
	cout << "Zadejte nazev prednasky: ";
	cin.getline(vstup, 80);
	soubor << vstup << "\n";
	soubor << "Pocet studentu: ";
	cout << "Zadejte pocet studentu: ";
	cin.getline(vstup, 5);
	soubor << vstup << "\n";
	cout << "\n";
	soubor.close();
	//čtení
	string vystup;
	soubor.open("prednaska.txt", ios::in);
	cout << "Rekapitulace zadanych dat. \n";
	cout << "========================== \n";
	soubor.getline(vstup, 80);
	while (!soubor.fail())
	{
		cout << vstup << "\n";
		soubor.getline(vstup, 80);
	}
	soubor.close();
	return 0;
}
```

19.15.4. Otevření souboru pro zápis i čtení

Užívá se pouze objekt fstream. Pokud soubor existuje, tak je zachován jeho obsah, ale pokud soubor neexistuje, tak to ničemu nebrání, protože je vytvořen. Zde nastává chvíle, kdy se použije bitový součet parametrů, takže například:

ftream soubor („studenti.dat“, ios::in | ios:out);

Tento způsob otevření souboru nám ummožní, abychom ze stejného souboru četli i do něj zapisovali, ale je nutné potom volat příslušné funkce.

19.15.4.1. Kontrola otevření souboru

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
	ifstream soubor;
	soubor.open("studenti.dat");
	if (soubor.fail())
	{
		cout << "Menohu najit uvedený datovy soubor. \n";
		return 0;
	}

}
```

19.15.5. Uzavření souboru

K uzavření souboru slouží členská funkce close, která se udává bez parametrů. Uzavření je stejné pro zápis i čtení, nebo i kombinovaný datový proud a je vždy nutno. Pozor, na zde může nastata fatální chyba, když na to zapomeneme a opět nás o tom překladač nebude informovat! Příklad: soubor.close();

19.16. Struktury, neboli sjednocení dat

Opět se jedná v této úvodní části o připomenutí minulé kapitoly o Javě, takže to vezmeme stručně a v rychlosti. Struktury slouží ke sjednocení souvisejících proměnných různých datových typů. Jednoduše by se dalo říct, že struktura je datový typ složený z více proměnných libovolného typu. Struktury si vytváří programátor sám, nejsou součástí jazyka, ale existují doplňující knihovny, které pomáhají s tvorbou nejčastěji užívaných struktur. Pokud strukturu předáváme jako parametr funkce, tak se předává vždy odkazem. Pro přehlednost a jednoduchost užívání je dobré struktury vnořovat, pokud chceme strukturu vnořit, musí být nořená struktura deklarována před strukturou, do které ji vnořujeme.

19.16.1. Deklarace

Deklarace začíná klíčovým slovem struct a pokračuje názvem struktury. Tělo struktury se pak uzavírá do složených závorek a za ukončující složenou závorkou se píše středník, aby došlo k odlišení od funkce. Proměnné, které struktura obsahuje, se deklarují v těle struktury (tzv. členské proměnné), členskou proměnnou může být i další struktura. Bývá zvykem struktury deklarovat nad funkcí main, ale v případě potřeby lze strukturu deklarovat i jinde. Příkladem může být následující kousek kódu, který dekladuje strukturu osoba.

```
struct Osoba
{
	string jmeno;
	string prijmeni;
	char pohlavi;
	int vyska;
	bool dospely;
}
```

19.16.2. Instance struktury

Pro deklaraci instance struktury se užije podobný postup jako při deklaraci proměnné, jen místo datového typu se uvede název struktury a za ním pokračuje název instance. Deklarace instance se zakončuje středníkem. Zde na rozdíl oukazatelů lze v jednom příkazu deklarovat více instancí najednou. Lze deklarovat i celé pole instancí, obdobně jako se deklaruje pole proměnných, zde se opravdu struktura neliší od proměnné. Rozsah platnosti a životnost je stejný jako u běžných proměnných, na což musíme myslet.

19.16.3. Přístup k členským proměnným

Pro přístup k členským proměnným slouží operátor tečka, obdobně jako pro přístup k členským funkcím objektu. Nemsíme zapomenout, že před tečku patří název instance!, nikoli celé struktury!! Příklad: osoba.jmeno = „Nikola“;

19.16.4. Inicializacce

Inicializace se dá provést více způsoby, tím asi nejrychlejším je inicializace seznamem hodnot, která se provádí obdobně jako u pole, tedy např.:

osoba o = {„Nikola“, „Horová“, ‚z‘, 175, true};

Další možnost je užití konstruktoru, kdy pokud se nenapíše vlastní konstruktor, tak překladač dodá universální konstruktor. Pokud píšeme vlastní konstruktor, tak se napíše jméno struktury a za něj se vloží prázdné kulaté závorky, dále pak v těle struktury nastavíme výchozí hodnoty. Vlastní konstruktor se velmi často vpisuje do deklarace struktury. Pokud se konstruktor udává mino deklaraci struktury, tak je nutné uvést jméno konstruktoru čtyřtečku (::) a následně konstruktor, aby jej překladač odlišil od funkce. Konstruktory je dobré oddělit od deklarace implementace, aby se implementace mohla libovolně měnit bez narušení zbytku kódu.

19.16.5. Lineární spojový seznam

U dynamické struktury není možné jednoduše přecházet mezi záznamy, pokud jsou uloženy v paměti samostatně, proto se záznam doplňuje o ukazatel, který obsahuje adresu dalšího záznamu, čímž dojde k jejich provázání. Aby byl lineární seznam obousměrný, je potřeba užít dvojice ukazatelů, kdy jeden ukazeue na následovníka a druhý z následovníka zpět na jeho předchůdce. Pro práci s lineárním spojovým seznamem je potřeba připojit hlavičkový soubor conio.h

Protože jsme si lineární spojový seznam ukazovali již u dynamického aplokování paměti, tak si ukážeme jeho malou modifikaci a tou je setřízený linerání spojový seznam.

```
	//v hlavním programu přivádíme záznamy pro kořen seznamu, do něhož vložíme
nulu, a ro zarážku, do níž budeme vkládat hodnotu,
	//kterou chcemem pomocí funkce search v seznamu vyhledat a doplnit, pokud tam
není.
	//ve funkci search procházíme seznam od kořene až k místu, kde najdeme prvek,
který není menší nežli hledaný
	//pokud nalezený prve se nerovná hledanému nebo jde o zarážku, tak jsme na tom
místě, kde seznam rozpojíme, vložíme náš rvek a opět seznam spojíme pomocí ukazatelů
	#include <iostream>
	#include <conio.h>
	using namespace std;
	struct record
	{
		struct record *next;
		int prvek;
	};
	int n;
	typedef struct record zaznam;
	zaznam *koren, *zarazka, *p;
```

```
int hledej(int x, zaznam *p);
void pruchod(zaznam *p);
void relace(zaznam *p);

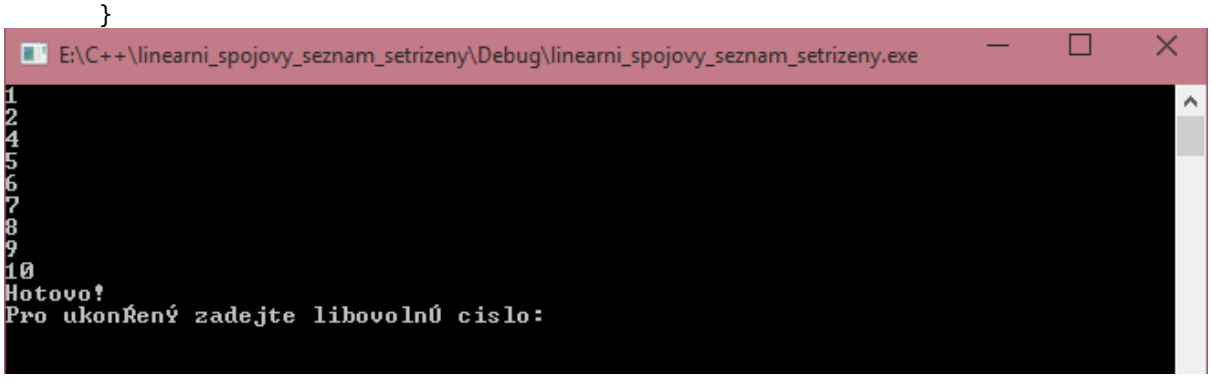
int main()
{
	int x, i;
	koren = (zaznam*)malloc(sizeof(zaznam));
	if (koren == NULL)
	{
		printf("Nedostatek pameti\n");
		return 1;
	}
	koren->prvek = 0;
	zarazka = (zaznam*)malloc(sizeof(zaznam));
	if (zarazka == NULL)
	{
		cout << "Nedostatek pameti\n";
		return 1;
	}
	koren->next = zarazka;
	int pole[10] = { 5, 8, 2, 2, 7, 1, 4, 6, 9, 10 };
	for (i = 0; i < 10; i++)
	{
		x = pole[i];
		if (hledej(x,koren)!=1)
		{
			;
		}
		else
		{
			cout << "Nedostatek pameti\n";
		}
	}
	pruchod(koren);
	cout << "Hotovo!\n";
	//relace(koren);
	int konec;
	cout << "Pro ukončení zadejte libovolné cislo: ";
	cin >> konec;
	return 0;
}

int hledej(int x, zaznam*p)
{
	zaznam *w1, *w2, *w3;
	w2 = p;
	w1 = w2;
	zarazka->prvek = x;
	while (w1 ->prvek < x) //prochází seznam, dokud nachází hodnoty menší
nežli x
	{
		w2 = w1;
		w1 = w2->next;
	}
	if ((w1 ->prvek != x)|| (w1 == zarazka))
	{
		w3 = (zaznam*)malloc(sizeof(zaznam)); //přidělení aměťového místa
		if (w3 == NULL)  //vložení w3 mézi w1 a w2
		{
			return 1;
		}
```

```
                w3->prvek = x;
                w3->next = w1;
                w2->next = w3;
        }
        return 0;
}

void pruchod(zaznam *p)
{
        do
        {
                p = p->next;
                if (p != zarazka)
                {
                        cout << p->prvek << "\n";
                }
        } while (p != zarazka);
}

void relace(zaznam *p) //uvolní paměť
{
        zaznam *q;
        while (p)
        {
                q = p->next;
                if (p)
                {
                        free(p);
                }
                p = q;
        }
}
```



19.17. Třídy

Třídy jen nastíníme v takovém tom pohledu, že něco takového existuje. Platí pro ně většina vlastností struktur a uvozují se klíčovým slovem class. Členské proměnné jsou privátní a mají k nim přístup pouze členské funkce třídy. Členskými funkcemi (nebili metodami třídy) lze spřístupnit členské proměnné i mimo třídu. Od tříd se jako instance odvozují objekty. Třídy využije jen ten, kdo bude chtít v C++ užít objektově orientované programování, protože jak jsme již zmínili v úvodu o jazyce, tak C++ umožňuje jak strukturovaný, tak objektově orientovaný přístup, nebo využití šablon a libovolnou kombinaci přístupů v rámci tvorby aplikace.

19.18. Seznam užívaných objektů standartní knihovny

Objekt	Umístění / skupina	popis
cout	SK / datové proudy	Objekt pro standartní výstup
iostream	SK	Připojuje se k programu pro užívání objektů pro vstup a výstup
string	SK / string	Objekt pro uložení textového řetězce
cin	SK / datové proudy	Objekt slouží pro standartní vstup
cctype	SK	Obsahuje funkce pro práci se znaky

19.19. Seznam užívaných operátorů

Operátor	popis
<<	Vkládá data do proudu
sizeof	Zjišťuje velikost datového typu
&	Zjistí adresu proměnné, která se píše za něj
=	Operátor přiřazení
==	Operátor rovnosti
+	Operátor součtu
-	Operátor rozdílu
*	Operátor součinu
/	Operátor podílu
>>	Čte data z proudu po klávesu enter
%	Operátor funkce modulo
//	poznámka
+=	Zvýšení hodnoty proměnné o hodnotu druhé (výsledek ve stejné proměnné)
>	Větší než
<	Menší než
>=	Větší nebo rovno
<=	Menší nebo rovno
==	Rovno
!=	Nerovno (růzo)=
&&	Slouží ke spojení dvojice booleovských výrazů, čímž vytvoří výraz pravdivý, pokud jsou pravdivé oba výrazy, které jsou spojovány
||	Slouží ke spojení dvojice booleovských výrazů, čímž vytvoří výraz pravdivý, pokud je pravdivý alespoň jeden z původních výrazů
!	Obrátí pravdivostní hodnotu výrazu (negace)
++	Zvýšení hodnoty proměnné o 1
--	Snížení hodnoty proměnné o 1
new	Zajišťuje dynamickou alokaci paměti
delete	Vrací systému dynamicky alokovanou paměť
.	Umožňuje přístup k členským funkcím a proměnným

19.20. Seznam využívaných funkcí

Funkce	Umístění	popis
main	Součást jazyka	Základní funkce programu
pow	cmath	Stará se o mocniny
getline	iostream	Načítá znakové řetězce, má dva parametry, první je objekt cin a druhým je cílová proměnná
get	iostream	Je funkcí objektu cin pro čtení znaku včetně netisknutelných
ignore	iostream	Je funkcí objektu cin pro zahození znaku z bufferu
toupper	cctype	Má jeden parametr char. Znak převede vždy na velký
tolower	cctype	Má jeden parametr char. Znak převede vždy na malý
isalpha	cctype	Vrátí true, pokud je znak písmenem
isalnum	cctype	Vrátí true, pokud je znak písmeno či číslice
isdigit	cctype	Vrátí true, pokud je znak číslice
islower	cctype	Vrátí true, pokud je znak malé písmeno
isprint	cctype	Vrátí true, pokud je znak tisknutelný + mezera
ispunct	cctype	Vrátí true, pokud je znak speciální tisknutelný
isupper	cctype	Vrátí true, pokud je znak velké písmeno
isspace	cctyp	Vrátí true, pokud je znak mezera, enter, tab.
strlen	cstring	Vrací délku textového řetězce uloženého v poli
length	cstring	Vrací délku řetězce proměnné string
size	cstring	Vrací délku řetězce string
strcpy	cstring	Zkopíruje obsah jednoho polole do druhého
strncpy	cstring	Zkopíruje maximálně daný počte znaků z jednoho pole do druhého
strcat	cstring	Připojí obsah druhého řetězce na konec prvního
strcmp	cstring	Porovnává céčkové řetězce
atoi	cstdlib	Převede céčkový řetězec na číslo int
c_str	cstring	Převede string na céčkový řetězec
file	fstream	Kontroluje, zda došlo k chybě při poslední relaci se souborem
open	fstream	Otevře soubor
close	fstream	Uzavře soubor

19.21. Vybrané třídící alegoritmy

V této kapitole si ukážeme vybrané třídící algoritmy, z niž mnohé jsou v jazyce C++ dostupné pomocí knihovních funkcí, ale my si je ukážeme napsané v příkazech, tak jak to vypadá, když se opravdu píší. Zastoupeny zde budou algoritmy, které třídí na místě, ale i takové, které užívají pomocných datových struktur. Každý algoritmus bude krom kódu obsahovat i krádký popis toho, jak pracuje a také, pokud se nic nepokazí, tak i snímek konzole.

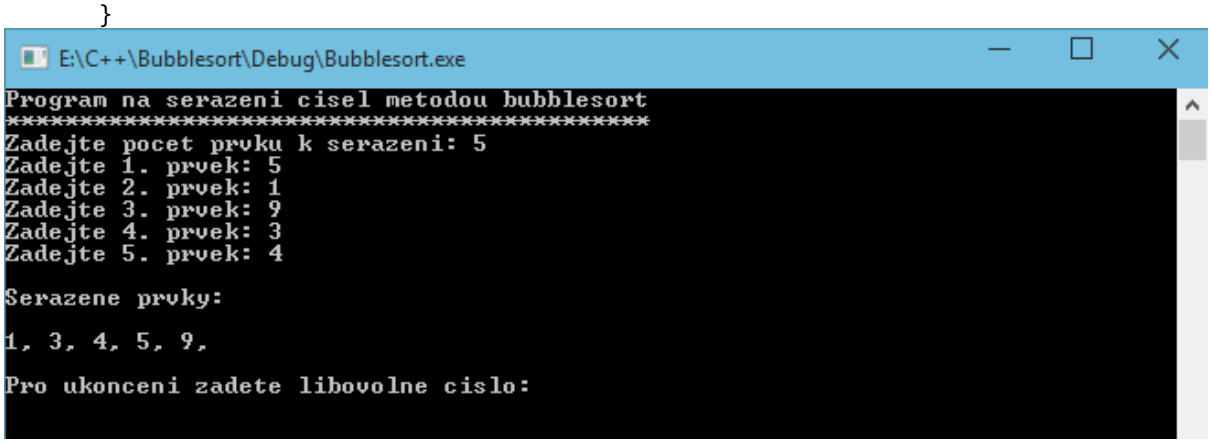
19.21.1. Bubblesort

Pokud si představíme řazená čísla jako bublinky, tak ty s menší hodnotou jsou lehčí než ty s vyšší hodnotou a stoupají proto ve vodě rychleji. Obdobně postupuje také bubble sort.

Porovnává dva sousední prvky, a pokud je nižší číslo nalevo od vyššího, tak je prohodí (nižší číslo je lehčí a rychleji stoupá ke konci pole) a se stejnou logikou pokračuje na dalším

indexu. Pokud jsou čísla ve správném pořadí, tak je neprohodí – pouze postoupí dále (algoritmus tím našel lehčí bublinku). Na konci iterace se tímto způsobem na konec pole vždy dostane ta nejlehčí bublinka (nejnižší číslo). Nyní algoritmus můžeme pustit znovu na redukovaný problém (na poslední pozici pole je již to správné číslo). Po n-1 průchodech (poslední bublinka je seřazena triviálně) je pole seřazeno.

```
#include <iostream>
using namespace std;
int main()
{
	cout << "Program na serazeni cisel metodou bubblesort\n";
	cout << "****************************************\n";
	int pocet;
	cout << "Zadejte pocet prvku k serazeni: ";
	cin >> pocet;
	int* p = new int[pocet];
	for (int i = 0; i < pocet; i++)
	{
		cout << "Zadejte " << i + 1 << ". prvek: ";
		cin >> p[i];
	}
	cout << "\n";
	//Zde začíná algoritmus
	int i, j;
	int x; //pomocná proměnná pro výměnu prvků
	for (i = 1; i < pocet; i++)//v prvním průchodu se dostane nejmenší prvek nahoru
	{
		for (j = pocet - 1; j >= i;j--)
		{
			if (p[j - 1] > p[j]) //inverze - provede se výměna
			{
				x = p[j - 1];
				p[j - 1] = p[j];
				p[j] = x;
			}
		}
	}
	cout << "Serazene prvky:\n\n";
	for (int i = 0; i < pocet; i++)
	{
		cout << p[i] << ", ";
	}
	int konec;
	cout << "\n\nPro ukonceni zadete libovolne cislo: ";
	cin >> konec;
	return 0;
}
```

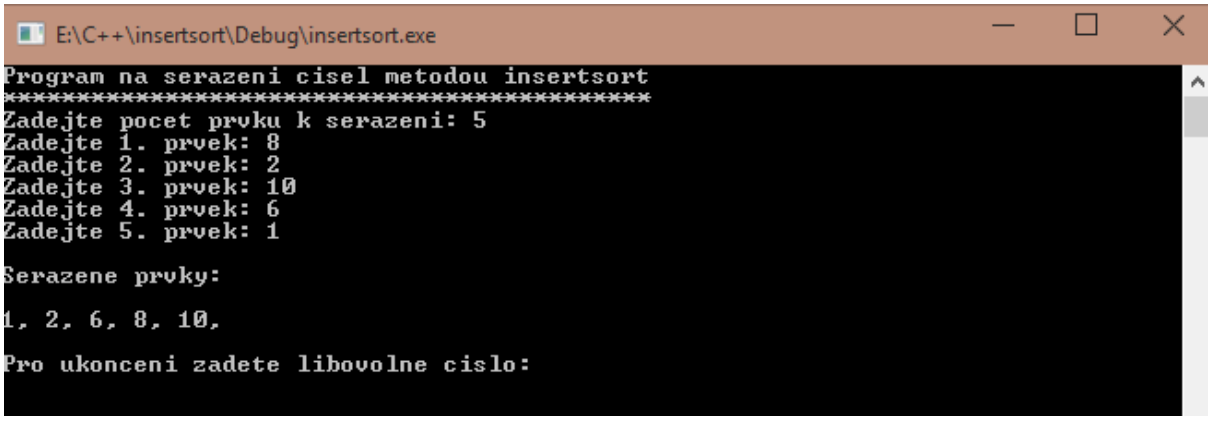


19.21.2. Insertion sort

Insertion sort využívá tohoto myšlenkového postupu:

1. Mějmě jeden prvek, ten je triviálně seřazen.
2. Vezměme následující prvek a zařaďme jej na správné místo v již seřazených prvcích. (Tímto je nyní seřazeno o jeden prvek více než v předchozím kroku.)
3. Dokud pole obsahuje nezařazené prvky, tak **proveď: 2**.

```
4. //množinu prvků rozdělíme na jednoprvkou podmnožinu a zbytek
5. //jednoprvková podmnožina je setřízená, takže stačí jen brát prvky ze zbytku a
   vkládat je na správné místo v původně jednoprvkové množině
6. //setřízené prvky jsou v poli umístěny od indexu 1, aby nám index 0 zůstal
   volný pro případné vložení nového čísla
7. #include <iostream>
8. using namespace std;
9. int main()
10. {
11.         cout << "Program na serazeni cisel metodou insertsort\n";
12.         cout << "****************************************\n";
13.         int pocet;
14.         cout << "Zadejte pocet prvku k serazeni: ";
15.         cin >> pocet;
16.         int* p = new int[pocet];
17.         p[0] = 0;
18.         for (int i = 0; i < pocet; i++)
19.         {
20.                 cout << "Zadejte " << i + 1 << ". prvek: ";
21.                 cin >> p[i+1];
22.         }
23.         cout << "\n";
24.         //Zde začíná vlastní algoritmus
25.         int i, j;
26.         for (i = 2; i <= pocet; i++)
27.         {
28.                 p[0] = p[i];
29.                 j = i - 1;
30.                 while (p[0]<p[j])
31.                 {
32.                         p[j + 1] = p[j];
33.                         j--;
34.                 }
35.                 p[j + 1] = p[0];
36.         }
37.         //Zde algoritmus končí
38.         cout << "Serazene prvky:\n\n";
39.         for (int i = 1; i <= pocet; i++)
40.         {
41.                 cout << p[i] << ", ";
42.         }
43.         int konec;
44.         cout << "\n\nPro ukonceni zadete libovolne cislo: ";
45.         cin >> konec;
46.         return 0;
47. }
```



19.21.3. Mergesort

Slévání

Předpokládejme, že máme dva seznamy (**A**, **B**) seřazené v sestupném pořadí. Tyto seznamy můžeme setřídit do jediného seznamu (**C**) následující procedurou.

V každém kroku vždy porovnejme první prvky obou seznamů (tj. nejvyšší prvky obou seznamů) a vyberme ten vyšší. Tento nakopírujme na konec seznamu **C** a odstraňme jej ze seznamu původního. Tento postup opakujme tak dlouho, dokud se jeden ze seznamů nevyprázdní. Potom překopírujme zbytek druhého seznamu do **C** (a odstraňme překopírované prvky z původního seznamu).

Ve skutečnosti slévá merge sort vždy dvě sousední části pole. Drží si dva ukazatele, každý na první prvek seznamu a po každém porovnání přesune jeden z prvků do pomocného pole a posune příslušný ukazatel o jedno místo (čímž se dostane na nový nejvyšší prvek příslušného seznamu). Poté, co zkopíruje všechny prvky obou seznamů do pomocného pole, tak celé původní pole přepíše seřazeným seznamem (pomocným polem).

Dělení

Dosud jsme nezmínili, jak algoritmus získá dvě seřazená sousední pole. Dělicí část merge sortu má na svém vstupu celé pole. Toto rozdělí na dvě stejně velké části, tyto dvě části pak dále rekurzivně dělí. V okamžiku, kdy rekurze narazí na seznamy jednotkové velikosti, tak se zastaví. Nyní má algorimus v každé větvi k dispozici dva sousední seznamy, které obsahují jeden prvek a jsou tedy triviálně seřazeny.

Řazení

Merge sort se tedy začne navracet z rekurze a při každém návratu sleje dva seznamy pomocí výše zmíněné procedury *slévání*. Algoritmus má jistotu, že buď slévá triviálně seřazené prvky nebo seznamy, které již byly slity. V okamžiku, kdy se merge sort plně navrátí z rekurze, tak terminuje. Pole je seřazeno od nevyšší hodnoty.

```
	//mergesort je třízení slučováním
	//vzpočívá v rozdělení množiny na dvě podmnožiny, které jsou srovnány samostaně
a následně sloučeny
	//setřízení dílčích úseků probíhá pomocí rekurze
	//jde o nejrychlejší algoritmus na třízení dat, jeho nevýhodou je nutnost exis-
tence pomocného pole, které je stejně velké jako pole třízené
	#include <iostream>
	using namespace std;
	int l, r;
	int p [6];
	int q [6];  //pomocné pole
	void merge(int p[], int q[], int l, int r);
	int main()
	{
		cout << "Program na serazeni cisel metodou mergesort\n";
		cout << "****************************************\n";
```

```
	for (int i = 0; i < 6; i++)
	{
		cout << "Zadejte " << i + 1 << ". prvek: ";
		cin >> p[i];
	}
	cout << "\n";
	merge(p, q, l, r);
	return 0;
}
void merge(int p[], int q[], int l, int r) //vlastní algoritmus
{
	int i, s, j, k;
	s = (l + r) / 2;
	if (l < s)
	{
		merge(p, q, l, s);
	}
	if (s + 1<r)
	{
		merge(p, q, s + 1, r);
	}
	//slučování úseků
	i = 1;
	j = s + 1;
	k = 1;
	while ((i <= s) && (j <= r))
	{
		if (p[i] <= p[j])
		{
			q[k] = p[i++];
		}
		else
		{
			q[k] = p[j++];
		}
		k++;
	}
	while (i <= s)
	{
		q[k] = p[i++];
		k++;
	}
	while (j <= r)
	{
		q[k] = p[j++];
	}
	//přeneseme setřízený úsek z q do p
	for (k = l; k <= r; k++)
	{
		p[k] = q[k];
	} //zde končí vlastní algoritmus
	cout << "Serazene prvky:\n\n";
	for (int i = 0; i < 6; i++)
	{
		cout << p[i] << ", ";
	}
	int konec;
	cout << "\n\nPro ukonceni zadete libovolne cislo: ";
	cin >> konec;
}
```

19.21.4. Quicksort

Zvolme v zadaném poli libovolný prvek a říkejme mu *pivot*. Nyní můžeme pole přeházet tak, aby na jedné straně byly prvky větší než pivot, na druhé menší než pivot a pivot samotný byl umístěn přesně mezi těmito částmi. Tento postup můžeme zopakovat pro obě rozdělené části (bez pivota, ten je již umístěn na správném místě). Proceduru opakujeme tak dlouho, dokud nenarazíme na všechny triviálně řešitelné podproblémy (pole velikosti 1). V tento okamžik je celé pole seřazeno od nejvyššího prvku.

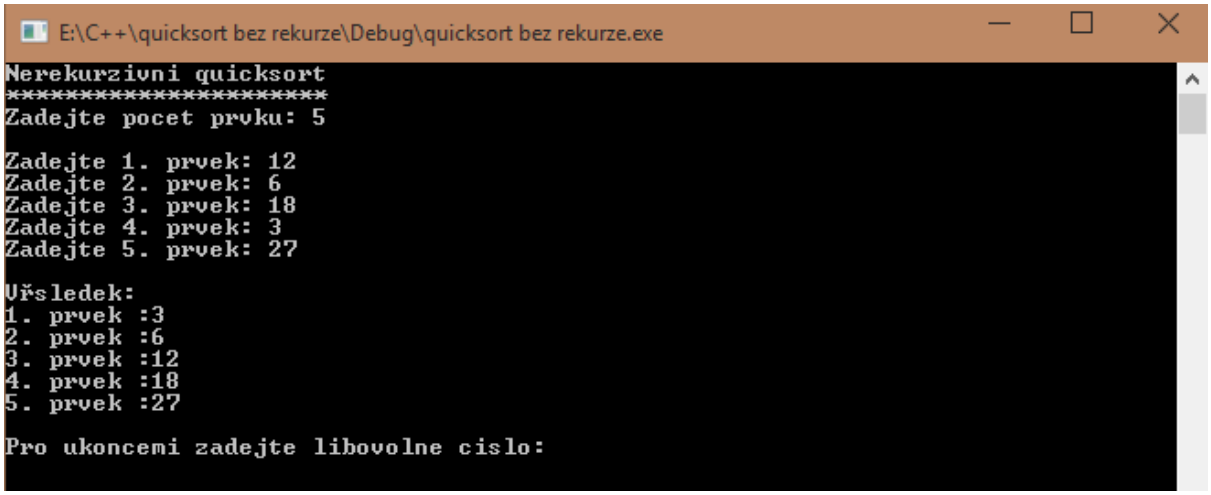
```
//tento algoritmus rozděluje množinu prvků pomocí voleného pivotu
//prvky se rozdělí na 3 skupiny: menší, pivot a větší
// v každé skupině (krom pivotu) opět volíme pivot a vše se opakuje
#include <iostream>
using namespace std;
int p[6], l, r;
void quick(int p[], int l, int r);
int main()
{
	cout << "Program na serazeni cisel metodou quicksort\n";
	cout << "*****************************************\n";
	for (int i = 0; i < 6; i++)
	{
		cout << "Zadejte " << i + 1 << ". prvek: ";
		cin >> p[i];
	}
	cout << "\n";
	quick(p, l, r);
	return 0;
}
void quick(int p[], int l, int r)
{
	int i, j, x, y;
	i = l;
	j = r;
	x = p[(l + r) / 2];
	do
	{
		while (p[i] < x && i < r)
			i++;
		while (p[j] > x && j > l)
		{
			j--;
		}
		if (i <= j)
		{
			if(i<j)
			{
				y = p[i];
				p[i] = p[j];
				p[j] = y;
			}
			i++;
			j--;
		}
	} while (i <=j);
	if (l < j)
	{
		quick(p, l, j);
	}
	if (i < r)
	{
```

```
                quick(p, i, r);
        }
        cout << "Serazene prvky:\n\n";
        for (int i = 0; i < 6; i++)
        {
                cout << p[i] << ", ";
        }
        int konec;
        cout << "\n\nPro ukonceni zadete libovolne cislo: ";
        cin >> konec;
        return;
}
```

19.21.4.1. Nerekurzivní varianta

```
//tato verze quicsortu nahrazuje dvojitou rekurzi použitím zásobníku
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
        int pocet;
        cout << "Nerekurzivni quicksort\n";
        cout << "**********************\n";
        cout << "Zadejte pocet prvku: ";
        cin >> pocet;
        cout << "\n";
        int* p = new int[pocet];
        for (int i = 0; i < pocet; i++)
        {
                cout << "Zadejte " << i + 1 << ". prvek: ";
                cin >> p[i];
        }
        cout << "\nVýsledek:\n";
        //Zde začíná algoritmus
        int n;
        int zacatek[12], konec[12]; //zásobníky pro označení začátků a koců o ve-
likosti 2*log(pocet)
        int i, j, l, r, x, y;
        int uk = 0; //ukazovátko v zásobnících
        zacatek[uk] = 0;
        konec[uk] = pocet - 1;
        do
        {
                l = zacatek[uk];
                r = konec[uk];
                uk--;
                i = l;
                j = r;
                x = p[(l + r) / 2];
                do
                {
                        while (p[i]<x && i<r)
                        {
                                i++;
                        }
                        while (p[j]>x && j>l)
                        {
                                j--;
                        }
                        if (i<=j)
                        {
                                if (i<j)
```

```
                        {
                                y = p[i];
                                p[i] = p[j];
                                p[j] = y;
                        }
                        i++;
                        j--;
                }
        } while (i<=j);
        if ((j-l)<(r-i)) //levý úsek je menší
        {
                if (i<r)   //ulož pravý aek do zásobníku
                {
                        uk++;
                        zacatek[uk] = i;
                        konec[uk] = r;
                        r = j;
                }
                if (j>l)  //ulož levý úsek do zásobníku
                {
                        uk++;
                        zacatek[uk] = l;
                        konec[uk] = j;
                        l = i;
                }
        }
        else  //pravý úsek je menší
        {
                if (j>l)
                {
                        uk++;
                        zacatek[uk] = l;
                        konec[uk] = j;
                        l = i;
                }
                if (i<r)
                {
                        uk++;
                        zacatek[uk] = i;
                        konec[uk] = r;
                        r = j;
                }
        }
    } while (uk>=0); //Zde končí algoritmus
    for (int i = 0; i < pocet; i++)
    {
            cout << i + 1 << ". prvek :" << p[i] << "\n";
    }
    free(p);
    int k;
    cout << "\nPro ukoncemi zadejte libovolne cislo: ";
    cin >> k;
    return 0;
}
```

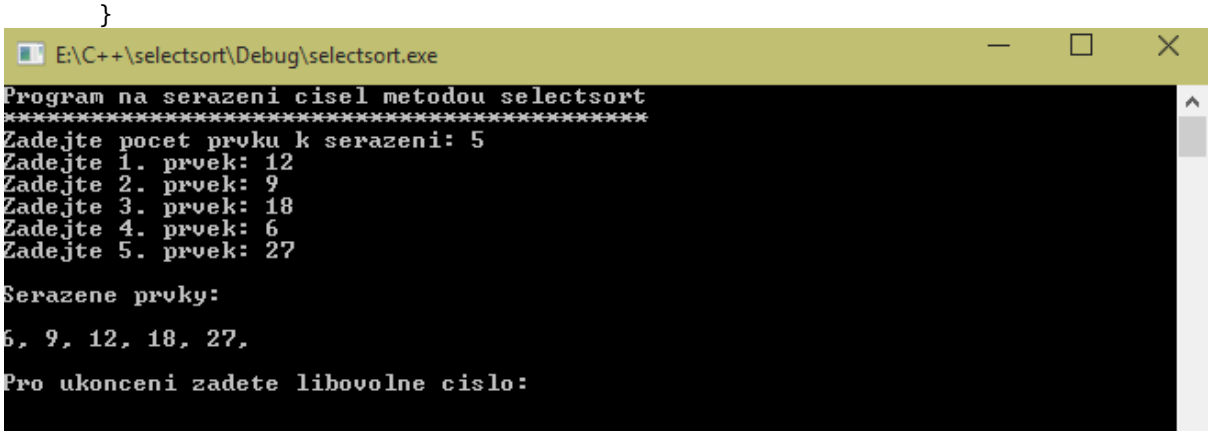


19.21.5. Selection sort

Selection sort vychází z myšlenky, že pokud řadíme pole od největšího prvku k nejmenšímu, tak první bude nejvyšší prvek, za ním nejvyšší prvek ze zbytku pole atd., čili stačí pouze postupně vybírat nejvyšší prvky z neseřazené části pole a umísťovat je na konec seřazené části.

```
	//třídění pole přímým výběrem, kdy se najde nejmenší posune na první místo, druhý nejmenší na druhé atd.
	//prvky jsou v poli p umístěny od indexu 0 a jejich počet je pocet
	#include <iostream>
	using namespace std;
	int main()
	{
		cout << "Program na serazeni cisel metodou selectsort\n";
		cout << "****************************************\n";
		int pocet;
		cout << "Zadejte pocet prvku k serazeni: ";
		cin >> pocet;
		int* p = new int[pocet];
		for (int i = 0; i < pocet; i++)
		{
			cout << "Zadejte " << i + 1 << ". prvek: ";
			cin >> p[i];
		}
		cout << "\n";
		int i, j, k, x;
		for (i = 0; i < pocet-1; i++)
		{
			k = i;
			x = p[i];
			for (j = i+1; j < pocet; j++)
			{
				if (p[j]<x)
				{
					k = j;
					x = p[j];
				}
			}
			p[k] = p[i];
			p[i] = x;
		}
		cout << "Serazene prvky:\n\n";
		for (int i = 0; i < pocet; i++)
		{
```

```
                cout << p[i] << ", ";
        }
        int konec;
        cout << "\n\nPro ukonceni zadete libovolne cislo: ";
        cin >> konec;
        return 0;
}
```



20. Databáze

Jedním z velkých problémů informatiky v současnosti, ale i v poměrně nedávné minulosti, je obrovské množství dat, které je potřeba ukládat, zpracovávat a udržovat jejich vzájemné souvislosti. Jedním z typů dat jsou tzv. proudová data, která se analyzují v reálném čase a ukládají se jen výsledky analýz, k tomu slouží služby jako je Microsoft Azure Stream analytics. Dalším fenoménem dnešní doby jsou tzv. big data, kterým budeme věnovat samostatnou podkapitolu v závěru této kapitoly.

Řešením problémů s ukládáním velkých objemů dat jsou různé typy databází. Databáze dělíme na dvě velké skupiny, relační (SQL) a nonSQL databáze. V našem dalším povídání se zaměříme právě nerelační databáze, protože jsou nejrozšířenější a také jejich pochopení tvoří základ pro všechno ostatní.

20.1. Základní pojmy

Data je potřeba zaznamenávat dohodnutým způsobem, a to nejen z důvodu kompatibility, ale také z důvodu jejich identifikace v paměti, protože data jsou v paměti jen posloupnosti nul a jedniček, a pokud by nebyla uložena dohodnutým zůsobem, tak by nebylo možné různá data od sebe odlišit.

Informace je jedním ze základních pojmů informatiky. Opravdu výrazně zjednodušeně lze říct, že informace jsou data s přiřaženým významem. Pokud vezmeme strukturovaná data a přiřadíme jim význam, dostaneme databázi.

Abychom mohli databázi vytvořit, spravovat a celkově provozovat, potřebujeme nějaký SW, který nám to umožní. Těchto SW existuje více, ale všechny jsou označovány společným názvem systém řízení báze dat (SŘBD) nebo anglicky (DBMS), který má následujíí vlastnosti:

- Služby ovládání souborů – vstup / výstup
- Datové služby – práce s daty
- Logika dat – definuje pravidla pro data
- Logika aplikace – algoritmus aplikace a jeho realizace
- Prezentační logika – řízení interakce
- Prezentační služby – vykreslování, generování výstupních souborů

Databáze mají dvě hlavní funkce, tou první je oddělení dat od aplikace a jejího kódu, což má velké výhody například v možnosti použít i jinou aplikaci pro jejich zpracování, nebo nezávislost dat na verzi aplikace. Druhou velkou výhodou databází je jejich snadná přenositelnost, která nepřímo souvisí s tím, že data nejsou závislá na své aplikaci.

V nejobecnějším případě lze databázi definovat jako kolekci souvisejících záznamů, které obsahují vlastní popis. Pro relační databáze se dá říct, že jde o kolekci souvisejících tabulek, které obsahují vlastní popis, což znamená, že popis struktury je součástí samotné databáze.

20.2. Systémy řízení báze dat

Data popisující strukturu databáze se nazývají metadata a jako příklad můžeme uvést názvy tabulek či sloupců v tabulkách. Všechny systémy řízení báze dat dokáží s jejich využitím zobrazit a upravit strukturu databáze. Kromě dat a metadat se v databázi nachází indexy, žurnály a další struktury, které umožňují zvýšit výkon a spolehlivost databáze.

Úkolem systému řízení báze dat je vytvářet, zpravovat a udržovat databáze, proto každý z nich obsahuje následující funkce:

- Vytvoření databáze
- Vytvoření tabulek
- Vytvoření podpůrných struktur

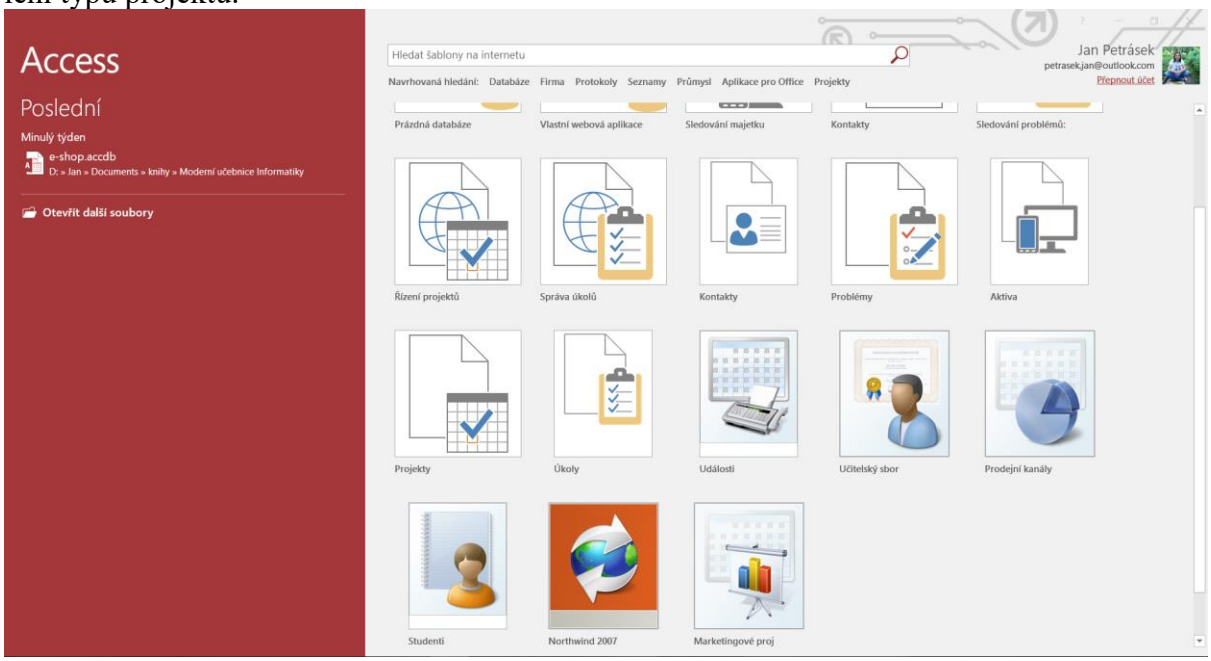
- Čtení dat z databáze
- Úpravy databázových dat
- Údržba databázových struktur
- Vynucování pravidel
- Kontrola souběžnosti
- Zajištění bezpečnosti
- Zálohování a obnova databází

Příkladem těchto systémů je Microsoft SQL server, Microsoft Access (součást vyšších sad MS Office, samostatně dokoupitelná komponenta), MySQL, DB2 (IBM) a Oracle Database. Obecně tyto systémy dělíme do dvou skupin: osobní (MS Access) a podnikové (MS SQL Server Enterprise). Pro praktické ukázky budeme využívat MSSQL Server 2016 Developer a MS Access 2016. Pro práci s MSSQL Server je potřeba vývojové prostřední, buď MS Management studio, nebo MS Visual studio.

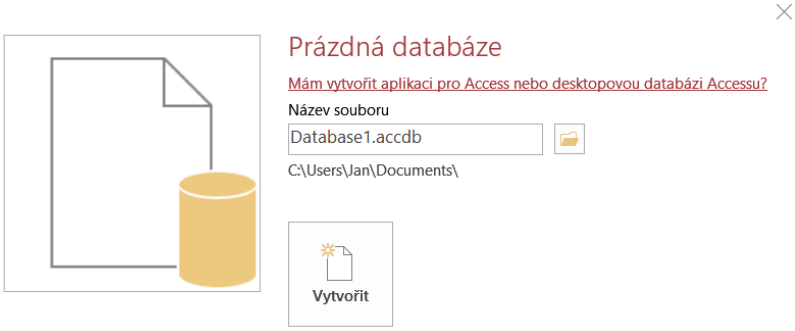
Systém přijímá příkazy v jazyce SQL, což je dotazovací jazyk 4. generace, kdy systému jen popíšeme, co chceme, aby udělal a on se snaží aktivně dotaz pochopit, nedefinujeme mu, jak to má udělat. SŘBD umožňuje deklarovat pravidla, která musí datové hodnoty v databázi splňovat, tato pravidla se nazývají omezení referenční integrity. Dále pak jsou tyto systémy optimalizované pro souběžbou práci více uživatelů, kteří k databázi přistupují a mají implementované mechanismy zabraňující kolidaci práce více uživatelů. Uživatelům lze nastavovat oprávnění na přístup a manipulaci s daty v databázi.

20.2.1. Micorosoft Access 2016

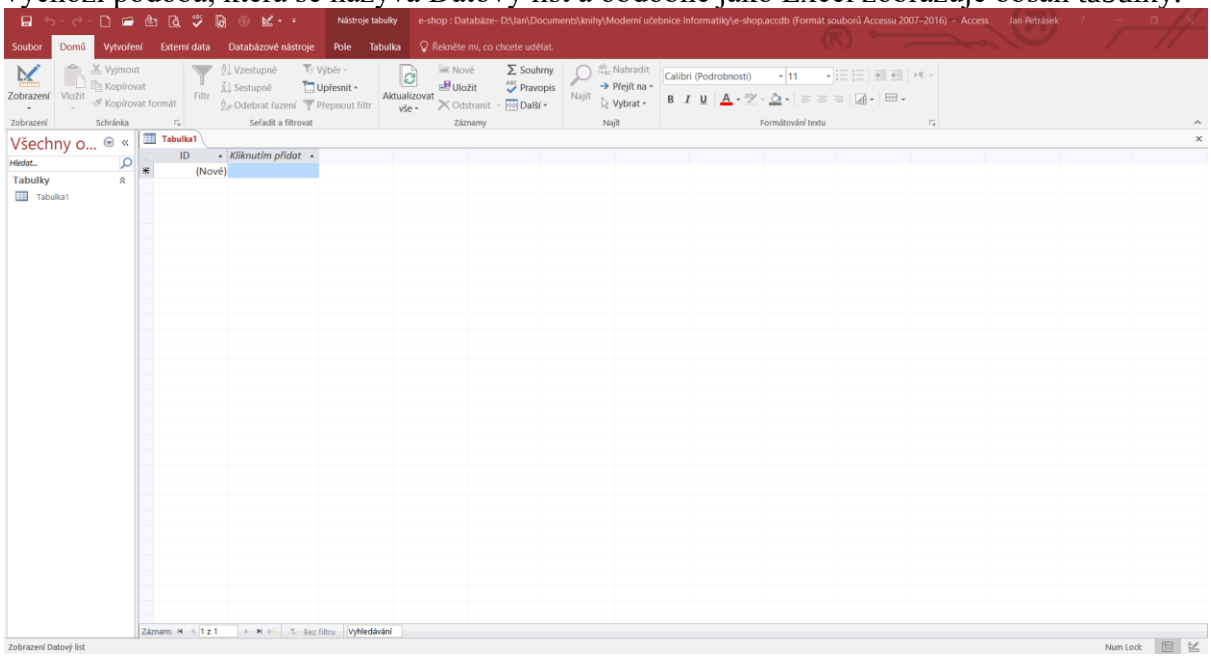
Na tomto místě si ukážeme základní rozložení a popíšeme základy zacházení s MS Access. Bližší další informace vždy budou ukázány v průběhu celé kapitoly o databázích, vždy jak to bude potřebné k výkladu. Access nás přivítá následujícím oknem, kde nás vybízí ke zvolení typu projektu.



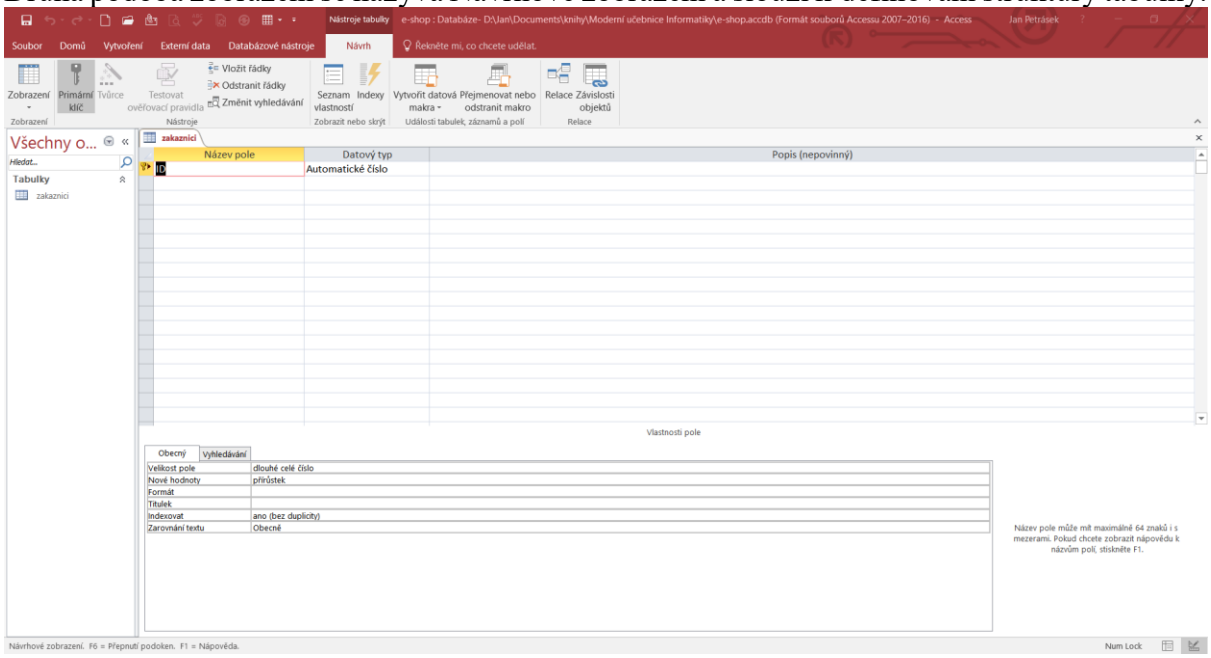
Po zvolení databáze nás Access vyzve k volbě názvu databáze a jejího umístění. Po vyplnění vše stvrdíme tlačítkem Vytvořit. Pro název databáze nepoužíváme diakritiku a mezery, pokud to nedodržíme, většinou se nic neděje, ale pokud se do databáze dotazujeme externě, tak často nastane problém při nedodržení tohoto pravidla.



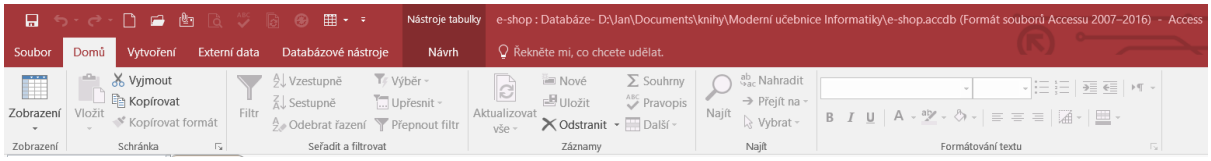
Access má dvě základní zobrazení (dvě podoby pracovního prostředí). Tou první je výchozí podoba, která se nazývá Datový list a obdobně jako Excel zobrazuje obsah tabulky.



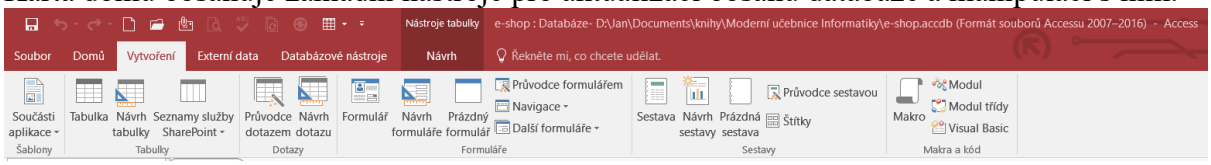
Druhá podoba zobrazení se nazývá Návrhové zobrazení a slouží k definování struktury tabulky.



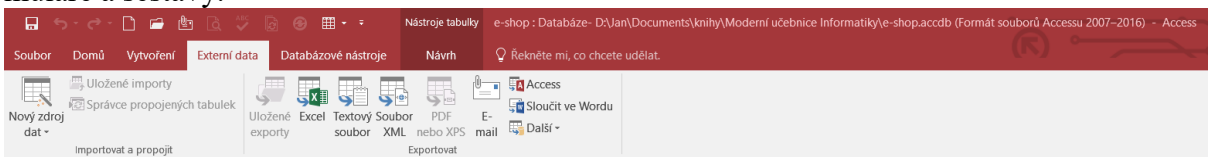
Nyní si pro návrhové zobrazení ukážeme, co se kde nahází, respektive, co obsahují jednotlivé karty bez bližšího popisu. Blíže se k jednotlivým nabídkám dostaneme při praktických ukázkách.



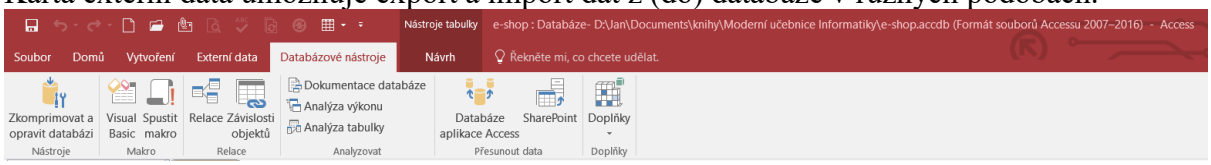
Karta domů obsahuje základní nástroje pro aktualizaci obsahu databáze a manipulaci s ním.



Karta Vytvoření nám nabízí různé komponenty, které můžeme v rámci projektu vytvářet. Nejdůležitější jsou Tabulka a průvodce, respektive návrh dotazu, poměrně časté jsou potom formuláře a sestavy.

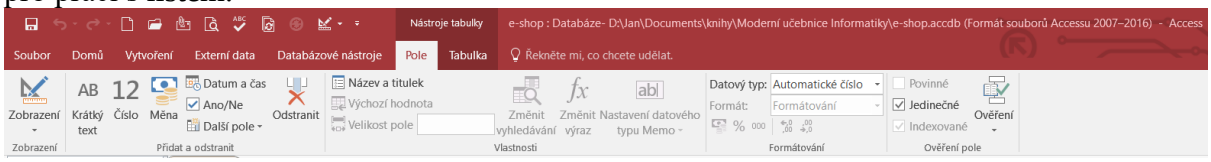


Karta externí data umožňuje export a import dat z (do) databáze v různých podobách.

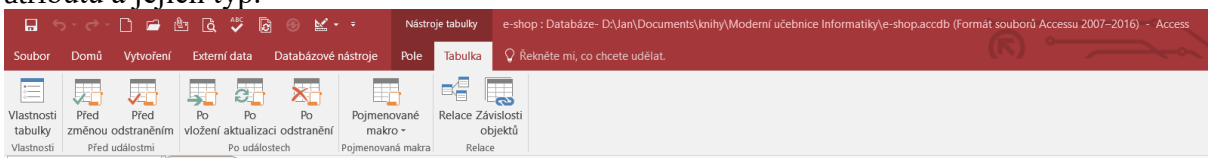


Karta databázových nástrojů patří mezi nejdůležitější karty pro vývoj databáze, zde se například zpravují relace mezi tabulkami.

Pokud přepneme zobrazení do Datového listu, objeví se nám dvojice nových karet, která slouží pro práci s listem.



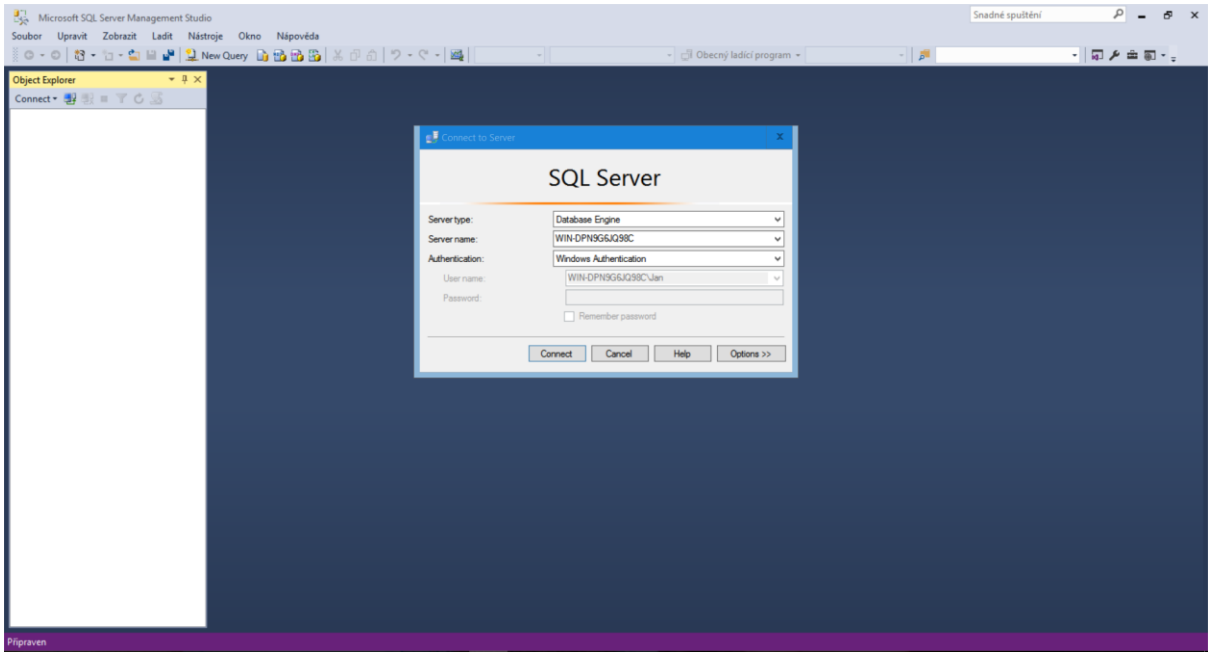
Při vytváření tabulky je asi nejpodstatnější karta pole, která umožňuje nastavovat datové typy atributů a jejich typ.



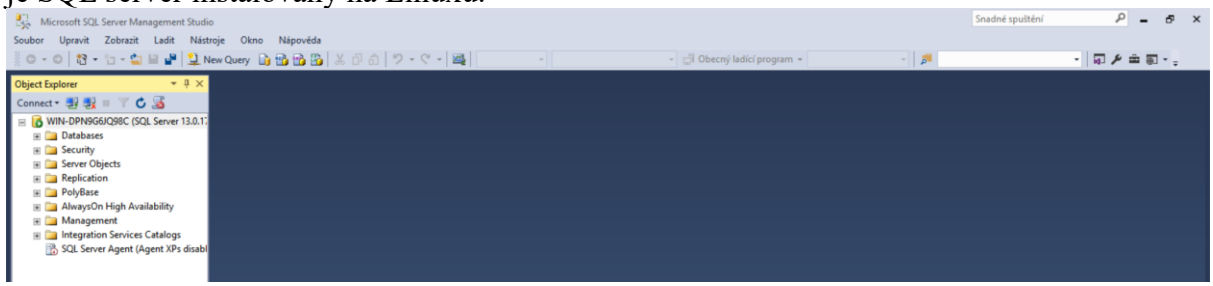
Karta tabulka slouží k vydefinování trigerů, vypisování a úpravám vlastností a relací.

20.2.2. Microsoft Management studio 2016

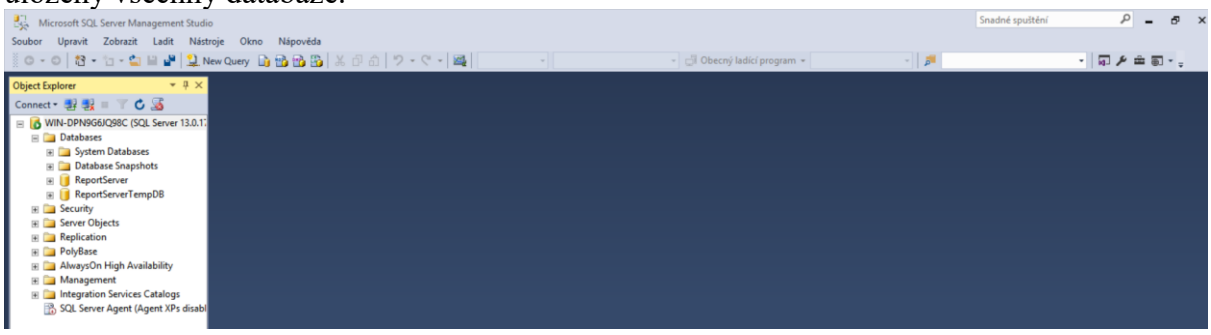
Pro práci s SQL serverem je vhodnější používat MS Management studio, které je specializovanou a odlehčenou verzí Visual studia, ale kdo je programátor může použít přímo Visual studio. Na rozdíl od Accessu vyžaduje instalovaný SQL server, ke kterému se připojí. Pro připojení k SQL serveru, který se nachází na stejném počítači, nebo intranetu se zadává název počítače, pokud připojuji externí server, tak použiji IP adresu, nebo doménové jméno zařízení, kde je SQL server instalován.



Typ autentizace volíme podle toho, jakou konfiguraci má SQL server, kam se připojujeme. Nejčastější je dnes výchozí možnost Windows Authentication, ale můžeme se ne vzácně setkat s tím, že databázový server požaduje vlastní ověření uživatele, typické to je především pokud je SQL server instalovaný na Linuxu.



Protože Managemet studio je komplexní nástroj pro správu SQL serveru, tak v průzkumníku máme vypsanou adresářovou strukturu celého serveru. Ve složce Databases jsou uloženy všechny databáze.

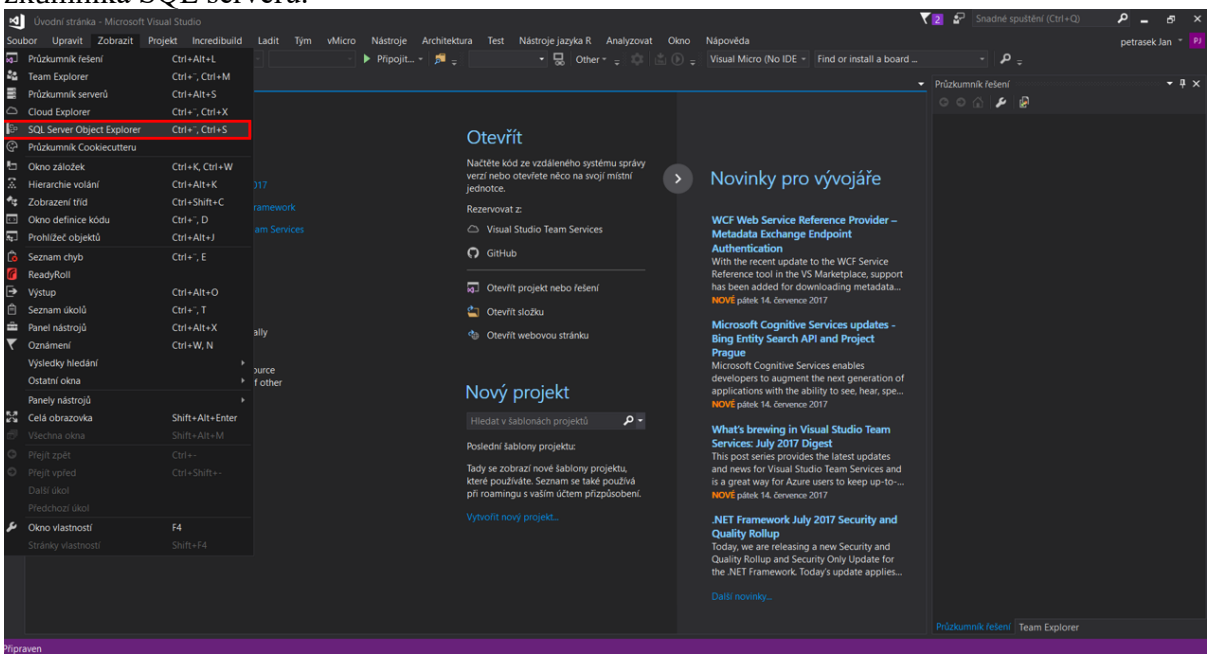


Grafické nástroje jsou v tomto prostředí značně omezeny, základ práce tvoří Query v jazyce SQL.

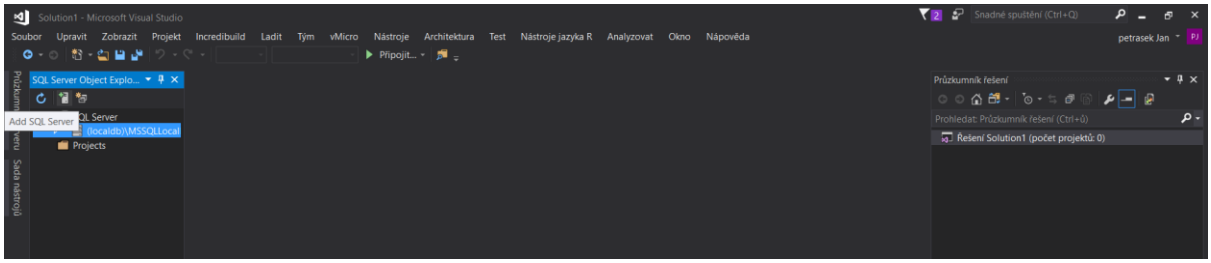
20.2.3. Připojení se k SQL serveru pomocí Visual studia

Visual studio je další z možností, jak pracovat s SQL serverem. Je zde možné vytvářet databázové projekty a podobně, ale to nechám případným zájemcům k samostudiu. Nyní si jen ukážeme, jak se dá připojit se SQL serveru a procházet jej. Hned na úvod bych měl říct, že Visual studio lze použít, pokud byla při instalaci zaškrtnuta podpora SQL, pokud ne, je nutné ji pomocí instalátoru přidat, ale stojí to za to.

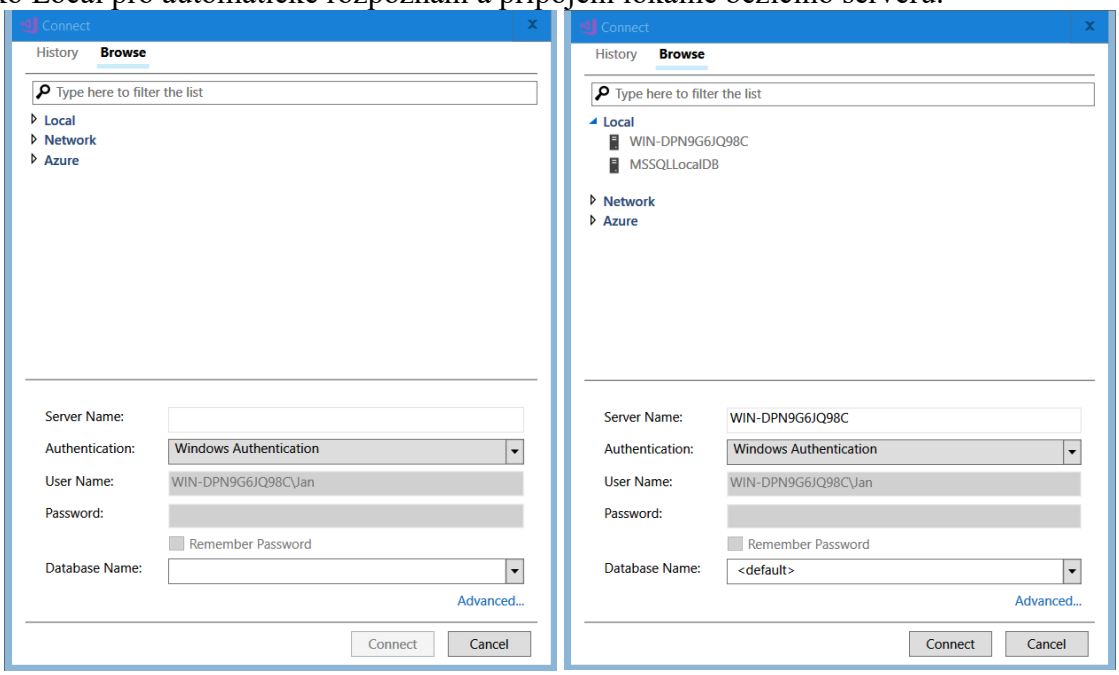
Po spuštění Visual studia můžeme, úplně bez tvorby projektu, je možné si otevřít průzkumníka SQL serveru.



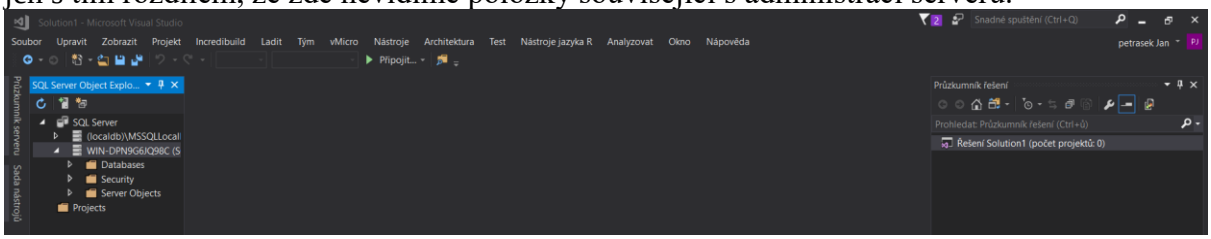
Nyní si po otevření průzkumníka připojíme požadovaný SQL serve pomocí tlačítka Add SQL Server.



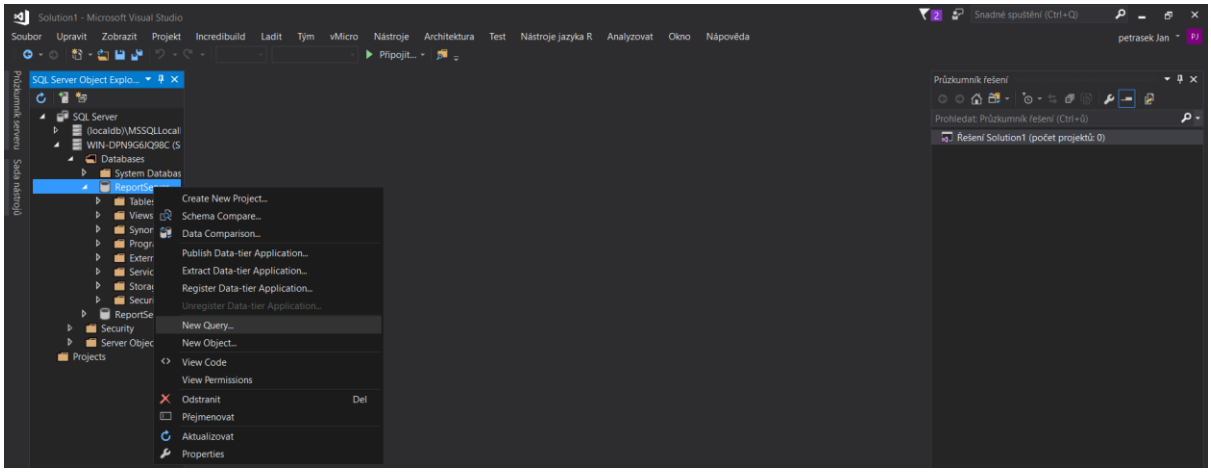
Obdobně jako v Management Studiu vyplníme požadované údaje o serveru. Výhodou je tlačítko Local pro automatické rozpoznání a připojení lokálně běžícího serveru.



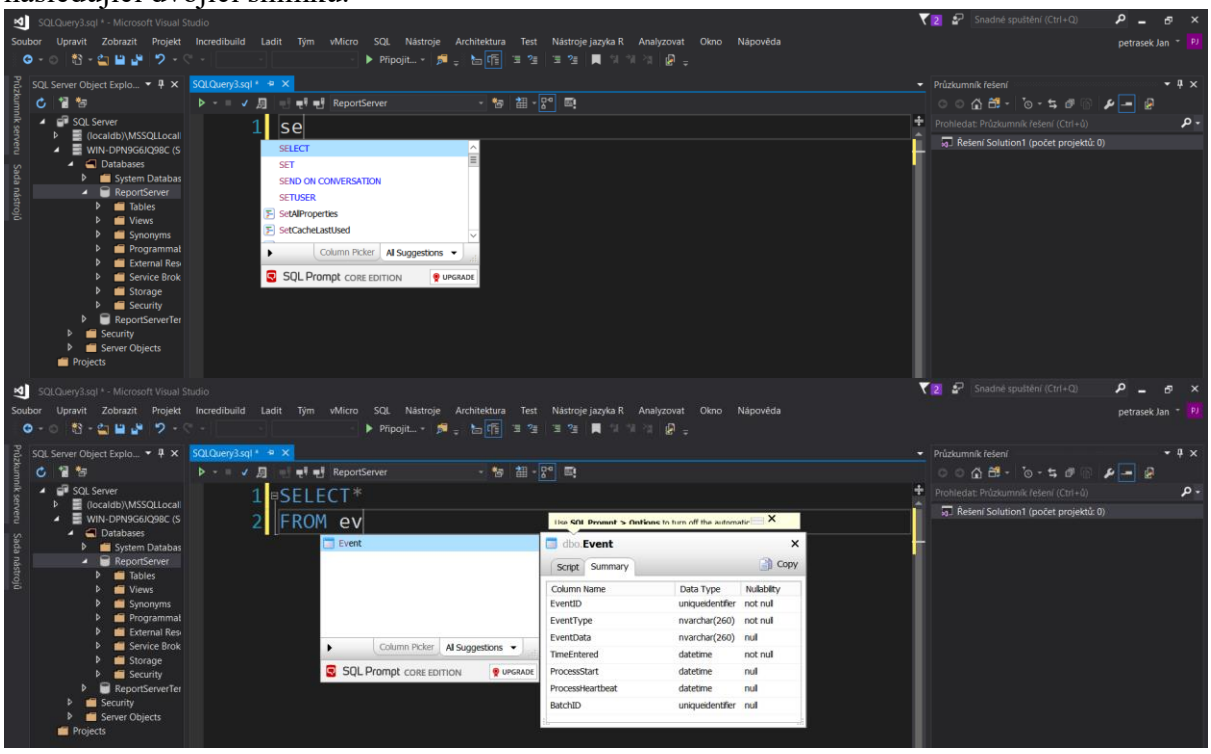
Následně vidíme v průzkumníku obsah serveru, obdobně jako v Management studiu, jen s tím rozdílem, že zde nevidíme položky související s administrací serveru.



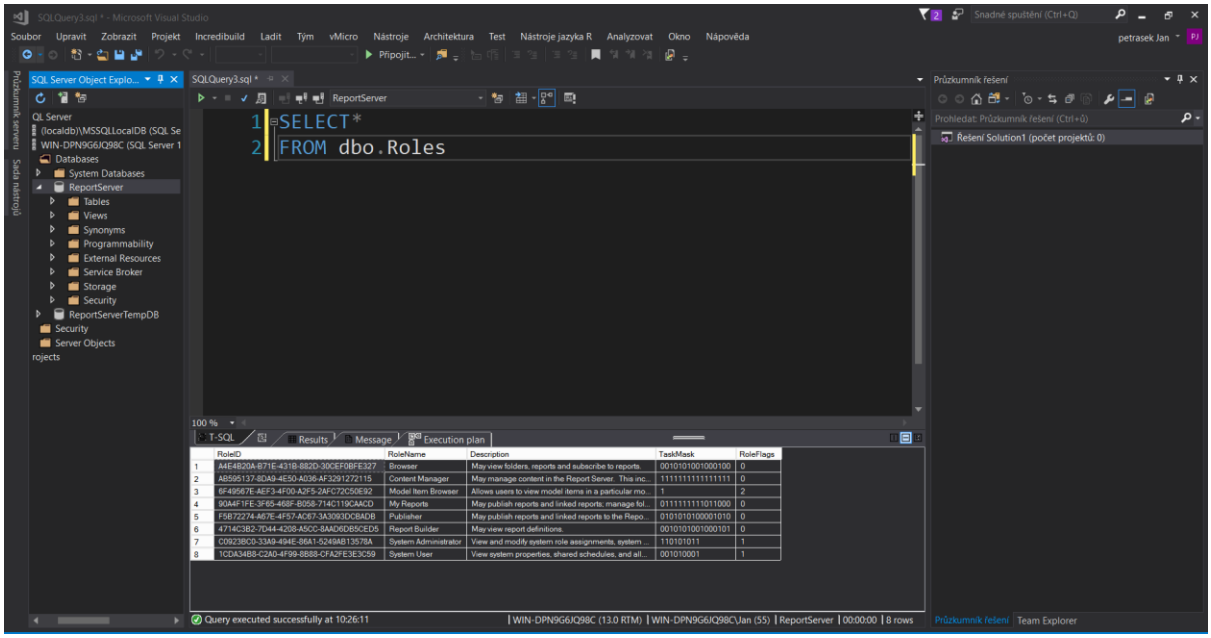
Po kliknutím pravým tlačítkem na databázi se nám rozbalí všechny možné činnosti, které můžeme v tuto chvíli s databází dělat.



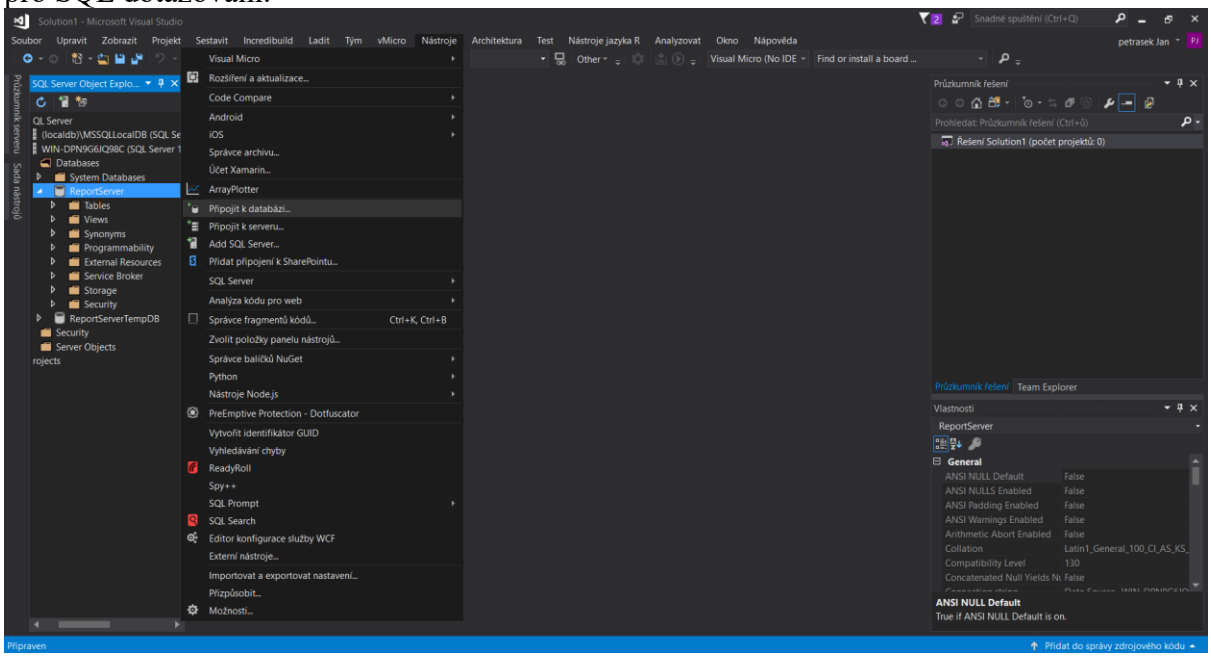
Těm, kteří máte Visual studio 2017, nebo 2015 s instalovanými databázovými nástroji doporučuji možná pro začátek využívat pro práci s příkazy v jazyce SQL právě Visual studio, protože krásně napovídá příkazy a obsahy tabulek, se kterými chceme pracovat, jak je vidět na následující dvojici snímků.



Dokončený dotaz se pouští zelenou šipečkou pod názvem souboru.



Na kartě Nástroje se ukrývá ještě možnost připojit se rovnou ke konkrétní databázi. Tato možnost je vhodná především pro vývoj aplikační logiky nad nějakou databází, ale je použitelná i pro SQL dotazování.



Tolik zhruba k úvodu do používání Visual Studia pro práci s databázemi. Jak je vidět, dnes kdo má Visual Studio, tak již nepotřebuje mít instalované Management studio, pokud není databázovým administrátorem.

20.3. Tvorba databází

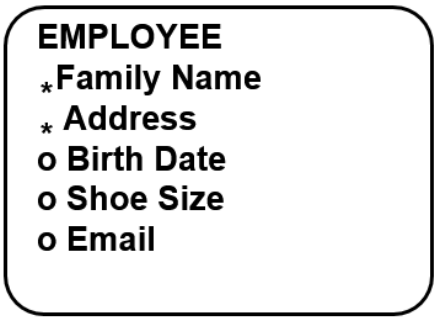
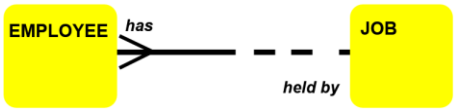
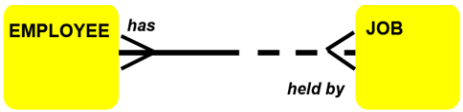
Databáze se tvoří pomocí datových modelů, které navrhuje analytik na základě toho, jaká data budou v databázi uložena a jaké mají mezi sebou souvislosti. Datový model popisuje realitu na straně zákazníka zjednodušeně, ale s dostatečnou přesností. Datových modelů existuje celá řada, ale Entitně relační model je použitelný pro libovolný typ databáze, což z něj

činní nejpoužívanější datový model. Kdo je trochu zběhlý v OOP, nebo pořádně studoval kapitolu o Javě, tak se setkal s class diagramem, což je obdobou Entitně relačního modelu v programování.

V relačních databázích je onou relací tabulka, nikoliv vztah mezi tabulkami. Entita představuje množinu instancí (objektů) s podobnými vlastnostmi. V zadání od zákazníka nejčastěji entity představují podstatná jména, což je jednou z pomůcek pro analytika, když se snaží identifikovat jednotlivé entity. Atribut je specifickým znakem entity, její určitou specifickou vlastností. Atributy se dělí na povinné a nepovinné, pokud je před atributem znak #, jde o identifikační atribut, pokud se před atributem nachází znak *, je atribut povinný a pokud se před atributem nachází znak „o“, jde o atribut nepovinný. Z jiného úhlu pohledu dělíme atributy na atomické a vícehodnotové. Pokud je to možné, tak při tvorbě datového modelu upřednostňujeme atomické atributy, pokud potřebujeme použít atribut složený, tak jej nejčastěji realizujeme jako entitu, která má vydefinované vztahy sama se sebou.

Vztah (relace) se definuje obvykle mezi dvěma entitami, ale entita může mít vztah i sama se sebou. Vztahy je potřeba číst z obou stran, protože se dle toho určují vlastnosti vztahu vzhledem k dané entitě. Vztahy jsou kardinální, tedy mají určitou mohutnost a rozlišujeme následující kardinality, 1 k 1 a 1 k N. u vztahů dále definujeme parcialitu (povinnost entity účastnit se vztahu), která nabývá dvou hodnot, a to povinný a nepovinný vztah k druhé entitě.

Při návrhu datového modelu si musíme dát pozor, abychom se vyvarovali redundancí (nadbytečnosti). Pokud máme abstraktní entitu, tak k ní netvoříme fyzickou tabulku a při návrhu databáze určité instituce se daná instituce neobjevuje jako entita.

Značka E-R diagramu	Význam
	Entita se svými atributy
	Vztah 1 k 1 povinný
	Vztah 1 k 1 nepovinný
	Vztah N k 1 povinný pro entitu Employee a nepovinný pro entitu Job. Jeden Job může mít více zaměstnanců, ale jeden zaměstnanec má jeden Job.
	Vztah N k M povinný pro entitu Employee a nepovinný pro entitu Job. Jeden Job může mít více zaměstnanců, ale jeden zaměstnanec má více Job.

20.3.1. Relační model

Je založen na teorii relační algebry a stojí za ním společnost IBM a byl publikován roku 1970 E.F. Coddem. Sledované "věci" jsou databázové entity, které se definují jako to, co je pro uživatele důležité a je potřeba reprezentovat v databázi. Relace jsou speciálním typem tabulek, který se definuje se jako dvojrozměrná tabulka skládající se z řádků a sloupců, která se vyznačuje následujícími vlastnostmi:

- Každý řádek tabulky obsahuje data týkající se nějaké entity nebo části této entity
- Každý sloupec tabulky obsahuje data, která reprezentují atribut entity
- Buňky v tabulce musí uchovávat jedinou hodnotu a nesmějí obsahovat opakující se prvky
- Všechny položky v každém sloupci musí být stejného druhu
- Všechny sloupce musí mít jedinečný název
- Na pořadí sloupců v tabulce nezáleží
- Nezáleží ani na pořadí řádků
- Sada datových hodnot na každém řádku musí být jedinečná

V praxi se běžně zaměňují termíny tabulka a relace, ale tabulkou se myslí automaticky tabulka splňující podmínky pro to, aby byla relací. Pro řádky tabulky se můžeme setkat s výrazem n-tice a sloupce tabulky obsahují jednotlivé atributy dané entity. V praxi ve výchozím nastavení systémy řízení databází neprovádějí kontrolu na unikátnost z důvodu úspory času a výkonu, ale i tak tabulku označujeme jako relaci.

Funkční závislost udává závislost mezi dvěma položkami a zapisuje se $x \rightarrow y$, což znamená, že x určuje y. Funkční závislosti hledáme v rámci relací, ale ne vždy musí mít funkční závislost matematický popis, ačkoliv je to zvykem. V literatuře se můžeme setkat s označením hodnoty x jako determinantu. Principy funkčních závislostí a determinantů pomáhají při návrhu relací.

Normalizaci definujeme jako proces, kdy tabulku či relaci s více tématy rozdělujeme na sadu tabulek tak, aby každá z nich obsahovala pouze jediné téma. Proces normalizace probíhá nějak následovně. Identifikujeme všechny kandidátní klíče a identifikujeme všechny funkční závislosti v relaci. Prozkoumáme determinanty funkčních závislostí, jestliže některý determinant není kandidátním klíčem, relace není ve správném formátu a provedeme:

- Umístíme sloupce funkční závislosti do nové samostatné relace
- Nastavíme determinant funkční závislosti jako primární klíč nové relace
- Ponecháme kopii determinantu v původní tabulce jako cizí klíč
- Vytvoříme omezení referenční integrity mezi původní a novou relací
- Provedeme proces znovu, kolikrát je potřeba, dokud nejsou všechny determinanty každé relace kandidátními klíči

Máme-li relaci ve správném formátu, tak je každý determinant kandidátním klíčem. Každou relaci, která není ve správném formátu je potřeba rozdělit na dvě nebo více relací.

20.3.2. Klíče

Již výše jsme se v textu zmínili o kandidátním klíči, proto je na místě si zde povědět něco o klíčích v databázích. Klíč je jeden nebo více sloupců v relaci, který umožňuje identifikovat řádek. Obecně může (ale nemusí) být jedinečný, jedinečný klíč definuje jeden konkrétní řádek, nejedinečný klíč definuje skupinu řádků. Super klíčů může být v tabulce obecně n, ale klíč je minimální super klíč a klíčů může být více v jedné tabulce.

Složené klíče, jak již název napovídá, obsahují dva či víc atributů, jejich kombinace poskytuje jedinečný (nejedinečný) identifikátor. Používání složených klíčů je zcela běžné, na rozdíl od atributů, kde se tomu snažíme vyhnout.

Kandidátní klíč jedinečně identifikuje řádek v relaci. Kandidátní klíče mohou být jedno, nebo více, atributové. Definují se jako jeden (nebo více) atributů, které funkčně určují všechny ostatní atributy relace, nesmí obsahovat hodnotu null, musí být časově neměnný a musí být minimální.

Primární klíč je kandidátní klíč, který je zvolen jako klíč, podle nějž systém řízení databáze identifikuje řádky relace. Na základě primárního klíče se realizuje dělení tabulek do bloků na disku a primární klíč slouží ve výchozím nastavení jako clustered index. Definuje se pomocí SQL, nebo s využitím nástroje pro práci s databází a je pouze jeden v rámci tabulky. Doporučení je vytvářet tyto klíče krádké a umělé (ideálně číslený typu int nebo guid). Typicky tímto klíčem je identifikátor ID, který se generuje automaticky pomocí autoinkrementace.

Náhradní klíč je sloupec s jedinečným identifikátorem přiřazený systémem řízení databáze. Sloupec s tímto klíčem si přidává návrhář databáze, pokud je PK dlouhý a není číselný, automaticky. Hodnoty jsou unikátní a pevně spjaté s daným řádkem, plní funkci primárního klíče a často jsou skryty před uživatelem. Jeho potlačení je, když je primární klíč označen jako identity (pro MS SQL). Pokud vlastností identity označíme jiný nežli primární klíč, tak je automaticky použit jako náhradní klíč.

Cizí klíč je atribut druhé relace uchovávající hodnoty PK první relace. Názvy sloupců primárního a cizího klíče se nemusejí shodovat, ale primární a cizí klíč musí mít shodný datový typ a shodnou sadu hodnot.

Pro propojení relací je nezbytné, aby tabulky měly referenční data v pořádku. Pro cizí klíče je dobré nastavit omezení referenční integrity tak, že hodnoty určitého sloupce z jedné tabulky musí existovat v určitém sloupci druhé tabulky, např.: Department in EMPLOYEE must exist in DepartmentName in DEPARTMENT. Pokud je PK více hodnotový, tak je potřeba nastavit více omezení pro referenční integritu.

20.3.3. Datové modelování

Proces vývoje databází zahrnuje 3 hlavní fáze: Analýzu požadavků, návrh komponent a implementaci. Ve fázi analýzy požadavků se dotazujeme uživatelů systému a získáváme ukázkové formuláře a sestavy s popisy aktualizačních aktivit. Na základě požadavků vytvoříme datový model, který reprezentuje obsah, vztahy a omezení dat, aby bylo možné splnit příslušné požadavky.

Ve fázi návrhu komponent se převádí datový model na databázový návrh. Databázový návrh obsahuje tabulky, vztahy a omezení, jejichž součástí jsou i názvy tabulek a sloupců. Zahrnuje i datové typy, vlastnosti sloupců a popisy primárních a cizích klíčů.

Ve fázi implementace konstruujeme databázi v databázovém systému a vkládáme do ní data, vytváříme nad ní dotazy a formuláře. Implementace je spojena i s psaním aplikačního SW a testováním, jak aplikace, tak databáze. V neposlední řadě je tato fáze spojena i se školením uživatelů, kompletováním dokumentace a spouštěním celého systému.

Vraťme se nyní k analýze požadavků, protože to je jediná fáze, která nebude dále blíže rozebrána. Jak již bylo zmíněno, tak v této fázi analytik komunikuje s uživateli a zjišťuje jejich požadavky, jako zdroje mu poslouží například:

- Uživatelské pohovory
- Formuláře
- Sestavy pro reporty
- Dotazy, na které má systém v datech najít odpovědi
- Případy použití
- Obchodní pravidla dané společnosti

Je potřeba zjistit stav stávajících formulářů, dotazů, sestav a zjistit potřebu případné modifikace spolu s návrhem nových. Případy použití popisují způsob, kterým uživatelé používají vlastnosti a funkce nového informačního systému (případně, co si přejí zachovat ze starého). Případy použití zahrnují popis rolí, které uživatelé zastávají při práci s novým systém a také popisy scénářů aktivit, definují vstupy zadávané do systému a výstupy generované systémem. Běžně se vytvářejí i desítky případů použití, protože jde nejen o jeden z nejlepších zdrojů informací, ale pomáhají i s ověřením datového modelu a implementace. Je nutné dokumentovat vlastnosti datových položek, pro každou položku formuláře, sestavy či dotazu je nutné určit datový typ, vlastnosti a omezení hodnot. Je nutné dokumentovat i obchodní pravidla, které stanovují omezení rozsahu databázových aktivit. Obchodní pravidla obvykle vycházejí z podnikových zásad a postupů.

20.3.4. Entity-relationship model

Je nejpoužívanější datový model, který byl publikován Peterem Chenem v roce 1976. Původní model byl rozšířen do dnešní podoby, původní verze modelu se již prakticky nepoužívá. Nejužitečnějšími prvky modelu jsou entity, atributy, identifikátory a vztahy.

Entity jsou to, co uživatelé chtějí sledovat. Entity stejného typu se seskupují do tříd entit a instance entity určité třídy entit je výskyt jisté entity. Třída entit je kolekce, která je popsána strukturou entit uvnitř sebe sama a obvykle obsahuje mnoho instancí entit. Na entity obvykle odkazuje jeden, či více formulářů nebo sestav, případně tvoří hlavní část jeden či více formulářů. Skupina formulářů (nebo i jednotlivý) může odkazovat na existenci nějaké třídy entit.

Entity mají atributy popisující vlastnosti dané entity. Všechny instance třídy entit mají stejné atributy. Atribut má datový typ a vlastnosti, které vycházejí z požadavků zjištěných při analýze. Vlastnosti určují, zda je atribut povinný, zda má výchozí hodnotu, zda jeho hodnota musí patřit do nějakého intervalu a zda má nějaká další omezení.

Instance entit mají identifikátory, které jsou atributy pojmenovávající (identifikující) jednotlivé instance. Zahrnují jeden, nebo více atributů. Identifikátory složené ze dvou či více

atributů se nazývají složené identifikátory. Identifikátory mohou být jedinečné nebo nejedinečné. Hodnota jedinečného identifikátoru identifikuje jednu, a právě jedenu instanci entity, zatím co hodnota nejedinečného identifikátoru identifikuje sadu instancí entity. Podobají se klíčům, ale liší se od nich 2 rozdíly, jsou to logické koncepce, nemusí být reprezentovány pomocí klíčů, jde o jeden nebo více atributů, které uživatelé považují za název entity. Na rozdíl od klíčů nemusí být jedinečné.

Vztahy vzájemně sdružují entity a v E-R modelu jsou obsaženy třídy vztahů a instance vztahů. Třídy vztahů jsou asociace mezi třídami entit a instance vztahů jsou asociace mezi instancemi entit. V původní specifikaci E-R modelu mohly i vztahy mít svoje atributy, dnes se to nepoužívá. Třída vztahů se může týkat mnoha tříd entit, počet tříd entit ve vztahu se označuje jako stupeň vztahu. Vztahy druhého stupně jsou nejčastější a označují se jako binární vztahy, vztahy třetího stupně nazýváme jako ternární vztahy a vztahy mezi entitami lze vyjádřit bez použití cizích klíčů. Díky nezávislosti na klíčích a celkové implementaci umožňují postup od obecného ke konkrétnímu. Správný postup návrhu je nejprve učit entity a vztahy, až poté určovat atributy. Vztahy máme:

- 1:1 (jedna instance entity souvisí s jednou instancí jiné entity)
- 1:N (jedna instance entity souvisí s mnoha instancemi jiné entity)
- N:M (mnoho instancí jedené entity souvisí s mnoha instancemi jiné entity)

Maximální kardinalita je nejvyšší počet instancí entity, které se mohou účastnit na instanci vztahu, naopak minimální kardinalita je nejmenší počet instancí entity, které se musí účastnit na instanci vztahu. Kardinalita vztahu se vyjadřuje několika způsoby, třeba přeškrtnutím čáry vztahu (entita ve vztahu musí existovat) a umístěním oválu přes čáru vztahu (entita ve vztahu může, ale nemusí být). Jestliže je minimální kardinalita ve vztahu nulová, tak to znamená, že entita se vztahu účastní dobrovolně, pokud je maximální kardinalita rovna 1, tak to znamená, že entita se vztahu účastní povinně a pokud je maximální kardinalita > 1, tak má každá entita sadu jiných entit.

E-R diagramy jsou standardizovány jen volně. Podle příslušného standardu, se třídy entit znázorňují obdélníky, vztahy se reprezentují pomocí kosočtverců, uvnitř kterých je uvedena maximální kardinalita a pomocí oválu, nebo křížku vedle entity, kde je vyjádřena minimální kardinalita. Název entity je uveden uvnitř obdélníku a název vztahu je uveden vedle kosočtverce. Takto vypadaly E-R diagramy původní varianty E-R modelu.

V současné době se užívají 3 respektive 2 varianty E-R modelu. Varianta IE byla vyvinuta Jamesem Martinem v roce 1990. Stranu vztahu s více instancemi symbolizuje pomocí "ptačího pařátu". Je snadno pochopitelný, proto je mezi vývojáři oblíbený a je vhodný pro výukové účely. Varianta IDEF1X je založená na jazyce UML. V roce 1993 byla prohlášena v USA za standart E-R model. Zahrnuje základní myšlenku E-R modelu, ale pracuje s odlišným grafickým znázorněním díky čemuž je hůře srozumitelný, ale jeho grafické vyjádření je shodné s grafickým popisem programů, takže pro čisté programátory odpadá nutnost se cokoliv nového učit. Uplatňuje se především ve vládních projektech USA. Komplikaci představuje objektová orientovanost UML, která si model přizpůsobuje a zavádí vlastní symboly. Při vytváření diagramů má velký vliv i užitý SW, který vždy zanáší drobná specifika.

Slabá entita je entita, která v databázi nemůže existovat, pokud databáze neobsahuje i jiný typ entity. Entita, která není slabá, se označuje jako silná.

ID závislá entita je speciální typ slabé entity, jejíž identifikátor obsahuje identifikátor další entity. Používá se v případě, kdy entita má složený identifikátor obsahující identifikátor jiné entity, protože nemůže sama bez druhé entity existovat. Příklad: Máme entity budova a byt. Budova má jako identifikátor jméno a byt má jako identifikátor jméno budovy a číslo. Byt nemůže fyzicky existovat bez budovy => byt je ID závislou entitou. ID závislé entity jsou v E-R diagramu reprezentovány rámečky se zaoblenými rohy. Vztah mezi ID závislou entitou a nadřazenou entitou se znázorňuje plnou čarou a nazývá se identifikující vztah. Vztah obsahující

přerušovanou čáru se objevuje mezi silnými entitami a nazývá se neidentifikující, protože nezahrnuje žádné ID závislosti. Výskyt ID závislých entit je poměrně častý. Minimální kardinalita směrem k nadřazené entitě je vždy 1. To, zda má nadřazená entita ID závislou entitu závisí na obchodních požadavcích. Instanci ID závislé entity nelze vytvořit, pokud neexistuje instance nadřazené entity a pokud dojde k odstranění nadřazené entity, tak musí dojít k odstranění všech ID závislých entit.

O ID nezávislé slabé entitě mluvíme, pokud je splněna podmínka, že bez existence nadřazené entity by nemohla existovat daná entita, ale zároveň má vlastní identifikátor nezávislý na nadřazené třídě. Vlastní definice slabé entity jako takové je nejednoznačná, ale převládá názor, že slabá entita je taková, která musí logicky záviset na jiné entitě, jinak se jedná o entitu silnou. Obdobně jako ID závislé slabé entity se označují rámečkem se zaoblenými rohy, ale s nadřazenou entitou se spojují přerušovanou čarou.

Entity podtypů jsou zavedeny v rozšířeném E-R modelu a představují speciální případ jiné entity, která se nazývá entitou nadtypu. Identifikátor nadtypu identifikuje i podtypy. Značí se jako kruh umístěný nad čarou. Každý podtyp je ID závislý na svém nadtypu => spojení plnou čarou a na koncích spojení se neuvádějí žádné symboly. Někdy atribut nadtypu vyjadřuje, který z podtypů je v danou chvíli vhodný. Atribut určující vhodný podtyp se označuje jako diskriminátor a tam, kde chybí diskriminátor, jeho funkci musí převzít aplikační vrstva. U exkluzivních podtypů instance nadtypu souvisí nejvýše s jedním podtypem, zatím co u inkluzivních podtypů instance nadtypu může souviset s jedním, nebo více podtypy a nemá diskriminátor. Pro exkluzivní podtypy se do kruhu vkládá symbol X. Podtypy v datovém modelu zajišťují, aby se nevyskytovaly nevhodné hodnoty null. Vztahy spojující nadtyp a podtypy se označují IS-A. Nadtyp a jeho podtypy reprezentují různé aspekty stejné entity.

Pokud při tvorbě modelu chybí data, není nutné modelování přerušit, ale je potřebné si zapsat, kde data chybí a jaká data by to měla být. V průběhu modelování je potřeba chybějící data doplňovat chybějící data a model korigovat dle nově získaných dat. Častým úkazem je doplňování dalších atributů v průběhu modelování. Vynucování obchodních pravidel často vyžaduje užití aplikační logiky, je poměrně velké množství obchodních pravidel, které nezvládne vynucovat přímo databáze, proto nemusíme věšet hlavu, když se nám nedaří všechna obchodní pravidla zahrnout do modelu.

Ověření datového modelu se obvykle provádí konfrontací uživatelů a modelu. Výsledný model je potřeba ověřovat po částech, nebo převést do podoby pro uživatele srozumitelné. Nad modelem je vhodné sestavit prototypy a až s těmito prototypy konfrontovat uživatele, je to pro ně jednodušší a nám to ušetří spoustu otázek, které rozptylují obě strany. Pro vytvoření prototypů je vhodné užít MS Access i v případě, že finální podoba jej nebude využívat, protože tento systém umožňuje veškerou logiku urýt před uživatelem. Datový model je nutné vyhodnotit především vůči případům použití.

20.3.5. Databázový návrh

Pro vytvoření databáze je potřeba transformovat datový model do databázového návrhu. Nejprve se vytvoří tabulka pro každou entitu v datovém modelu, následuje kontrola normalizace tabulek a posledním krokem je vytvoření vztahů mezi tabulkami.

Vytvořenou tabulku pro danou entitu příslušně pojmenujeme a jako primární klíč relace slouží identifikátor entity. Následně vytvoříme v relaci sloupec pro každý atribut entity a aplikujeme proces normalizace. Při vytváření sloupce je nutné definovat jeho vlastnosti a nesmíme opomenout nastavit datová omezení.

Pokud je ID nezávislá lze ji reprezentovat tabulkou a je potřeba nastavit patřičná omezení on delete cascade a dále ošetřit tvorbu záznamů v aplikační logice. Pokud je ID závislá, je nutné zajistit, aby v tabulce byl jak identifikátor nadřazené entity, tak její vlastní. Tabulka ID závislé entity má vždy složený primární klíč.

20.3.5.1. Normální formy

E.F. Codd definoval 3 normální formy, dnes známe až 5 normálních forem, ale běžně se používají 3 původní.

1. NF jsou shrnuté vlastnosti relace. Řádky obsahují data o entitě, zatím co sloupce obsahují data o atributech. Buňky v tabulce uchovávají jedinou hodnotu a zároveň platí, že všechny položky ve sloupci jsou stejného druhu a všechny sloupce mají jedinečný název. Na pořadí řádků i sloupců v tabulce nezáleží. A konečně žádné dva řádky nesmí obsahovat identické sady hodnot.

2. NF požaduje, aby byly splněny všechny požadavky 1. NF. Dále pak udává, že všechny neklíčové atributy jsou určeny celým primárním klíčem. Za neklíčové považujeme všechny atributy, které netvoří primární klíč a všechny neklíčové atributy musí být určeny celým klíčem, nikoli pouze jeho částí samostatně.

3. NF stanovuje, že relace musí splňovat 2. NF a nesmí existovat neklíčové atributy, které by závisely na jiném neklíčovém atributu.

Boyce-Coddova normální forma (BCNF) je splněna tehdy, když je každý determinant kandidátním klíčem. Všechny neklíčové sloupce musí záviset pouze na celém klíči a jednoznačně musí splňovat 3. NF. Při běžném návrhu se snažíme dosáhnout BCNF (pracovně se to považuje za plnou normalizaci), ale zároveň uplatňujeme přirozenou míru normalizace pro problém, není žádnou překážkou skončit u 3. NF.

4. NF slouží k odstranění vícehodnotové závislosti. Do samostatných tabulek přesouváme více hodnot vícehodnotového atributu a tabulky poté spojujeme pomocí vztahu 1:N s původní tabulkou. Byly do samostatné tabulky přesunuty vícenásobné hodnoty, které jsou určeny případnou vícehodnotovou závislostí? Pokud ano, tak je splněná 4. NF.

Doménová/klíčová normální forma (DK/NF) je splněna, pokud relace neobsahuje žádné anomálie. Byla navržena roku 1981 R. Faginem, ale nikdy se neujala mezi návrháři. Vyžaduje, aby všechna omezení datových hodnot byla logickými důsledky definic domén a klíčů. Každý determinant funkční nebo vícehodnotové závislosti musí být kandidátním klíčem. Obecně si klademe si otázku, zda tabulka splňuje BCNF a 4NF, pokud ano, tak se dá říci, že je v DK/NF.

20.3.5.2. Vztahy mezi silnými entitami

Vztah 1:1 je nejjednodušší formou binárního vztahu. Primární klíč z jedné tabulky vložíme jako cizí klíč do druhé tabulky, na který je možné nastavit omezení referenční integrity. Klíč kterékoliv tabulky je možné umístit do druhé tabulky, zde neplatí žádná omezení. Pro hodnoty cizího klíče se běžně nastavuje omezení unique.

Vztah 1:N "nadřízená" relace je na jednotkové straně vztahu a "pořízená" relace je na vícebodové straně vztahu. Klíč nadřízené tabulky umisťujeme jako cizí klíč do podřízené tabulky.

Vztah N:M je problematický. Řeší se pomocí zavedení průnikové tabulky (tzv. „kočkopes“), která reprezentuje vztah a jde o podřízenou tabulku, která je propojena s dvojicí nadřízených tabulek pomocí vztahů 1:N. Průniková tabulka nám tedy rozkládá vztah N:M na dvojici vztahů 1:N. Primárním klíčem průnikové tabulky jsou primární klíče nadřízených tabulek, z čehož je vidět, že jde o slabou, ID závislou entitu.

20.3.5.3. Ostatní vztahy mezi entitami

Pokud máme tabulku, která je ID závislá (třeba průnikovou a je doplněna o další údaje), tak nám vznikne asociační entita a vztah, který udržuje s nadřazenou tabulkou, se nazývá asociační vztah.

Pokud jde o podtypy, tak identifikátor nadtypu slouží jako primární klíč podtypu a zároveň jako cizí klíč propojující podtyp s nadtypem.

Rekurzivní vztah vnikne tak, že tabulku rozšíříme o sloupec "cizího" klíče a následně vytvoříme vztah jako by šlo o dvě rozdílné tabulky. Aby bylo možné spojení realizovat, je nutné pracovat se zástupnými symboly jmen tabulky v SQL a jedné tabulce "nelogicky" přiřadit dva zástupné symboly, zbytek dotazu již pomocí zástupných symbolů dokončíme, jako by šlo o dvojici tabulek.

20.3.5.4. Vybrané datové typy

Každý sloupec tabulky musí mít určitý datový typ. Pro ty z nás, kteří se shlédli spíše v programování a v databázích stále ještě po přečtení nudné úvodní teorie tápou, si můžeme tabulku v paměti představit jako dvojrozměrné pole, které má tu vlastnost, že každý jeho sloupec může uchovávat hodnoty jiného typu. Výčet těch nejběžnějších datových typů, které můžeme v databázích využívající MS SQL server vidět, uvádí následující tabulka.

Datový typ	Popis
Binary	Binární proměnná od 0 do 8 000 bajtů
Char	Znaková proměnná od 0 do 8 000 bajtů
DateTime	Hodnota datumu a času s 8 bajty. Rozsah od 1.1.153 do 31.12.9999 s přesností na 3 setiny sekundy
Image	Binární data s proměnnou délkou, maximální délka 2 147 483 647
Money	Peněžní hodnota 8 byjtů, s přesností na setinu měnové jednotky
Numeric(n,m)	Desetinné číslo – lze nastavit přesnost a měřítko. Rozsah od $(-10^{38} +1)$ do $(10^{38} - 1)$. Umožňuje uložit desetinné hodnoty obsahující až n hodnot, z nichž m je napravo od desetinné čárky. Desetinná čárka se neukládá
Smalldatetime	Hodnota data a času se 4 bajty. Rozsah od 1.1.1990 do 6.6.2079 s přesností na minutu
Smallint	Celé číslo se 2 bajty
Smallmaney	Peněžní hodnota se 4 bajty a přesností na desetinu měnové jednotky
Int	Celé číslo 4 bajty
Text	Text s proměnnou délkou a maximálně 2 147 483 648 znaků
Tinyint	Celé číslo s 1 bajtem
VarChar	Znaková hodnota s proměnnou délkou do 8 000 bajtů
BigInt	Celé číslo 64 bit
NVARCHAR	Znaková hodnota s proměnnou délkou do 8 000 bajtů v UTF
NCHAR	Znaková hodnota s pevnou délkou do 8 000 bajtů v UTF
DATE	Datum jako textový řetězec ve tvaru 'rrrr-mm-dd', rozsah od '0001-01-01' do '9999-12-31'
TIME	Čas jako textový řetězec ve tvaru 'hh:mm:ss'

Datový typ	Popis
Blob(M)	Rozsáhlý binární objekt, maximálně 65 535 znaků
float(n)	Číselný datový typ s plovoucí přesností, kde n je počet bitů mantisy (1-24, přesnost 7 číslic, velikost 4 byty a 25-53, přesnost 15 číslic a velikost 8 bytů).
real	Číselný datový typ s plovoucí přesností, který je definovaný jako float(24).
sql_variant	Pro uložení hodnoty jiného typu (ne text, ntext, image, timestamp, sql_variant) o max. délce do 8016 bitů.
timestamp	Datový typ automaticky generující binární čísla, jedinečná v dané databázi, používané většinou k identifikaci řádků. V tabulce smí být pouze jediný sloupec tohoto datového typu.
uniqueidentifier	Datový typ pro uložení GUID (nového pomocí NEWID funkce nebo existujícího z řetězce ve tvaru: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, například: 6F9619FF-8B86-D011-B42D-00C04FC964FF).
enum	Výčet, pouze jedna hodnota vybraná ze seznamu
set	Množina, nula nebo více hodnot zvolených ze sezanmu
bit	Binární proměnná (hodnoty true a false)
geometry	Datový typ s proměnnou délkou jazyka CLR pro ukládání dat v rovině reprezentující body, účečky a polygony v souřadném systému
geography	Datový typ s proměnnou délkou jazyka CLR pro ukládání elipsoidních dat reprezentující geografické objekty na povrchu Země
hierarchyid	Datový typ s proměnnou délkou jazyka CLR šifující informace o jednom uzlu ve stromové struktuře
xml	Speciální datový typ pro ukládání XML dat

Objekty skupiny Geospatial jsou objekty datových typů Geometry a Geography. Oba datové typy jsou vnitřně binárními řetězci a jejich bližší popis je dostupný v .NET, odkud si volají příslušné funkce pro zpracování. Obdobně jako při programování v .NET si voláme instanční metody tečkovou notací a statické metody čtyř tečkovou notací. Při zadávání se souřadnice oddělují mezerou, čárka pak odděluje jednotlivé body v rámci obrazce. Polygon musí být uzavřený obrazec, první a poslední souřadnice musí být shodné. Množiny bodů vždy uzavíráme v kulatých závorkách. Možná trochu nelogicky se body do množiny zapisují proti směru hodinových ručiček, ale je potřeba na to dát, protože jinak se nám neuloží to, co chceme. S objekty můžeme dělat množinové operace mimo doplňku.

Geometry používá Euklidovský prostor a je bezrozměrný, nastavuje se jednotka vzdálenosti. Pokud má objekt koncovku string, znamená to jen obvodovou křivku a nefunguje výpočet plochy křivkou ohraničené. Objekty tohoto datového typu mají povinný parametr SRID = 0, aby se jednoznačně odlišili od objektů typu Geography. Protože by byl nesmysl objevovat znovu již vymyšlené, tak jsou základní geometrické obrazce sdruženy do Geometrycollection, která nám s nimi umožňuje pracovat jako s hotovými objekty v parametrické podobě.

Geography používá SRID a uvažuje zakřivení zemského povrchu a řeší vzdálenosti pomocí SRID, které udává jednotky (metrické / imperiální), polohu počátku a úhlové jednotky. Převody jednotek v SRID jsou vztaženy k metrickému systému a SQL 2016 udržuje až 390 různých SRID. Pokud chceme porovnávat objekty, musí mít stejné SRID, jinak nám je databázový engine neporovná a vrátí chybu. Samotné souřadnice by nám často moc neřekly, takže můžeme používat externí mapové podklady esri.com a soubory formátu .shp, podporovány jsou také PowerMap a PowerView, ale je nutné zadávat data ve formátu pro Bing mapy.

Geotypy mají nízkou výkonnost díky vnitřní binární podobě a často je nutný crossjoin sama na sebe, aby se dosáhlo požadovaného výsledku. Server se snaží si moc vytvořením tzv. prostorového indexu, který je čtyřvrstvý rastr a vyžaduje clustered primary key. Nad každým z objektů geotypu může být až 249 indexů, ale ne vždy znamená větší množství indexů vyšší výkon, někdy to může vést k pravému opaku. Od MS SQL serveru 2012 prostorový index vytváří a udržuje automatika.

20.3.5.5. Příklad databázového návrhu

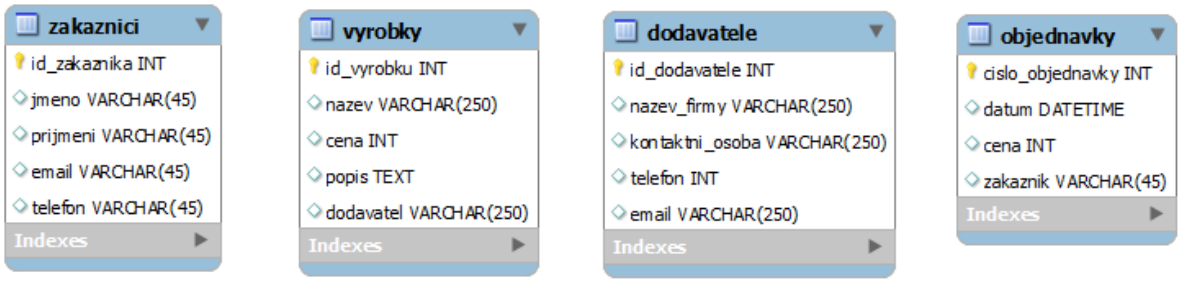
Nyní konečně od nudné teorie přejdeme k praxi a ukážeme si, jak se vytváří jednoduchá databáze. Mějme na paměti, že níže uvedený je jen školní příklad na pochopení principu tvorby, reálná databáze e-shopu je výrazně složitější! Vytvoříme si databázi pohybu zboží na e-shopu. O zboží potřebujeme vědět, kdo nám jej dodal, co je to za zboží, kdo si to objednal a co obsahovaly jednotlivé objednávky. Můžeme tedy navrhnout následující tabulky Zákazníci, Výrobky, Dodavatelé a Objednávky. Tyto tabulky potom budou obsahovat jednotlivé atributy, které je třeba k nim přidělit podle toho, co všechno chceme evidovat nebo co bude aplikace pro své fungování potřebovat. Zatím neřešíme jednotlivé vztahy, ale pouze vytváříme seznam atributů jednotlivých tabulek.

Tabulka	**Atributy**
Zákazníci	Jméno, Adresa, E-mail, Telefon
Výrobky	Název, Cena, Dodavatel, popis
Dodavatelé	Jméno firmy, Kontaktní osoba, Telefon, E-mail
Objednávky	Číslo objednávky, Datum, Celková cena

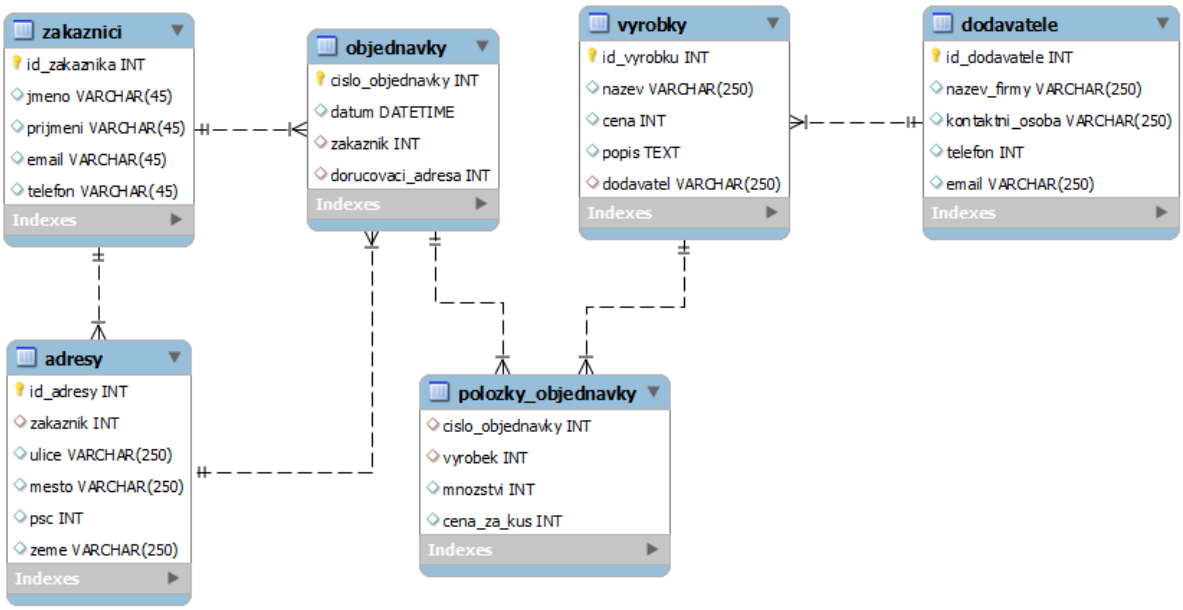
Relevantních atributů bychom pro každou z těchto tabulek vymysleli mnohem více, ale pro naše cvičení postačí výše uvedené. Musíme si dát pozor, aby se informace v tabulkách neobjevovaly duplicitně, jelikož to vede k nekonzistentnosti dat. Nastanou pak problémy při úpravě takového údaje a promítnutí změny na dalších místech. V jedné tabulce evidujte pouze atributy, které se k dané entitě vztahují!

Jednotlivé atributy budou reprezentovat sloupce tabulky. Nyní je potřeba určit, jak budou sloupce (atributy) pojmenovány a jaké sloupce doopravdy potřebujeme. U jednotlivých sloupců je pak zapotřebí určit datový typ pro vkládanou hodnotu. Data se snažíme do atributů rozdělovat do co nejmenších logických celků. Např. v tabulce Zákazníci, je vhodné rozdělit atribut jméno na dva samostatné atributy – jméno a příjmení. Stejně tak atribut adresa by měl být rozdělen na atribut ulice, město, PSČ, případně atribut země.

V další fázi určete unikátní identifikátory jednotlivých tabulek – primární klíče. V našem případě použijme umělé primární klíče datového typu INT, což si vyžádá vložení dalšího sloupce do tabulky. Po navržení pár databází již budeme mít tento postup zažitý všichni a ve více jak 90% automaticky vložíme sloupec s umělým primárním klíčem aniž bychom nad tím více uvažovali. Následující obrázek ukazuje, jak mohou vypadat tabulky pro náš příklad.



Nyní je potřeba tabulky vzájemně propojit vztahy. Je třeba si uvědomit vzájemné vazby atributů z různých tabulek a jejich vzájemnou závislost (typ vazby). Nejčastějším případem je typ vazby 1:n. Při zjištění, že náš databázový model obsahuje vazbu n:m, bude třeba vytvořit další pomocnou tabulku, jejímž obsahem budou primární klíče z dvou původních tabulek. U relací typu 1:1 zvažte, není-li výhodnější sloučit danou entitu do jedné tabulky. V našem případě jsme propojili tabulky pomocí primárních a cizích klíčů následovně. Vždy je primární klíč jedné tabulky cizím klíčem závislé tabulky.

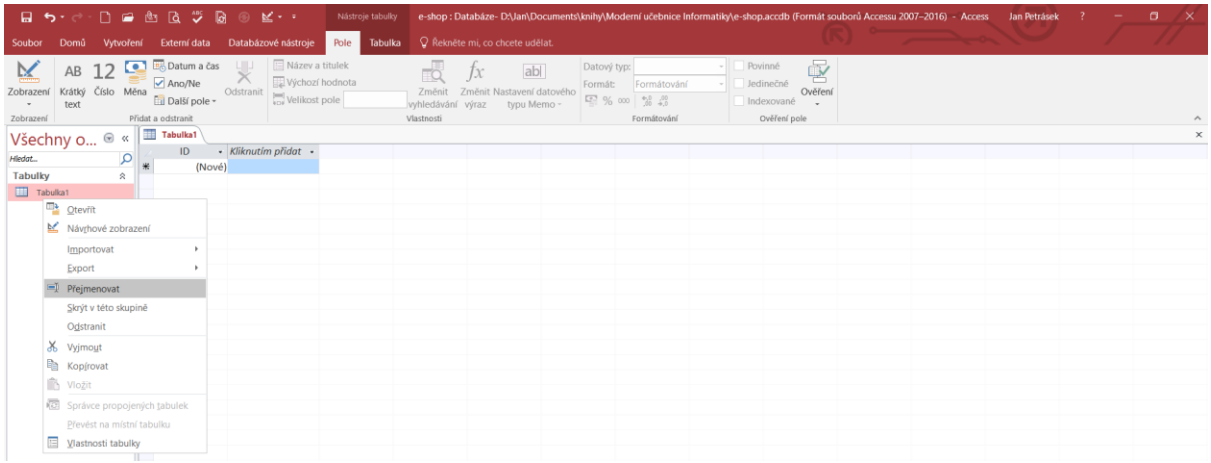


Jak je možné si všimnout, tak jsme doplnili tabulku pro adresy, která se nám rozpadá na vícehodnotový atribut a doplnili jsme též ID závislou tabulku polozky_objednavky, která nám zprostředkovává vztah N:M a zároveň doplňuje další informace o položce objednávky jako je třeba počet kusů daného zboží, či cena za kus.

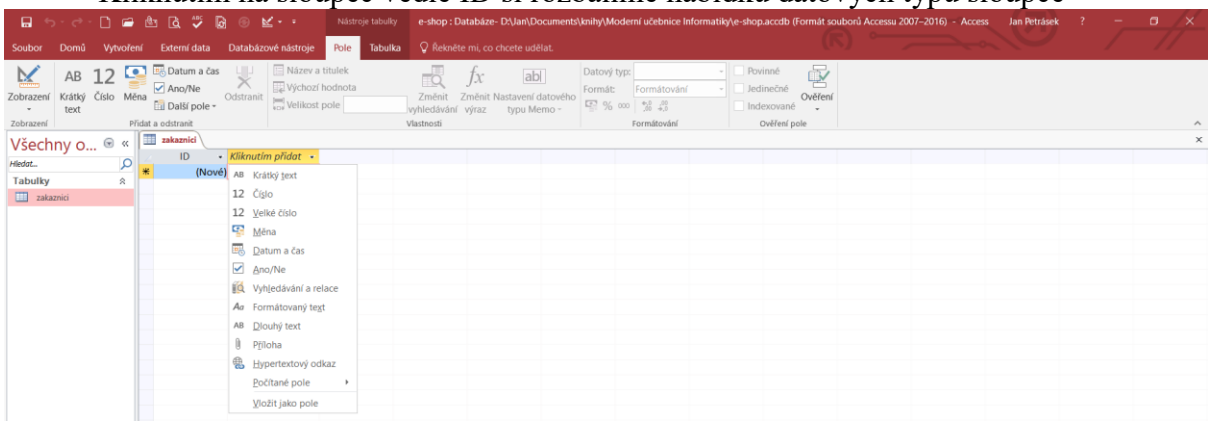
V tuto chvíli můžeme začít všechno vytvářet přímo v systému řízení báze dat. Ukážeme si vytvoření této databáze v Access a pomocí grafických nástrojů Management studia. V následující kapitole si pak ukážeme, jak databázi vydefinovat pomocí příkazů v Management studiu nebo Visual studiu.

20.3.5.5.1. Tvorba v Access

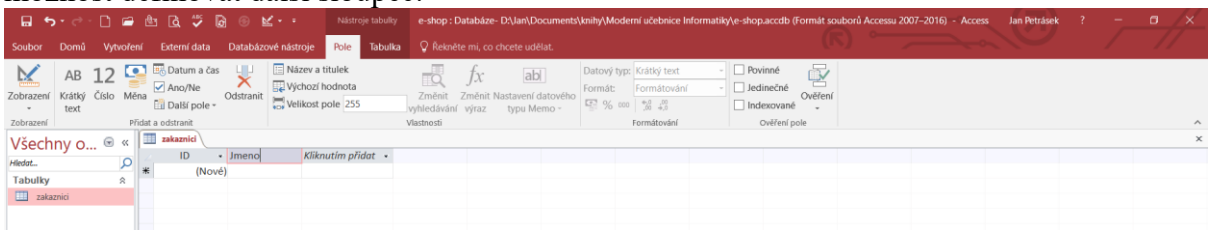
Založení databázového projektu jsme si popsali v kapitole o Access 2016, nyní v tomto projektu budeme pokračovat a vytvoříme databázi. Access nám při svém spuštění do databáze založí jednu prázdnou tabulku, nyní z ní vytvoříme tabulku Zákazníci. Kliknutím pravým tlačítkem myši na tabulku ji přejmenujeme.



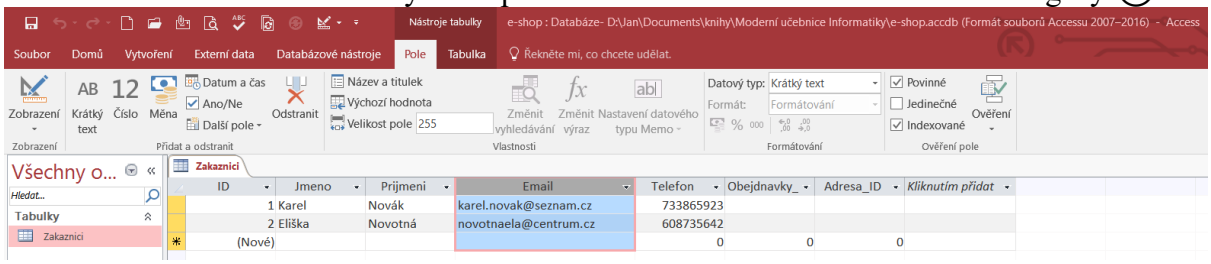
Kliknutím na sloupec vedle ID si rozbalíme nabídku datových typů sloupce



Po zvolení datového typu napíšeme název sloupce a enterem potvrdíme a zároveň otevřeme možnost definovat další sloupec.

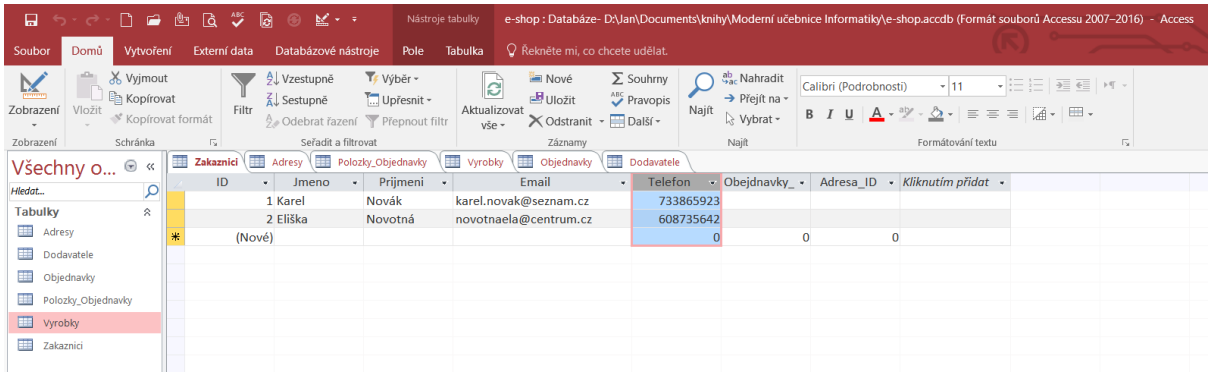


Takto vydefinujeme všechny sloupce tabulky zákazníci. Až vydefunjeme všechny sloupce, vložíme si 2 testovací zákazníky. Nezapomeňme nastavit omezení referenční integrity 😊

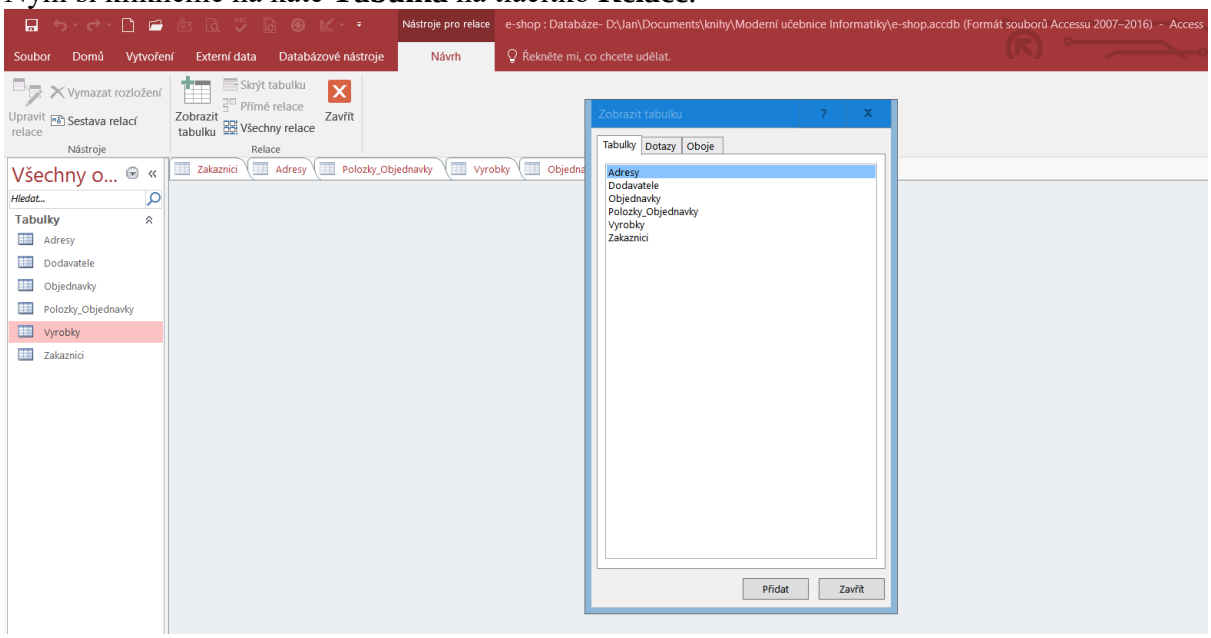


ID	Jmeno	Prijmeni	Email	Telefon	Obejdnavky_	Adresa_ID	Kliknutím přidat
1	Karel	Novák	karel.novak@seznam.cz	733865923			
2	Eliška	Novotná	novotnaela@centrum.cz	608735642			
(Nové)				0	0	0	

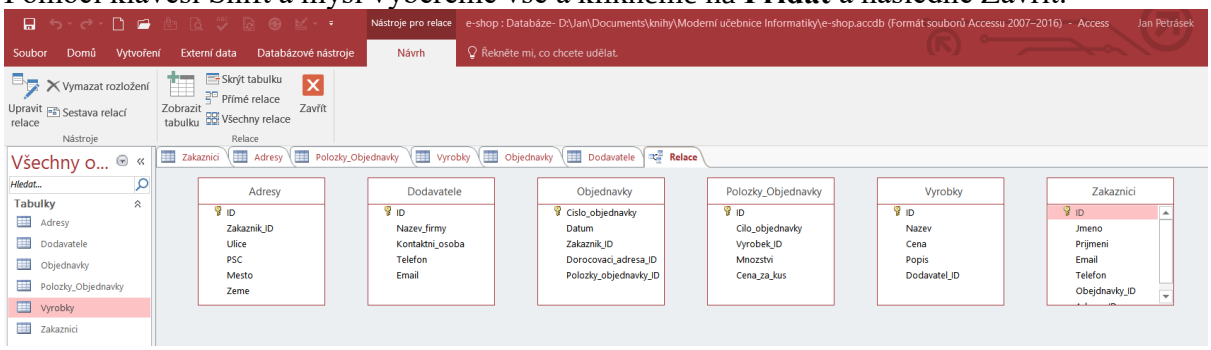
Obdobně zatím odděleně vytvoříme i další tabulky databáze.



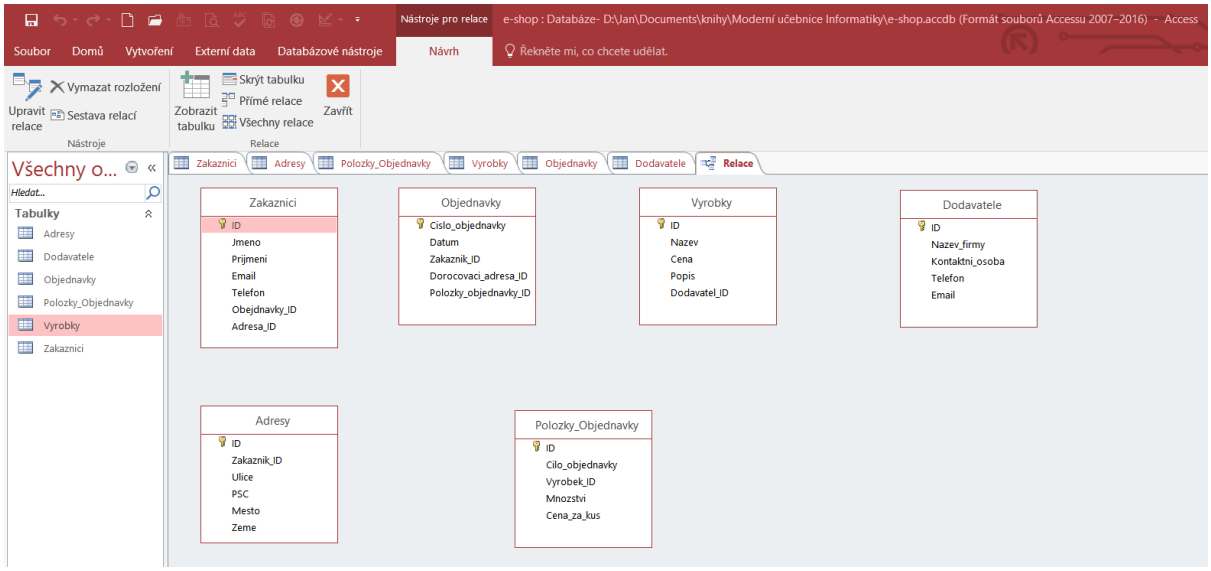
Nyní si klikneme na katě **Tabulka** na tlačítko **Relace**.



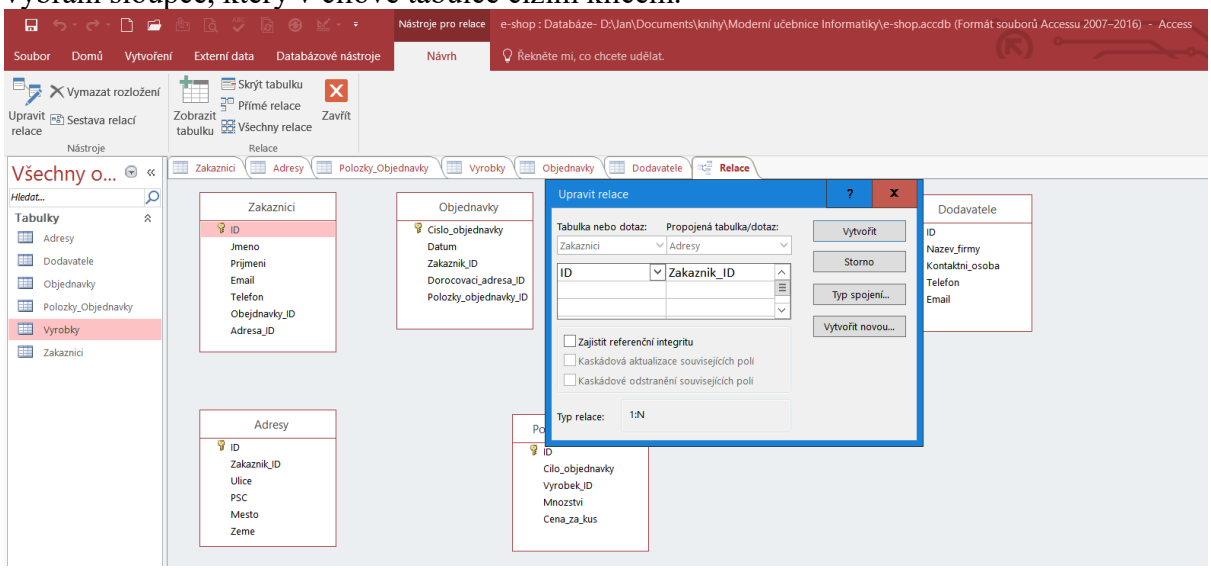
Pomocí klávesi Shift a myši vybereme vše a klikneme na **Přidat** a následně Zavřít.



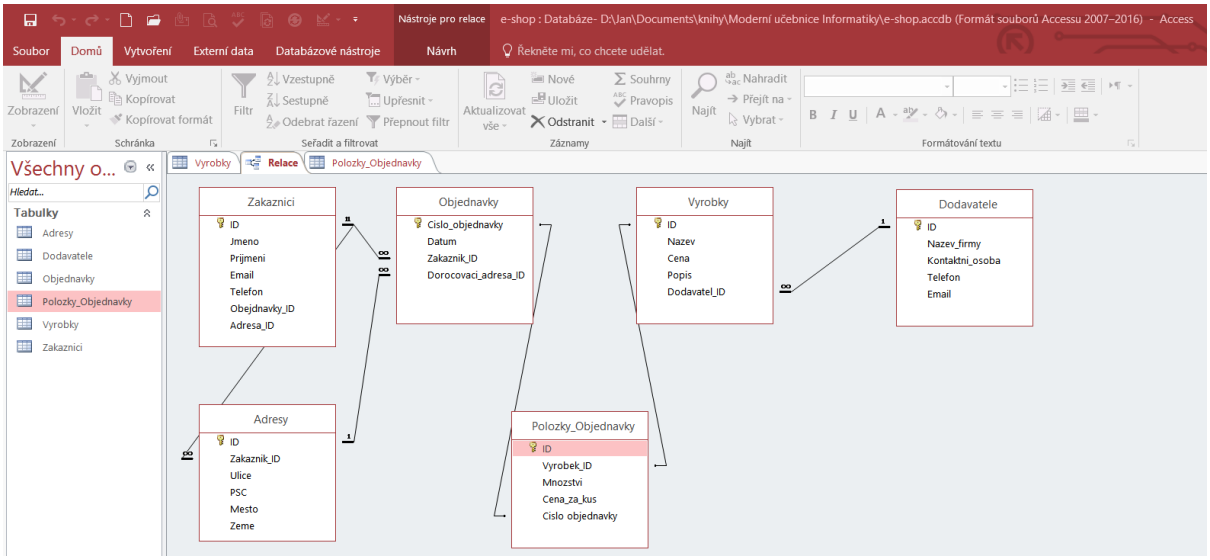
Přehledně si uspořádáme pracovní prostor.



Chytneme primární klíč jedné tabulky a přetažením na druhou tabulku vyvoláme dialog pro vybrání sloupce, který v cílové tabulce cizím klíčem.



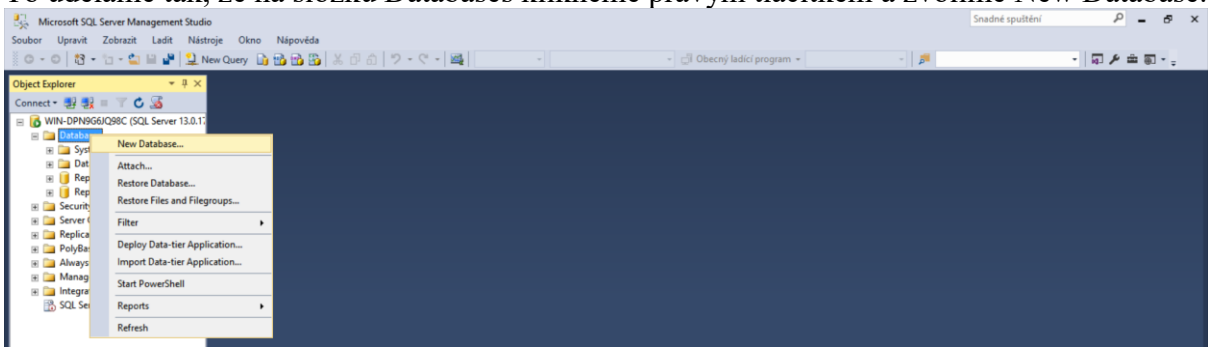
V tomto dialogu je potřeba zkontrolovat názvy sloupců, přes které má vazba vzniknout a zvážit, zda se má zaškrtnout možnost Zajistit referenční integritu a jak. Zajištění integrity doporučuji používat, ale ne vždy jde použít. Je potřeba dát si pozor, aby tabulky nezůstaly otevřené.



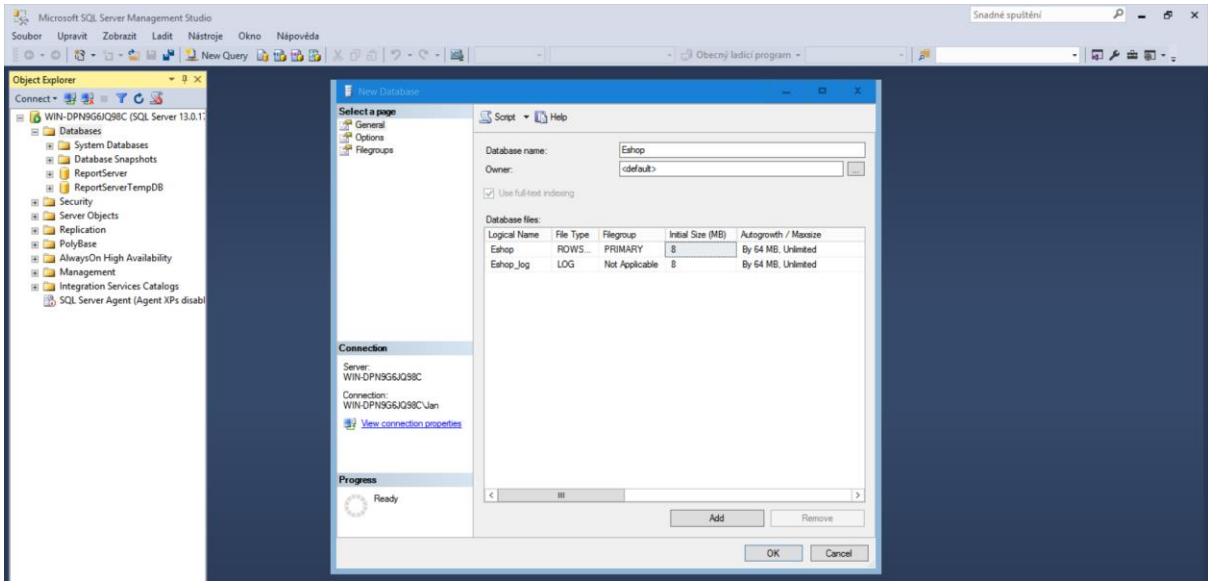
Nyní databázi doplníme daty a máme hotovo. Můžeme se dát do tvorby ověřovacích dotazů, ale to až v další kapitolce.

20.3.5.5.2. Tvorba v MS Management Studiu – grafické nástroje

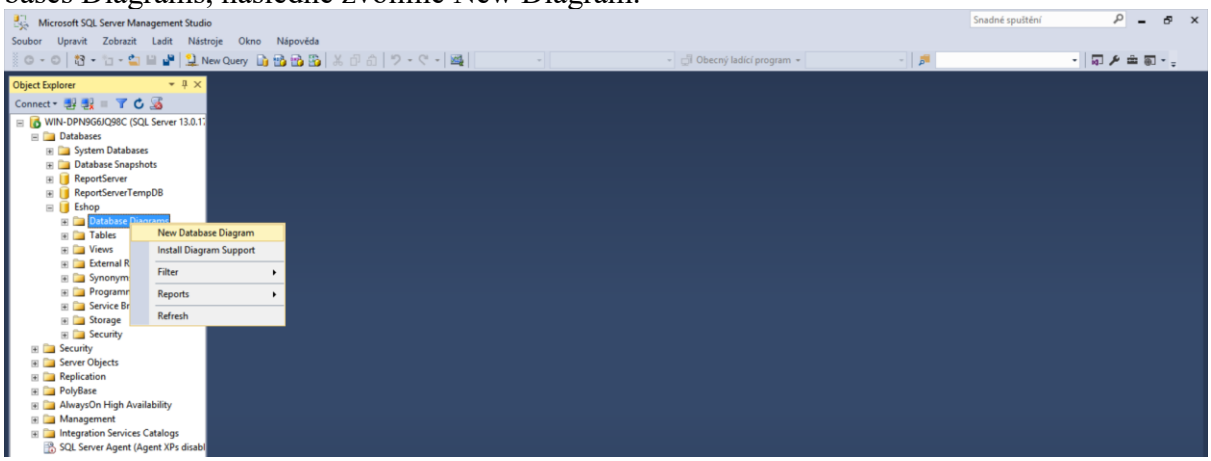
Nyní si ukážeme, jak tu samou databázi vytvořit na MS SQL serveru. Nyní ještě využijeme grafických nástrojů, takže je potřeba použít Management studio, jak to udělat pomocí Visual studia si ukážeme v kapitole o jazyce SQL. První, co musíme udělat je vytvořit databázi. To uděláme tak, že na složku Databases klikneme pravým tlačítkem a zvolíme New Database.



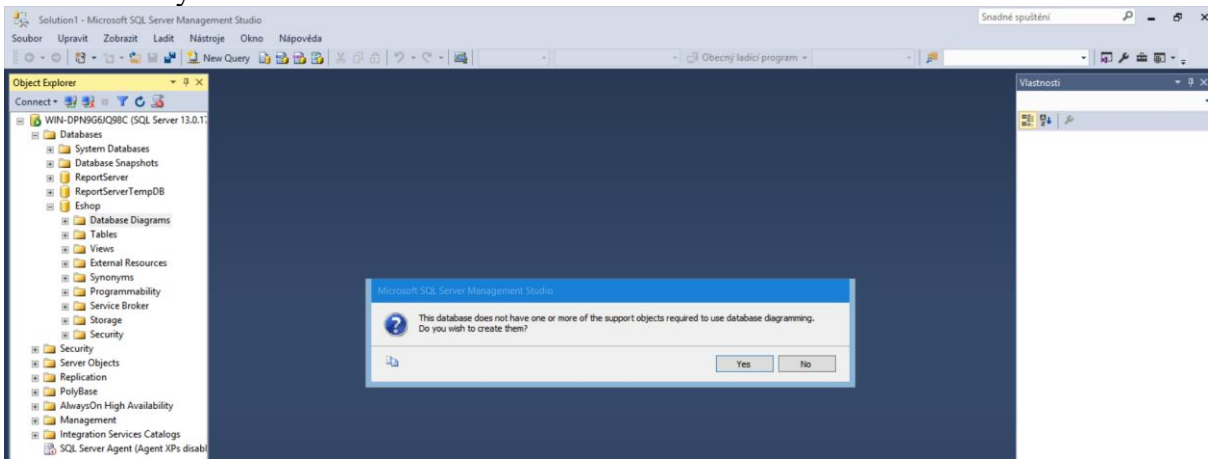
V otevřeném dialogovém okně vyplníme název databáze a u školních databází necháme vše ostatní ve výchozím stavu. Změna dalších parametrů je doporučená až pro zkušené uživatele, kteří buď absolvovali školení, nebo mají za sebou alespoň rok databází na vysoké škole.



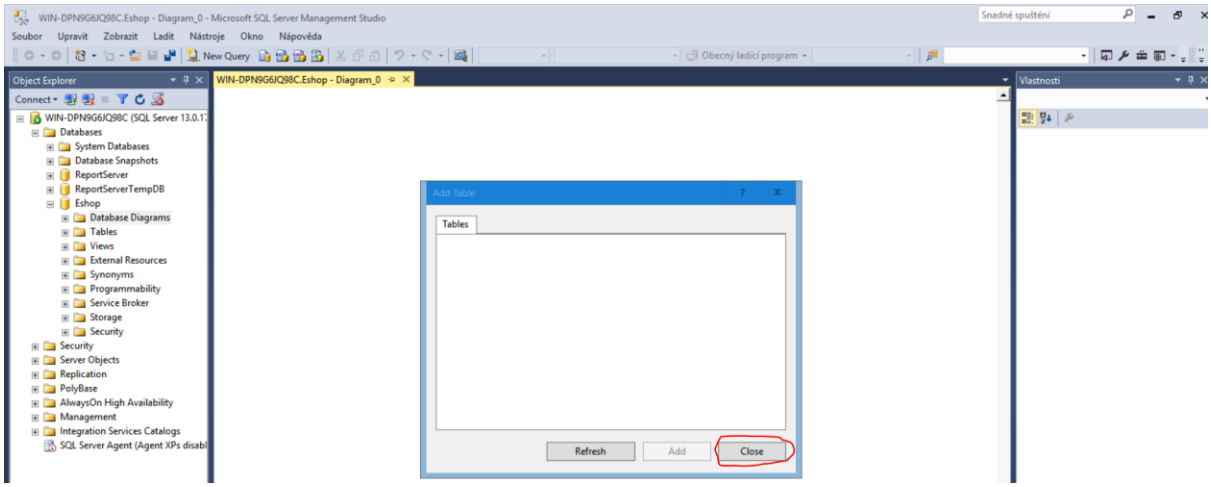
Po vytvoření databáze si ji rozbalíme v průzkumníku a klikneme pravým tlačítkem na Databases Diagrams, následně zvolíme New Diagram.



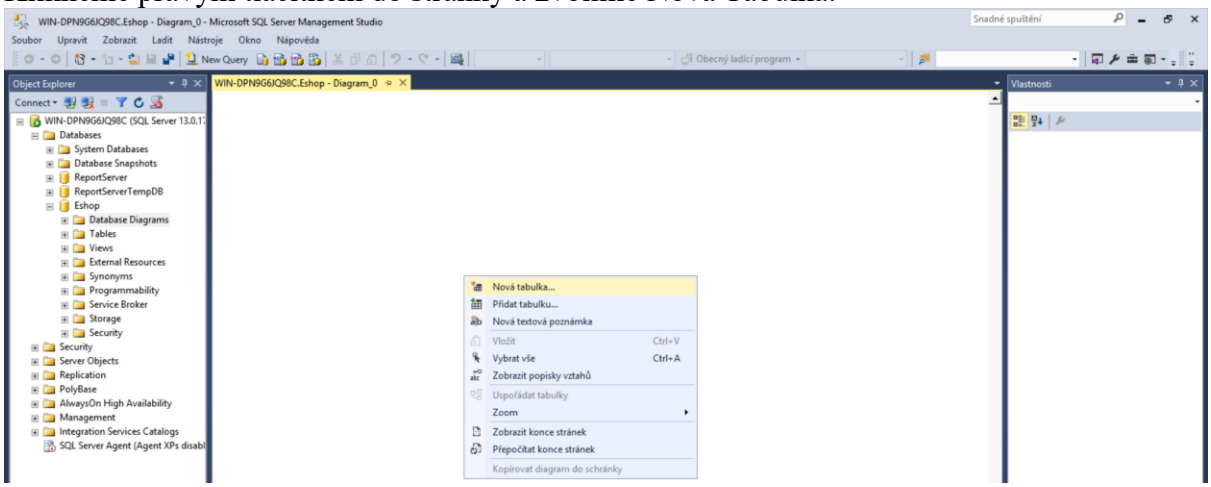
Potvrdíme chybové hlášení.



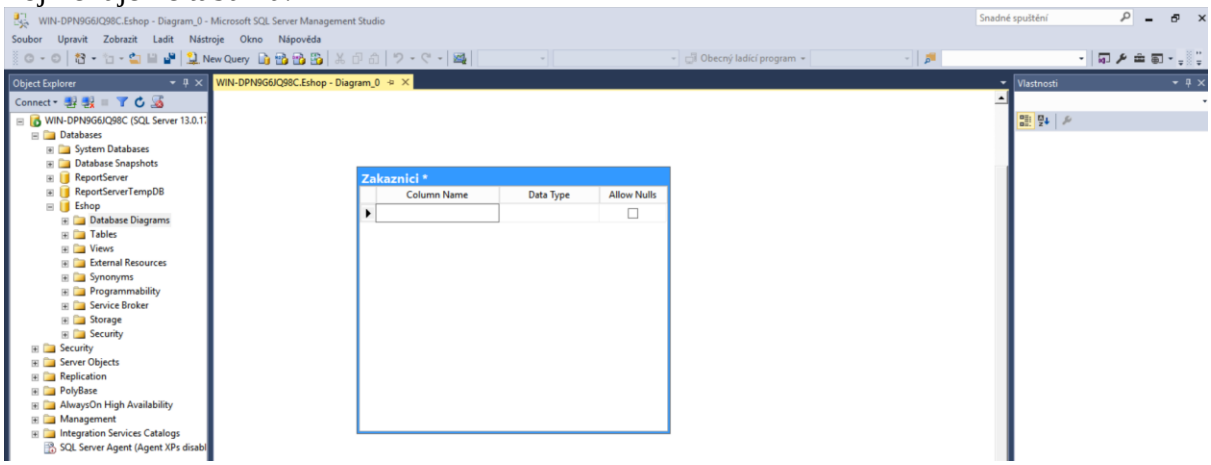
Zavřeme dialog pro přidání tabulek do diagramu, protože žádné tabulky zatím nemáme.



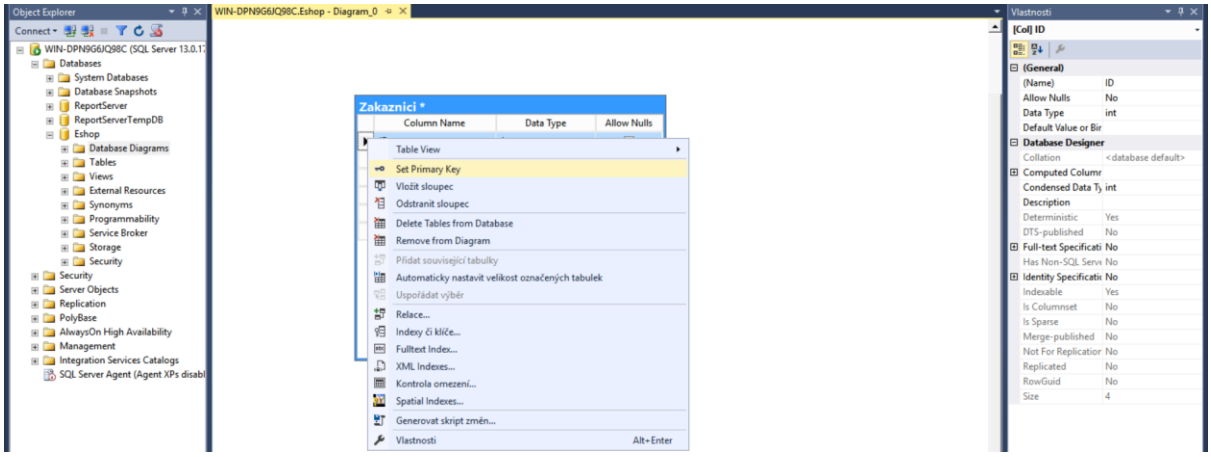
Klikneme pravým tlačítkem do stránky a zvolíme Nová Tabulka.



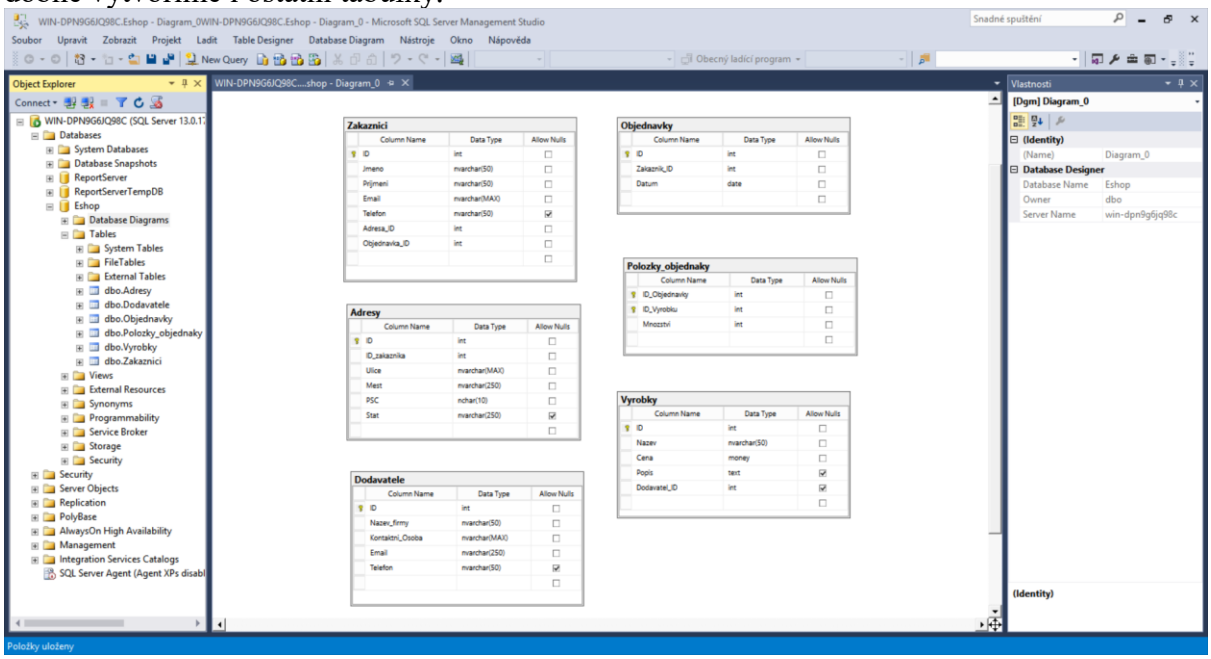
Pojmenujeme tabulku.



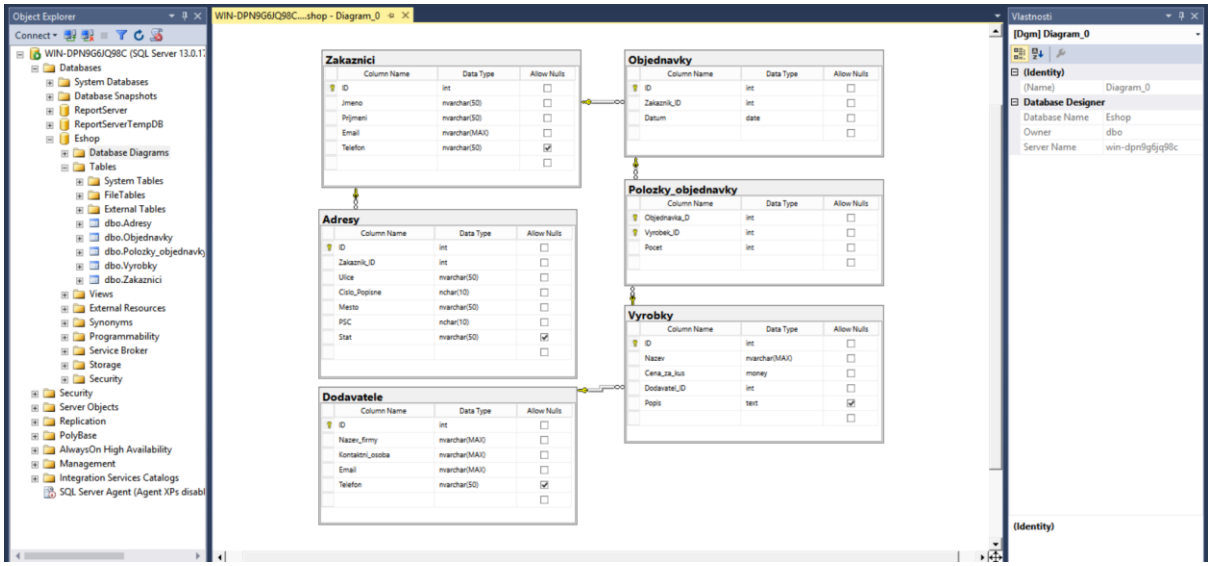
Do řádků pod sebe začneme definovat atributy. První sloupec obsahuje název atributu, v druhém volíme datový typ atributu a třetí sloupec, pokud **je zaškrtnutý**, znamená **nepovinný** atribut.
Na černou šipku v řádku s atributem ID klikneme pravým tlačítkem a nastavíme Primární klíč.



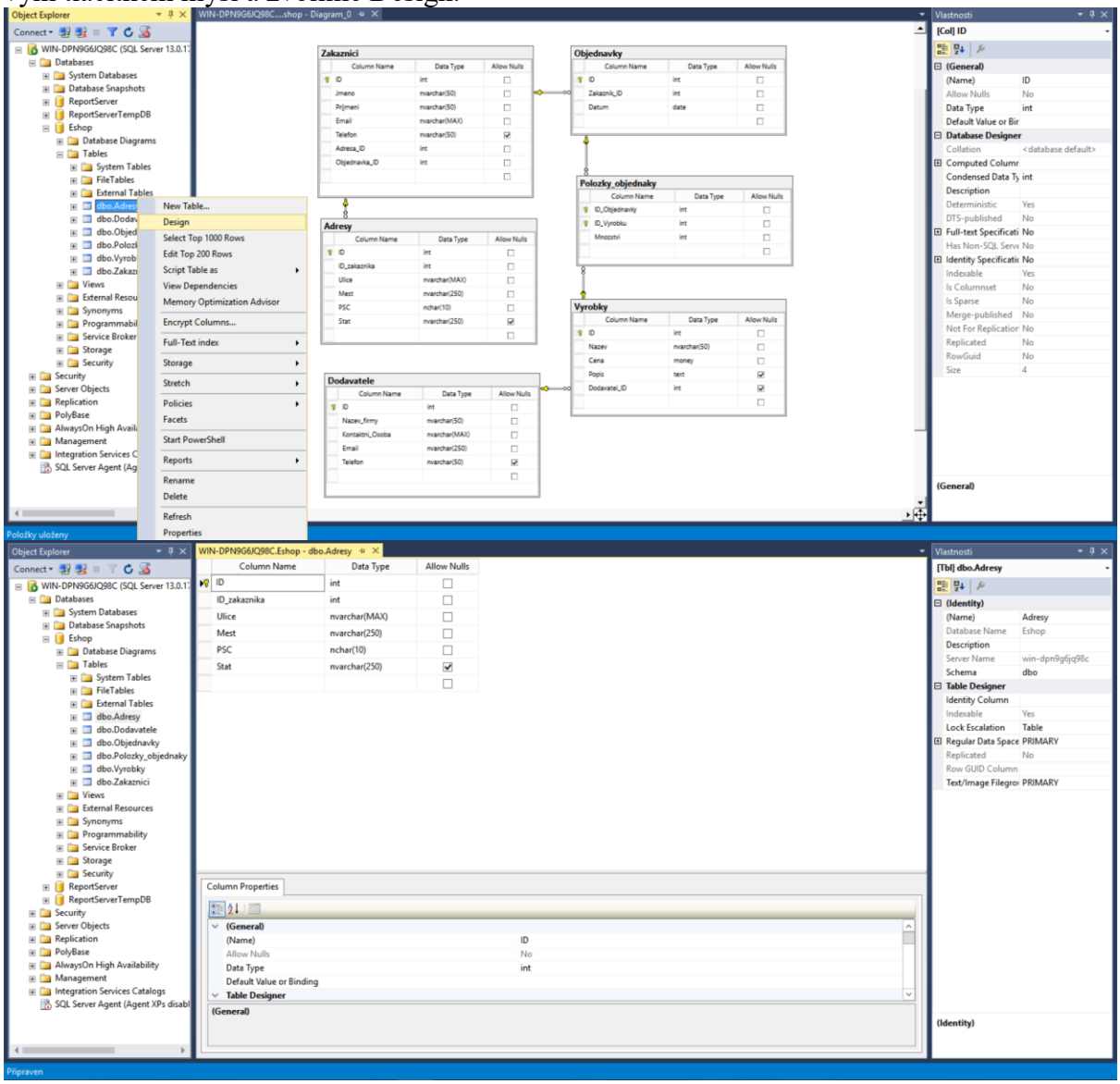
Kliknutím na diskety a pojmenováním diagramu diagram uložíme a tabulku vytvoříme. Obdobně vytvoříme i ostatní tabulky.



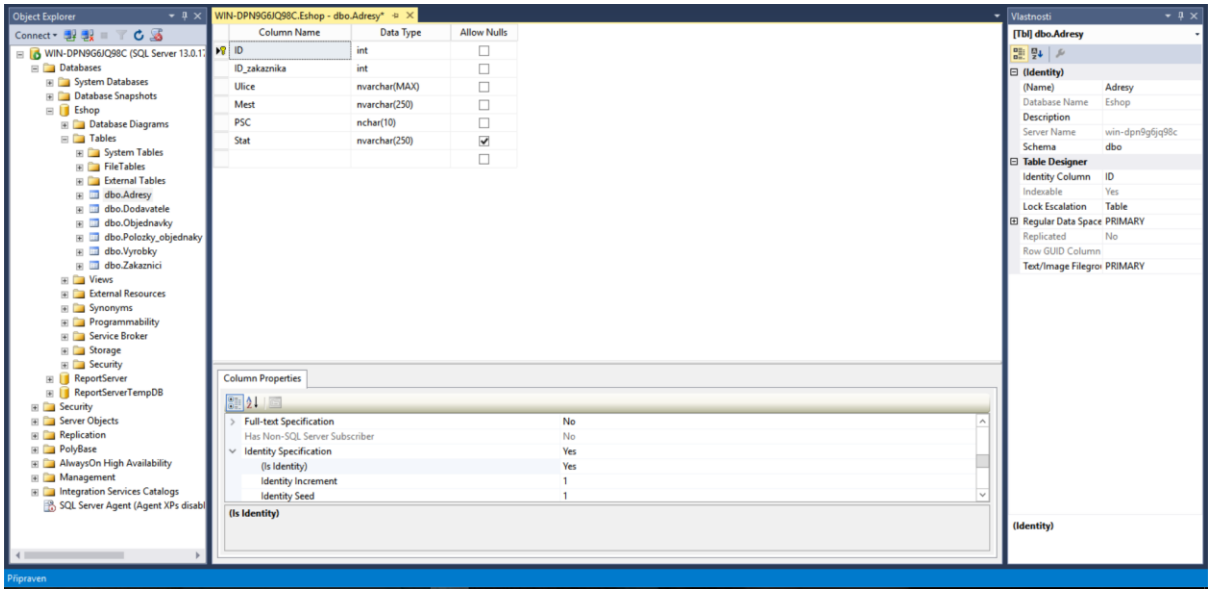
Nyní je na čase nastavit relace mezi tabulkami. Podobně jako v Access to provedeme tak, že přetáhneme primární klíč jedné tabulky na cizí klíč v druhé tabulce. Dostaneme dialog podobný tomu z Accessu.



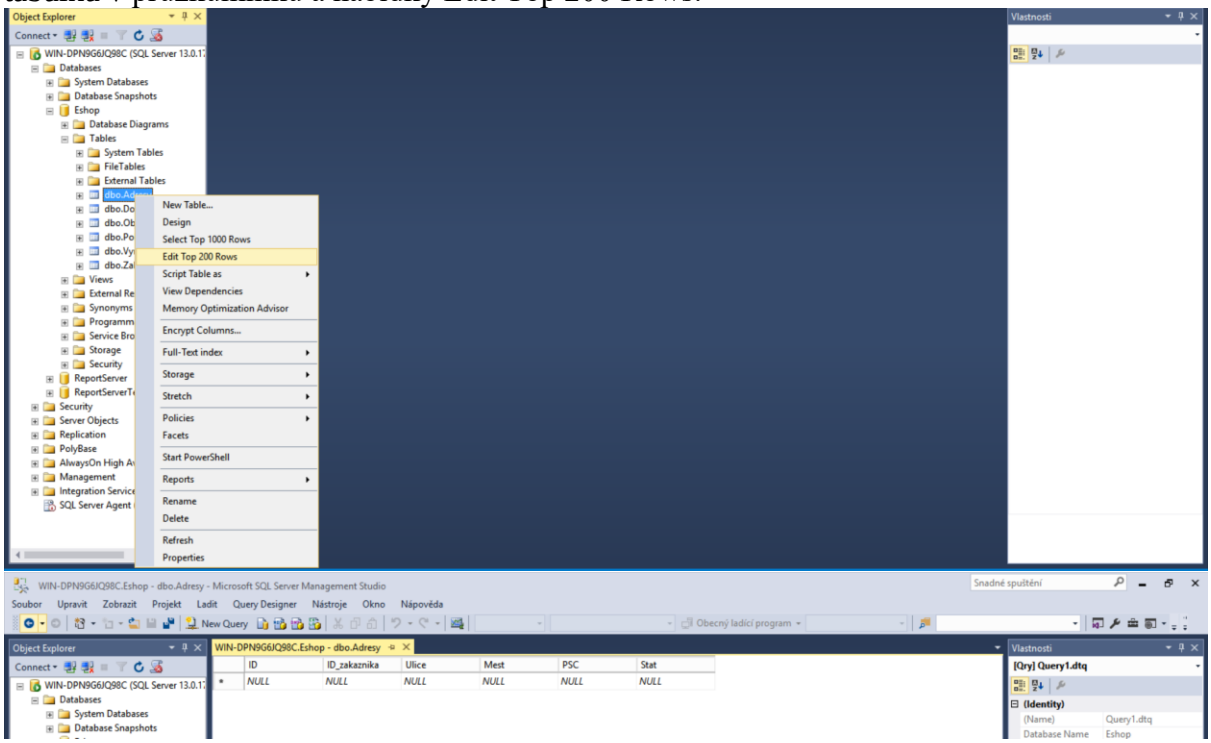
Až do teď byla práce hodně podobná tomu, jak jsme pracovali s Accessem, ale teď nastává jeden ze základních rozdílů, na který musíme myslet při tvorbě aplikace. Na rozdíl od Accessu se nám ID automaticky nevyplňuje, zde je potřeba to pro každou tabulku nastavit (což doporučuji!), nebo to ošetřit na úrovni aplikační vrstvy. Klikneme si na tabulku v průzkumníku pravým tlačítkem myši a zvolíme Design.



Pro sloupec s primárním klíčem ve vlastnostech najdeme položku Identity Specification, rozklikneme ji a u položky Is Identity změníme No na Yes. Změny uložíme.

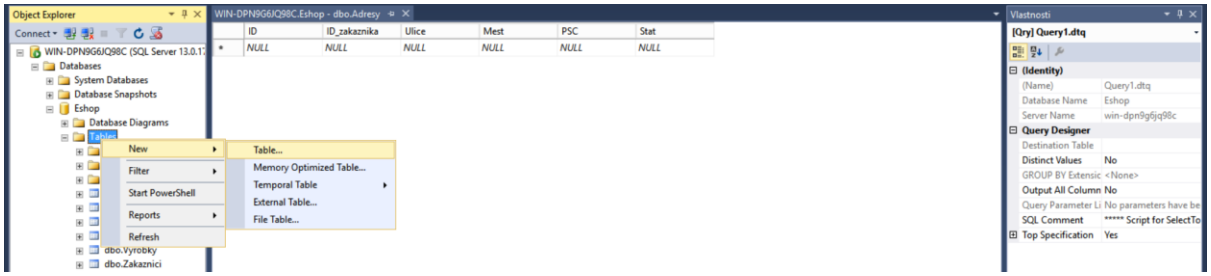


Pro naplnění dat v grafickém prostředí jde doplnit data pomocí nabídky pravého tlačítka na tabulku v průzkumníku a nabídky Edit Top 200 Rows.



Nyní začneme kliknutím do buněk zobrazené tabulky vyplňovat jednotlivé atributy cvičnými daty stejně, jako v Access.

Mnohem vhodnější je definovat vše pomocí příkazů, nebo vytvořit tabulky pomocí pravého tlačítka myši na složku a nabídky New. Takto vytvořené tabulky se nám neprováží, ale je méně problémové nastavení bussines omezení v tabulce. Dále takto vydefinované tabulky přidat do diagramu a vytvořit vazby, tak jak jsme to dělali.



Nyní je databáze po doplnění testovacích dat připravena k testování správnosti návrhu, generování sestav a pohledů. Pro testování správnosti návrhu budeme potřebovat jazyk SQL, o kterém je následující kapitola.

20.4. Jazyk SQL

SQL – Structured Query Language (Strukturovaný dotazovací jazyk) - je obecný nástroj pro manipulaci, správu a organizování dat uložených v databázi počítače. Je v prvé řadě určen uživatelům, i když jej v mnoha směrech využívají i tvůrci aplikací. Je adaptovatelný pro jakékoliv prostředí.

Název tohoto nástroje je sice stručný, není však výstižný. SQL totiž není pouze dotazovací jazyk, s jeho pomocí je možno také definovat data, tedy strukturu tabulky, naplňovat sloupce tabulky (pole záznamů) daty, definovat vztahy a organizaci mezi položkami dat. Dále umožňuje řízení přístupu k datům, tedy udělování a odebírání přístupových oprávnění na různých úrovních, čímž chrání data před náhodným nebo úmyslným zničením, neautorizovaným čtením nebo manipulaci s nimi. Dále také umožňuje sdílené využívání dat a zajišťuje hladký průběh činností, přistupuje-li k datům více uživatelů současně.

SQL je specializovaný programovací jazyk, který se používá ve vhodném prostředí buď uživatelsky nebo interaktivně k okamžitému řešení úloh (nejčastěji dotazy), nebo se jeho příkazy vkládají do hostitelského jazyka. SQL není však plnohodnotným samostatným programovacím jazykem, např. proto, že se v něm ve většině implementací nenachází řídicí programové konstrukce a další požadované prvky, které by měl obsahovat každý obecný programovací jazyk. SQL je tedy standardizovaný nástroj pro práci s relačními databázemi. Nepředstavuje databázový systém, ale různě integrovanou součást systému řízení bází dat.

SQL je především interaktivní dotazovací jazyk - umožňuje získat odpovědi i na velmi komplikované dotazy téměř ihned. Je nástrojem neprocedurálním, s množinovým přístupem k datům a je jazykem standardizovaným, je srozumitelný, protože chápe data v podobě tabulek, což je snadno pochopitelné i uživatelům. Pracuje s relačními databázemi, ve kterých se uživatel dívá na data v podobě soustavy provázaných tabulek. Každá tabulka představuje množinu dat, která je uspořádaná v řádcích (záznamech) a sloupcích (položkách). Na hodnotu dat se uživatel odkazuje jako na prvek v matici.

V převážné většině případů je výsledkem úlohy popsané v SQL nějaká množina dat z jedné nebo více tabulek, tzv. tabulka výsledků, která nemusí být vždy konečným produktem. Může sloužit jako množina vstupních údajů pro další zpracování. Např. pro výtisk etiket nebo vykreslení grafu atd.

SQL může sloužit jako „společná řeč“, zejména při provozu v sítích, na kterých se používají různé databázové produkty.

Jazyk SQL se používá také k vytváření náhledů (dotazů). Náhledy SQL umožňují vytvořit pro různé uživatele různé pohledy na struktury tabulek a na data. Každý uživatel tak vidí pouze ta data, která má vidět. Přitom data vidí uživatel opět v podobě jednoduché tabulky, i když ve skutečnosti data pocházejí z různých tabulek. Data zobrazovaná v náhledech jsou dynamická, tzn. změní-li se data v tabulkách (databázových souborech), změní se také data, které zobrazuje náhled. Také naopak. Pracuje-li se s aktualizačním náhledem (speciální typ náhledu,

který umožňuje nejen data prohlížet, ale i přidávat a aktualizovat) a změní se v něm data, promítnou se změny do patřičných tabulek (databázových souborů).

Klíčovým pojmem v jazyku SQL je příkaz. Každý příkaz začíná klíčovým slovem. Slovo vyjadřuje orientačně, jakou činnost daný příkaz provádí. Za klíčovým slovem následuje jedna nebo více volitelných klauzulí, které blíže specifikují povahu vykonávané činnosti, nebo určují data, s nimiž má příkaz pracovat.

Každá klauzule začíná klíčovým slovem, jako jsou např. FROM nebo WHERE. Některé klauzule jsou povinné, jiné volitelné. Každá implementace SQL používá kromě standardních klauzulí, daných konvencemi ANSI/ISO, své vlastní klauzule, někdy se i činnost standardních klauzulí mírně či více liší.

Jména objektů v jazyku SQL mají ve standardu délku 1-18 znaků, z toho první znak musí být písmeno, a jméno nesmí obsahovat mezery nebo interpunkční znaky. Při udávání jména je někdy potřeba jméno kvalifikovat, vyskytují-li se stejná jména, a není jednoznačně jasné, na který objekt se jméno odkazuje. Kvalifikace jména se provádí pomocí kvalifikátoru ´.´ (tečka). Často se kvalifikace používá se jmény tabulek: <vlastník>.<jméno_tabulky> nebo při kvalifikaci sloupců, což se v databázových systémech používá velmi často:
< jméno_tabulky>.< jméno_sloupce> , obecně se v SQL připouští i kvalif. víceúrovňová:
<vlastník>.<jméno_tabulky>.< jméno_sloupce>

Číselné konstanty se píší klasickým způsobem. Jediný problém může vzniknout v implicitním nastavení oddělovače desetinných míst, tzn. že v různých implementacích SQL se často může psát místo desetinné čárky tečka. Dále také některé "dialekty" SQL nemusí povolovat mezeru mezi znaménkem a číslem. Řetězcové konstanty se standardně píší v oddělovačích apostrofy, v některých implementacích je možno používat i uvozovky. Datové a časové konstanty se obvykle v SQL píší jako znakové řetězce. V mnoha implementacích se dá použít (vestavěnou) konverzní funkci CTOD(), která převádí položku typu řetězec na typ datum. Některé implementace umožňují navíc používat tzv. symbolické konstanty; ve standardu ANSI/ISO je taková jediná, a to USER, jež označuje uživatelské jméno, pod kterým se aktuálně přistupuje k databázi. Součástí standardu ANSI/ISO je zvláštní indikátor (požadavek na zacházení s chybějícími hodnotami), psaný obvykle symbolicky jako NULL. Takto uvedená hodnota v tabulce znamená, že tato hodnota chybí, není použitelná nebo není známá. Další obvyklé konstanty jsou např. CURRENT DATE, CURRENT TIME, CURRENT TIMESTAMP pro systémové datum, systémový čas a systémové datum a čas. Jiné implementace používají místo symbolických konstant vestavěných funkcí. Pro systémové datum se užívá funkce DATE(), pro čas TIME().

Výrazy se skládají z operandů (konstanty, proměnné, jména sloupců, vestavěné funkce) a operátorů, s jejichž pomocí se získá výsledná hodnota výrazu. ANSI/ISO standart SQL uvádí čtyři základní aritmetické operace: sčítání, odečítání, násobení a dělení, přičemž priorita je obvyklá – násobení a dělení mají vyšší prioritu než sčítání a odečítání. Prioritu provádění operací je samozřejmě možno změnit použitím kulatých závorek. Mnohé implementace podporují další operátory a umožňují také fungování základních operací pro znakové řetězce a data typu datum nebo čas. Např. operátor + (sčítání) aplikovaný na znakové řetězce znamená zřetězení, operátor - (odečítání) aplikovaný na hodnotu typu datum dává jako výsledek rozdíl mezi dvěma daty ve dnech. Standardně je také v SQL zajištěna automatická konverze z celých čísel na desetinná a z desetinných na čísla v pohyblivé řadové čárce.

Mnohé implementace SQL obsahují řadu vestavěných funkcí, i když tyto nejsou ve standardu ANSI/ISO vůbec zmiňovány. V příkazech jazyka SQL se mohou, např. v databázovém prostředí Microsoft Access, prakticky používat nejen vestavěné funkce danému prostředí vlastní, ale jsou dokonce povoleny i uživatelsky definované funkce. Obecně je možno se řídit zásadou, že vestavěná funkce vracející údaj určitého typu se může použít všude tam, kde je možno uvést i konstantu tohoto typu.

V řadě implementací se příkazy SQL, i když těchto příkazů není mnoho, rozdělují do dvou základních skupin. První skupinu tvoří příkazy pro definice dat – DDL (Data Definition Language), jež zabezpečují v podstatě tři základní činnosti vykonávané nad objekty databáze, tj. vytvořit objekt, vymazat existující objekt a změnit definici objektu. Druhou skupinu tvoří jazyk pro manipulaci s daty – DML (Data Manipulation Language), jež zahrnuje tyto tři základní příkazy: přidání (vymazání) jednoho nebo více řádků z (do) existující tabulky a aktualizace sloupcových hodnot určené množiny řádků. Kromě rozdělení do dvou hlavních výše uvedených příkazových skupin určujeme další typy skupin příkazů, a to příkazy pro kontrolu přístupu k datům (ochrana před neoprávněným přístupem) - DCL (Data Control Language), řízení transakcí, dále pak programové příkazy, agregované funkce (i konverzní) a predikáty.

20.4.1. Definování dat pomocí SQL

Na tomto místě se sluší připomenout, že budeme užívat základy jazyka SQL, ale budeme se konkrétně bavit o implementaci Microsoftu, takže o T-SQL, kterému budeme věnovat samostatnou kapitolu. Když to vezeme historicky, tak tato kapitola je věnována jazyku DLL. Vytvoření tabulky se provádí pomocí Create Table, což definuje strukturu tabulky. Syntaxe příkazu je

CREATE TABLE NazevNoveTabulky(
Definice sloupce, která se skládá ze tří částí,
Definice sloupce, která se skládá ze tří částí,
…
Volitelná omezení tabulky
….
);

Definice sloupce v příkazu CREATE TABLE obsahuje Název sloupce, datový typ sloupce a volitelná omezení hodnot ve sloupci, která mohou být Primary key, Not null, Null, Unique a Check. Pokud je na sloupci omezení NOT NULL, tak musí být vyplněn při vytvoření nového řádku tabulky. Ona hodnota NULL se chápe jako nevím, či neuvedeno a pokud je na sloupci omezení NULL (v případě MS SQL serveru není potřeba explicitně uvádět), tak je možný nechat daný atribut při tvorbě záznamu bez hodnoty. Příkazy se, dle ISO standartu ukončují středníkem stejně, jako v jiných programovacích jazycích, ale v praxi to není vývojovými nástroji ani servery požadováno. Příklad:

```
Create Table Zakaznici(
	ID int NOT NULL IDENTITY(1,1),
	Jmeno nvarchar(50) NOT NULL,
	Prijmeni nvarchar(50) NOT NULL,
	Email nvarchar(MAX) NOT NULL,
	Telefon nvarchar(50),
	Primary Key (ID), UNIQUE(ID)
		);
```

Definování primárních klíčů se provádí pomocí klíčového slova CONSTRAINT v omezeních tabulky, nebo jak je vidět v příkladu nastavením parametru a jména sloupce v závorce. Sloupec sloužící jako primární klíč musí mít omezení not null a nutné je, aby název omezení byl jedinečný, často se odvozuje od názvu sloupce. Pomocí omezení tabulky je možné definovat více sloupcové primární klíče a také definovat náhradní klíč.

Definování cizích klíčů se provádí pomocí omezení tabulky. Fráze **on update** udává, jakou akci má systém provést při změně hodnoty primárního klíče. Klíčové slovo CASCADE určuje, že stejná operace se provede v souvisejícím sloupci druhé tabulky. Fráze on delete uvádí, jakou akci má systém provést při smazání řádku v tabulce s primárním klíčem, standardně se vypouští, její výchozí nastavení je no action. Příklad definování cizích klíčů:

```
CREATE TABLE Adresy(

	ID INT NOT NULL IDENTITY(1,1) PRIMARY KEY,
	Zakaznik_ID INT NOT NULL FOREIGN KEY REFERENCES dbo.Zakaznici(ID),
	Ulice NVARCHAR(MAX) NOT NULL,
	Cislo_popisne NVARCHAR(50),
	Mesto NVARCHAR(250) NOT NULL,
	PSC NVARCHAR(10) NOT NULL,
	Stat NVARCHAR(MAX)
		);
```

Názvy tabulek začínají dbo., což značí jmenný prostor database owner. Příkazy jazyka SQL napsané v Management studiu, nebo Visual Studiu provedeme kliknutím na Execute (provede všechny příkazy v souboru). Pokud chceme pustit jen některé příkazy ze souboru, tak dané příkazy označíme a následně klikneme na Execute.

Pro vkládání dat do tabulek složí jazyk DML, který je součástí SQL. Data se vkládají pomocí příkazu insert into. Příkaz insert into má dvě varianty, jedna slouží pro zápis do všech sloupců v pořadí, jak jsou v tabulce uloženy a má syntaxi:

INSERT INTO *table_name*
VALUES (*value1, value2, value3, ...*);

Nebo pro zápis dat jen do specifických sloupců, případně do sloupců na přeskáčku:

INSERT INTO *table_name* (*column1, column2, column3, ...*)
VALUES (*value1, value2, value3, ...*);

Příkladem pak může být:

```
INSERT INTO dbo.Zakaznici
	(
	    Jmeno,
	    Prijmeni,
	    Email,
	    Telefon
	)
	VALUES
	(   N'Pavel', -- Jmeno - nvarchar(50)
	    N'Novotný', -- Prijmeni - nvarchar(50)
	    N'novotnypavel@gmail.com', -- Email - nvarchar(max)
	    N'733486789'  -- Telefon - nvarchar(50)
	)
```

Pokud databázový systém poskytuje hodnotu náhradního klíče, nemusíme nutně specifikovat primární klíč, ale je to vhodné. Pro vkládání číselných hodnot (int a Numeric…) nejsou hodnoty uzavřeny do apostrofů, ale hodnoty datových typů Char, VarChar a DateTime apod. se uzavírají do apostrofů! Pozor na to, že apostrofy musí být základní, které jsou v obyčejném txt, jinak dojde k chybě, protože Management studio ne vždy dobře interpretuje jiné kódování znaků pro editaci SQL skriptů. Pokud nejsou vyplněny všechny sloupce, tak systém do nevyplněných sloupců automaticky vloží hodnotu null. Nesmíme zapomenout, že, musíme vložit data do všech sloupců, které mají omezení not null, jinak nám celá dávka vkládání dat skončí chybou. Při importu dat do tabulek záleží na pořadí, což je způsobeno omezením referenční integrity cizího klíče.

Někdy je potřeba dodatečně upravit záznam v tabulce, pak použijeme příkaz update. Syntaxe tohoto příkazu je velmi jednoduchá:

UPDATE *table_name*
SET *column1 = value1, column2 = value2, ...*
WHERE *condition*;

A příkladem může být:

```
UPDATE dbo.Zakaznici
SET Jmeno = 'Karel'
WHERE dbo.Zakaznici.ID = 1
```

Jde o velmi silný příkaz, který umí způsobit velké škody!, pokud je nevhodně použit. Jím provedené změny jsou nevratné, proto nikdy nesmí chybět klauzule where!!, pokud by tato klauzule chyběla, tak by došlo k přepsání všech hodnot daného atributu na tu jednu hodnotu, která je uvedená v příkazu. Pomocí příkazu update lze také upravit více hodnot v jedné tabulce najednou, ale všechny musí být jednoznačně definované pomocí WHERE klauzule.

Příkaz update lze nahradit začně složitějším příkazem merge, který je k dispozici od verze SQL 2003 a jeho činnost je taková, že pokud je v dané buňce tabulky hodnota NULL, tak zapíše hodnotu zadanou, pokud je tam jakákoliv jiná hodnota, tak ponechá hodnotu původní. Jeho syntaxe vypadá zhruba následovně:

MERGE INTO *jmeno_tabulky* [USING *jmeno_tabulky* ON (*podmínka*)]
WHEN MATCHED THEN UPDATE SET *sloupec1* = hodnota1 [, *sloupec2* = hodnota2 ...]
WHEN NOT MATCHED THEN INSERT (*sloupec1* [, *sloupec2* ...]) VALUES (hodnota1 [, hodnota2 ...])

Více se k tomuto příkazu dozvíme v msdn dokumentaci společnosti Microsoft, pro naše školní účely v začátcích databází je to složitý, ale zajímavý příkaz.

K odstranění řádků slouží příkaz delete. Pozor, je stejně náchylný na chyby jako příkaz update. U příkazu delete je nutné se ještě zaměřit na nastavení on delete cascade!, abychom udrželi referenční integritu dat. Pozor na to, aby data nebyla ještě nikde vázána, pokud tomu tak je, operace končí chybou, nebo smazáním i vázaných dat v závislosti na on delete cascade
Příklad:

```
DELETE
FROM dbo.Zakaznici
WHERE ID = 1
```

Ještě máme možnost si tabulku zcela vyprázdnit a ponechat si v databázi jen její strukturu a reference. Tento příkaz je opravdu zvlášť nebezpečný, takže doporučuji jeho nepoužívání obdobně jako všech dalších mazacích příkazů, o kterých bude dále řeč. Vymazání celé tabulky se provádí pomocí příkazu TRUNCATE TABLE. Tento příkaz je je dostupný od verze SQL 2008. Neuplatňujeme klauzuli WHERE nebo jiné podmínky, data jsou vždy z tabulky odstraněna všechna! Nemůžeme jej provést u tabulky, na kterou je odkazováno pomocí omezení cizího klíče. Příklad může být: `TRUNCATE TABLE dbo.Adresy`

Odstranění sloupce a omezení se provádí se pomocí příkazu ALTER TABLE. Příkaz alter table umožňuje přidat/odebrat/upravit sloupce a omezení. Jako příklad si ukážeme odstranění sloupce:

```
ALTER TABLE dbo.Adresy DROP COLUMN Stat
```

Příkaz ALTER TABLE umožňuje i změnu datového typu sloupce tabulky, ale zde je vysoké riziko ztráty dat!

Odstranění celé tabulky se provádí se pomocí příkazu DROP TABLE. Patří mezi nejnebezpečnější příkazy jazyka SQL, protože data uložena v tabulce jsou nenávratně ztracena a je rozbito i schéma databáze. Příkaz drop nefunguje, pokud tabulka obsahuje nebo by mohla obsahovat hodnoty, které jsou potřebné pro omezení referenční integrity, takže je potřeba nejdříve rozvázat referenční integritu a až následně mazat tabulku. Příklad:

```
DROP TABLE dbo.Adresy
```

Opakem mazání je přidávání, asi nikoho z nás nepřekvapí, že i tuto činnost obstarává příkaz ALTER TABEL. Přidání omezení se provádí se pomocí příkazu alter table … add a klauzule CHECK. Omezení CHECK se podobají klauzulím WHERE v dotazech. Mohou obsahovat klíčová slova IN, NOT IN a LIKE. Lze kontrolovat rozsah pomocí symbolů „<“ a „>“. Jako příklad si můžeme uvést:

```
ALTER TABLE dbo.Adresy ADD CONSTRAINT zakaznik UNIQUE(Zakaznik_ID)
```

Obdobně vložíme i nový sloupec, akorát místo CONSTRAINT píšeme rovnou definici sloupce začínající názvem.

20.4.2. Přehled vybraných příkazů jazyka SQL

Význam	Příkaz
Vytvoření tabulky s názvem a danými slupci	CREATE TABLE [název](seznam názvů sloupců a jejich datové typy oddělené čárkou)
Vyžaduje vždy vyplnění	NOT NULL
Definuje slupec primárního klíče	PRIMARY KEY
Do sloupce se budou automaticky dosazovat číla 1,2,3,…	IDENTITY(1,1)
Textový řetězec o 50 znacích	VARCHAR(50)
Deklarace vazby bezi tabulkami (tzv. cizí klíč). Za ním následuje název tabulky, do které odkazujeme a v závorce název sloupce	REFERENCES
Pokud smažeme záznam, na který se odkazuje smaže se i tento záznam	ON DELETE CASCADE
Pokud smažeme záznam, na který se odkazuje nastaví hodnotu v tomto sloupci na NULL	ON DELETE SET NULL
Vložení dat do tabulky	INSERT INTO [nazev tabulky] ([nazev sloupce], [nazev sloupce2]) VALUES ('hodnota sloupce', 'hodnota sloupce2')
Výpis dat z tabulky	SEKECT *seznam sloupců*
Výběr zdrojové tabulky	FROM *název tabulky*
Podmínka pro vrácená data	WHERE *podmínka*
Logická spojka *a* v podmínkách	and
Seřadit výsledky dle	order by [nazev sloupce]
Je nevyplněn	IS NULL
Sestupné řazení	order by [nazev sloupce] desc
Seřazení dle více kritérií	order by [nazev sloupce], [nazev sloupce]
Spojení dvou tabulek s vynecháním cizích klíčů obsahujících null	Join (automaticky se převede na inner join)
Spojení dvou tabulek s povolením cizích klíčů obsahujících null	Left join
Změna záznamu	UPDATE [nazev tabulky] SET [sloupec] = '2.A' WHERE [sloupec2] = 1

Význam	Příkaz
Odstranění duplicit výstupu	distinct
Smazání záznamu	DELETE FROM [nazev tabulky] WHERE [sloupec] = 3
Zobrazní n prvních výsledků	top
Vrácení skupiny záznamů	Group by
Vrátí počet záznamů v tabulce	Count(*)
Vrací součet číslených typů	Sum()
Vrací průměrnou hodnotu číselných typů	Avg()
Vrací směrodatnou odchylku	Stdev()
Filtrování aplikované na skupiny	having
Dotazování na text s využitím zástupných symbolů	Like 'text'
0 až n libovolných symbolů po like	%
Textový řetězec	'text'
Jeden libovolný symbol po like	_
Rozsah symbolů po like	[symbol - symbol]
Vrací první nenulový parametr z výčtu	Coalesce(par1, par2, …,parN)
Pro každý řádek vyhodnocený poddotaz spojený s daným řádkem (použije se místo join)	Cross apply(*poddotaz*)
Pro každý řádek vyhodnocený poddotaz spojený s daným řádkem (použije se místo join) s umožněním Null	Outer apply(*poddotaz*)
Export výsledku dotazu do XML uložený do proměnné	SET @xml (*SQL dotaz* FOR XML AUTO, ROOT('název kořenového tagu'))
Začátek transakce	begin transaction
Konec transakce	commit transaction
Ukončení tranasakce při méně závažných chybách SQL	SET XACT_ABORT ON
Odchytávání výjimek	Begin try *dotazy* end try
Obsluha vyjímek	Begin catch *obslužný kód* end catch
Výpis textu do terminálu	Print 'text'

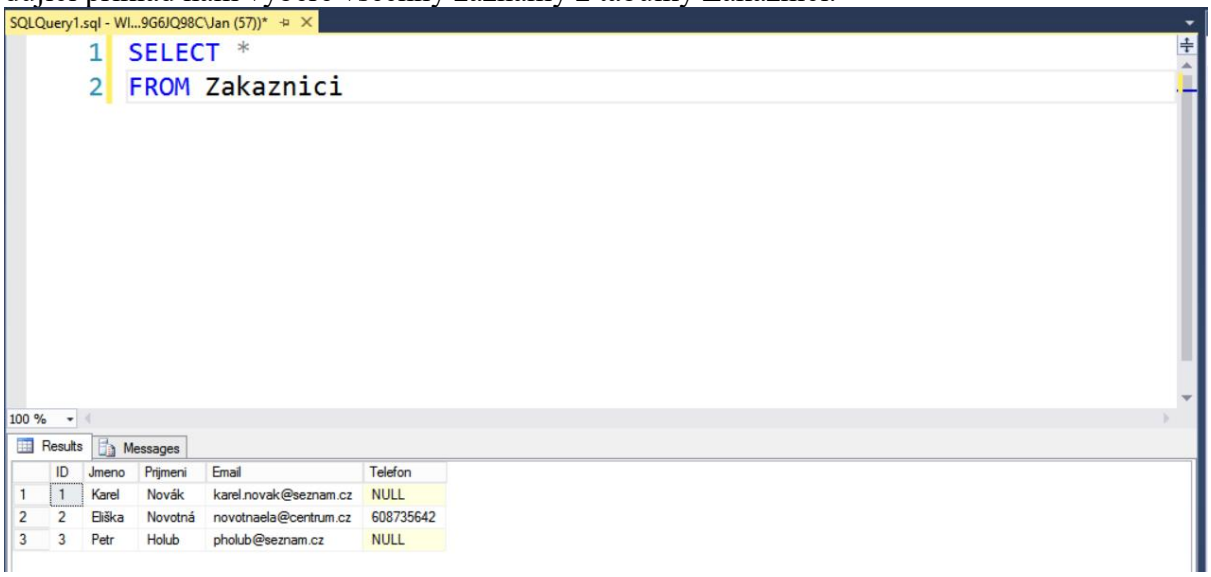
Význam	Příkaz
Vrácení změn v databázi provedených transakcí předcházející příkazu	Rollback transaction
Vytvoření indexu	CREATE INDEX index_name ON <object> (sloupec [ASC | DESC])
Zjištění údajů v cache	https://msdn.microsoft.com/en-us/library/ms187743.aspx
komentář	/* komentářový text */
Řádkový komentář	--
Logické spojky	And, or…
Výběr ze sady podmínek	IN('pod1', 'pod2'…)
Rozsah hodnot v klauzuli Where	Between *hodnota* and *hodnota*
Změna hodnot, nebo vytvoření záznamů, pokud neexistují	MERGE INTO jmeno_tabulky [USING jmeno_tabulky ON (podmínka)] WHEN MATCHED THEN UPDATE SET sloupec1 = hodnota1 [, sloupec2 = hodnota2 ...] WHEN NOT MATCHED THEN INSERT (sloupec1 [, sloupec2 ...]) VALUES (hodnota1 [, hodnota2 ...])
Odstranění tabulky	drop teble *název tabulky*
Odstranění všech dat z tabulky	truncate table *název tabulky*
Omezení referenční integrity	*Jméno sloupce* in *jmeno tabulky* must exist in *jmeno sloupce* in *jmeno tabulky*
Deklarace kurzoru	declare cursor *název* as select …

V další tabulce se podíváme na operátory jazyka SQL.

Operátor	Význam
=	Rovná se
>	Větší než
<	Menší než
>=	Větší rovno
<=	Menší rovno
<>	Nerovná se

20.4.3. Dotazy v jazyce SQL

Dotazování zařizuje jazyk DML, který je jednou z historických součástí jazyka SQL. Základní struktura dotazu je SELECT/FROM/WHERE, kdy klíčové slovo SELECT udává, že se jedná o dotaz, znamená vyber, klauzule FROM pak udává odkud a zde se vyskytuje zdrojová tabulka, nebo častěji spojení zdrojových tabulek. Ve více jak 95% se v dotazu vyskytuje klauzle WHERE, která udává podmínky. Základní struktura umožňuje dotazování na jednu tabulku, pokud je za klauzulí FROM uvedená jen tabulka bez použití příkazů JOIN. Bezprostředně za příkazem Select následují názvy sloupců, které mají být ve výsledku zastoupeny, nebo zástupný simbol * (hvězdička), který znamená všechny. Výsledkem příkazu SELECT je relace, tedy výstupní tabulka. Pokud je výsledkem pouze číslo, tak se to považuje za relaci o jednom řádku a jednom sloupci. Pořadí sloupců výsledné relace závisí na pořadí za příkazem SELECT a pokud chceme odstranit z výsledku duplicitní záznamy, tak příkaz SELECT rozšíříme o klíčové slovo Distinct. Pokud pracujeme s textovými řetězci zavřenými do uvozovek (jak je to potřeba), jsou rozlišována malá a velká písmena, takže je nutné na to myslet. Následující příklad nám vybere všechny záznamy z tabulky Zákazníci.



Nyní náš dotaz upravíme tak, aby vrátil jen Jméno a Příjmení zákazníků, u kterých není vyplněno telefonní číslo (ve sloupci Telefon se nachází hodnota NULL):

```
SELECT Jmeno, Prijmeni
FROM Zakaznici
WHERE Telefon IS NULL
```

Vybírané řádky záznamů se určují pomocí klauzule WHERE, zatím co o vybírání sloupců byla řeč víše. Pokud je sloupec datového typu Char či VarChar je nutné porovnávané hodnoty uzavřít do jednoduchých uvozovek (apostrofů). Podmínky lze spojovat pomocí logických operátorů AND a OR. Pokud chceme, aby sloupec nabýval některou z uvedených hodnot, tak použijeme v klauzuli WHERE klíčové slovo IN(*seznam hodnot*), opačný případ se vytvoří pomocí dvojice klíčových slov NOT IN(*seznam hodnota*) sloupec pak nesmí nabývat žádné z hodnot uvedených v seznamu u IN. Následující příkaz vypíše všechny, kteří nemají uvedené telefonní číslo a zároveň nemají příjmení Holub.

```
SELECT Jmeno, Prijmeni
FROM Zakaznici
WHERE Telefon IS NULL and Prijmeni NOT IN('Holub')
```

Nebo správnější varianta tohoto datazu:

```
SELECT Jmeno, Prijmeni
FROM Zakaznici
WHERE Telefon IS NULL and Prijmeni <> 'Holub'
```

Pokud chceme vyhledat něco v určitém rozsahu, tak to uděláme pomocí klíčového slova BETWEEN ke kterému se zadají krajní hodnoty intervalu. Následující příklad nám vypíše informace o výrobcích, jejichž ID je mezi 1 a 3.

```
SELECT *
FROM Vyrobky
WHERE ID BETWEEN 1 and 3
```

Alternativně by se dal dotaz zapsat:

```
SELECT *
FROM Vyrobky
WHERE ID >= 1 and ID <= 3
```

Pokud chceme vyhledat něco, u čeho nevíme celý název, tak použijeme klíčové slovo LIKE, kde můžeme užít zástupných symbolů pro část textu. Klíčové slovo LIKE lze užít v kombinaci s klíčovým slovem NOT, čímž získáme inverzní výsledek oproti LIKE. Pokud chceme najít řádky, které mají v daném sloupci hodnotu null, tak použijeme klíčového slova IS NULL (viz příklad s nevyplněnými telefonními čísly). Inverzní výsledky k podmínce s IS NULL získáme užitím klíčového slova IS NOT NULL. Náš další příklad nám vypíše jméno, příjmení a email všech zákazníků, jejichž jméno začíná na Nov. Znak „%“ dává, že za znaky „Nov“ následuje libovolný počet dalších znaků.

```
SELECT Jmeno, Prijmeni, Email
FROM Zakaznici
WHERE Prijmeni LIKE 'NOV%'
```

Někdy z jakéhokoliv důvodu potřebujeme seřadit řádky výsledné tabulky podle nějakého sloupce. Řazení se provádí pomocí klauzule ORDER BY. Standardně jsou hodnoty řazeny vzestupně, ale v případě potřeby pomocí klíčových slov ASC a DESC nastavit vzestupné či sestupné řazení. Pokud klauzuli ORDER BY neuvedeme, tak jsou data na výsledku nejčastěji řazena dle ClustedIndex, ale mohou být i dle jiného indexu, pokud byl využit pro provedení dotazu. Řazení lze založit na více sloupcích, potom se řazení provádí nejprve dle prvního a následně v rámci výsledku ještě dle druhého atd.

```
SELECT Jmeno, Prijmeni, Email
FROM Zakaznici
WHERE Prijmeni LIKE 'NOV%'
ORDER BY Jmeno
```

Pokud nám data z jedné tabulky nestačí (většinou ne, proto máme databázi, a ne tabulku v Excelu), tak máme dvě možnosti, jak se dotázat na data z více tabulek. První a méně obvyklou je využití poddotazu, protože poddotazy se využívají především pro definování složitějších struktur, a druhou je užití příkazu JOIN. Nejprve si ukážeme to méně častou variantu, abychom věděli jak se tvoří poddotaz. Poddotazy jsou často v klauzuli where za klíčovým slovem in a jsou obaleny kulatými závorkami. Poddotazy mohou mít libovolně vnořovány jeden do druhého, ale vždy musí jako celek vrátit tabulku výsledků. Jako jednoduchý příklad si vypišme jména a příjmení zákazníků, kteří svou objednávku učili v průběhu měsíce července a seřaďme si je abecedně dle příjmení.

```
SELECT Jmeno, Prijmeni
FROM Zakaznici
WHERE ID IN (
      SELECT Zakaznik_ID
      FROM Objednavky
      WHERE Datum BETWEEN '2017-07-01' and '2017-07-31'
      )
ORDER BY Prijmeni
```

Druhou variantou, jak se dotázat na data z více tabulek je použít příkaz JOIN. S příkazem JOIN je to malinko složitější, pro připomenutí se podívejme do tabulky vybraných příkazů. Existují dva základní typy spojení pomocí příkazu JOIN, prvním je vnitřní (INNER) a druhým je vnější (UOTHER), který dále ještě dělíme na LEFT, RIGHT a FULL. Pokud zapíšeme samotné JOIN, tak se automaticky provede INNER JOIN. Teď je otázkou, jaký je v tom rozdíl.

INNER JOIN spojí pouze ty řádky z obou tabulek, které mají k sobě protějšek a pokud někde je hodnota NULL místo vazebné hodnoty, je tento řádek z výsledku vypuštěn. Jednoduše řečeno zobrazuje jen data řádků, které si na základě podmínek spojení odpovídají. INNER JOIN lze někdy nahradit také pomocí prostého oddělení názvů tabulek čárkou. Pokud se provede OUTHER JOIN, tak záleží na jeho typu, na které straně smí chybět párovací hodnota, nebo zda na obou. Pokud chceme provedení LEFT nebo RIGHT OUTHER JOIN, tak se dá zápis zkrátit vypuštěním klíčového slova OUTHER, ale pozor, pokud chceme FULL OUTHER JOIN, tak zde se žádným způsobem zkracovat nedá. Principem fungování příkazu JOIN je, že ze dvou či více relací utvoří jednu dočasnou, na kterou pak aplikuje podmínky klauzule WHERE. Není-li jednoznačné, ze které tabulky názvy sloupců pocházejí, tak se musí před název sloupce uvést název tabulky, který je s názvem sloupce spojen pomocí tečkové notace. Ukažme si nyní ten samý příklad, který jsme dělali poddotazem pomocí příkazu JOIN.

```
SELECT Jmeno, Prijmeni
FROM Zakaznici, Objednavky
WHERE Zakaznici.ID = Objednavky.Zakaznik_ID and Objednavky.Datum BETWEEN '2017-07-01'
and '2017-07-31'
ORDER BY Prijmeni
```

Nebo pomocí JOIN ON, což umožňuje spojit i více tabulek dohromady a výrazně zjednodušuje klauzuli WHERE, kde je možné přehledněji umístit poddotaz. Opět si ukážeme ten samý dotaz.

```
SELECT Jmeno, Prijmeni
FROM Zakaznici JOIN Objednavky
             ON Zakaznici.ID = Objednavky.Zakaznik_ID
WHERE Objednavky.Datum BETWEEN '2017-07-01' and '2017-07-31'
ORDER BY Prijmeni
```

Pokud budeme chtít vědět, co si objednali, tak stačí dotaz mírně upravit. V následujícím dotazu se objevuje klíčové slovo AS, které slouží k přejmenování sloupce ve výstupní tabulce.

```
SELECT Jmeno, Prijmeni, Nazev as Vyrobek
FROM Zakaznici JOIN Objednavky
             ON Zakaznici.ID = Objednavky.Zakaznik_ID
             JOIN Polozky_objednavky
             ON Objednavky.ID = Polozky_objednavky.Objednavka_D
             JOIN Vyrobky
             ON Polozky_objednavky.Vyrobek_ID = Vyrobky.ID
WHERE Objednavky.Datum BETWEEN '2017-07-01' and '2017-07-31'
ORDER BY Prijmeni
```

OUTHER JOIN má syntaxi plně závislou na použitém databázovém serveru, protože není definován původní specifikací SQL-92. Jeho asi nejčastější podoba je LET JOIN, ale jak již bylo řečeno v úvodu do práce s příkazem, má celkem 3 varianty a záleží na tom, co chceme dotazem získat a jak k sobě připojíme tabulky. Jako příklad si můžeme vypsat čísla objednávek jednotlivých zákazníků seřazená podle příjmení zákazníka.

```
SELECT Zakaznici.prijmeni, Objednavky.ID
FROM zakaznici LEFT JOIN objednavky
               ON Zakaznici.ID = Objednavky.Zakaznik_ID
ORDER BY Zakaznici.prijmeni
```

20.4.4. Agregační funkce MS SQL server

Jsou funkce, které nám dokáží provést nad daty základní výpočty. Výpočet se provádí nad celou množinou dat (nad všemi řádky) a často se s nimi používá GOUP BY, který dle nějakého atributu seskupuje data.

Funkce	Popis
Count()	Udává počet řádků ve výsledku
SUM()	Vrací součet za sloupec
AVG()	Vrací aritmetický průměr sloupce

Funkce	Popis
MAX()	Vrací maximum v daném sloupci
MIN()	Vrací minimum v daném sloupci
FIRST()	Vrací první hodnotu v grupě

Právě s agregačními funkcemi se hodně pojí výše zmiňované klíčové slovo AS, které přejmenovává sloupec výstupu. Pokud chceme počet jedinečných výskytů, tak použijeme funkce Count(Distinct *název sloupce*). Jako příklad si ukažme jednoduchý dotaz, který vrátí průměrnou cenu výrobků.

```
SELECT AVG(Cena_za_kus) AS 'Průměrná cena výrobků'
FROM Vyrobky
```

Protože jsme začali používat bez jakéhokoliv bližšího vysvětlení seskupování dat, tak by bylo vhodné si k tomu něco málo říct. Zmiňujeme ro právě zde, protože tato problematika úzce souvisí s agregačními funkcemi. Řádky se seskupují dle společných hodnot pomocí klauzule GROUP BY a pokud se vyskytují klauzule GROUP BY a WHERE, tak může dojít k nejasnostem při vyhodnocování dotazu, nicméně přednostně se uplatní klauzule WHERE. Integrované funkce daného SQL serveru lze aplikovat na skupiny řádků. Pokud chceme skupiny filtrovat na základě nějaké podmínky, tak použijeme klauzuli HAVING za kterou následuje podmínka.

20.4.5. Transakce

Transakce je jediný správný způsob modifikace, zápisu a často i dotazu v provozních databázích. Transakce obsahuje jeden až N příkazů, které se mají vykonat společně. Příkazy tvořící transakci ohraničují příkazy BEGIN TRANSACTION a COMMIT. V případě závažné chyby se příkazy neprovedou a již provedené změny se vrátí pomocí vyvolání příkazu ROLLBACK na pozadí. Pokud potřebujeme transakci ukončit při méně závažné chybě, tak použijeme možnost SET XACT_ABORT ON, jejíž aktivace platí v rámci jedné session (jednoho připojení) vůči serveru. MS Management studio vytváří novou session pro každý SQL soubor, nebo každou volbu New Query. Rizikové operace je dobré uzavřít do bloku begin try, end try pro odchycení výjimky, kdy kód pro obsluhu chyb zachycených v try bloku se využije blok kódu mezi begin catch a end catch.

Transakce je nedělitelná, buď se provede celá, nebo vůbec, a jako atomická jednotka práce s databází převádí DB z jednoho konzistentního stavu do jiného konzistentního stavu. Dokud není transakce potvrzena, dílčí změny nejsou viditelné pro ostatní transakce, naproti tomu efekty úspěšné transakce jsou již trvalé. Řízení transakcí je převážně automatické, explicitně lze transakce řídit pomocí příkazů COMMIT a ROLLBACK. Pro zajištění chodu transakcí je nutné žurnálování.

Transakce je vlastně posloupnost akcí (čtení, zápis, výpočet...), s nimiž se zachází jako s jedním celkem. Jako příklad uveďme typický úkol – převod peněz z účtu na účet. V tomto úkolu se provádějí dvě dílčí operace: - odečíst peníze ze zdrojového účtu – přičíst peníze na cílový účet. Aby databáze zůstala v konzistentním stavu, musí proběhnout buď obě dílčí operace, nebo žádná.

Je třeba říci, že transakce je možné explicitně definovat, nicméně pokud je nějaká operace nad databází zavolána bez „zabalení“ do transakce, databázový stroj stejně zavede transakci, obsahující tuto jedinou operaci. Tedy transakce jsou používány automaticky při jakékoli manipulaci s daty v databázi, nejen na vyžádání. Z toho též plyne, že každý příkaz je zapisován do žurnálu. Transakce se může nacházet v následujících stavech:

- Aktivní – počátek provádění
- Částečně potvrzený – těsně před COMMIT
- Chybný – výskyt chyby
- Zrušený – zavolání příkazu ROLLBACK

- Potvrzený – provedení příkazu COMMIT

Jako příklad můžeme uvést:

```
BEGIN TRANSACTION
	INSERT INTO dbo.Zakaznici
	(
	    Jmeno,
	    Prijmeni,
	    Email,
	    Telefon
	)
	VALUES
	(   N'Karolína', -- Jmeno - nvarchar(50)
	    N'Malá', -- Prijmeni - nvarchar(50)
	    N'karolmala@outlook.cz', -- Email - nvarchar(max)
	    N'745789654'  -- Telefon - nvarchar(50)
	)
COMMIT
```

Konzistentní transakce se označují jako ACID, kde A znamená tomická, C je konzistentní, I znamená izolovaná a D je trvalá, je taková, u které jsou všechny potvrzené změny trvalé (databázový systém takové změny neodstraní ani v případě selhání). Konzistentní značí, že žádné změny dat v průběhu zpracování nebudou povoleny, aktualizace výsledku se vztahuje ke chvíli zpuštění dotazu. Konzistence na úrovni příkazů znamená, že každý příkaz nezávisle zpracovává konzistentní řádky, ale během intervalu mezi příkazy je možné tyto řádky měnit, ale konzistence na úrovni transakce znamená, že všechny řádky ovlivněné libovolným z příkazů SQL jsou chráněny před změnami po celou dobu vykonávání transakce.

20.4.6. Techniky T-SQL

Transact-SQL (T-SQL) je proprietární rozšíření pro SQL od společností Microsoft a Sybase. Microsoft tento jazyk používá ve svých produktech Microsoft SQL Server. T-SQL poskytuje širokou škálu funkcionality pro programátory nebo databázové administrátory. Transact SQL umožňuje řízení SQL dotazů, podmíněné zpracování konstrukcemi IF, ELSE, CASE, nebo WHILE, využití zabudovaných funkcí MSSQL serveru, nebo také sekvenční zpracování pomocí kurzorů. Velká síla T-SQL je v uložených procedurách díky nimž mohou programátoři definovat dávky SQL a znovu využívat kód, definovat triggery, nebo například řešit část aplikační logiky už na databázovém serveru. Administrátoři mohou pomocí systémových uložených procedur administrovat server, systémové procedury totiž umožňují pomocí T-SQL nastavovat velkou škálu parametrů SQL serveru, vytvářet databáze, zálohovat, nastavovat práva atd.

20.4.6.1. Podmíněné zpracování

T-SQL umožňuje vracet hodnoty dotazu na základě dalšího zpracování a logického vyhodnocení výrazů. Můžeme vykonat blok příkazů, pokud je splněna podmínka, nebo vykonat úplně jiný kód, pokud podmínka splněna není. Na základě získaný hodnot můžeme tyto hodnoty dále zpracovávat nebo úplně měnit či transformovat. Základem je programátorům dobře známá konstrukce IF, ELSE. Struktura IF, ELSE umožňuje vyhodnotit logický výraz a dle jeho výsledku vykonat dotaz. Syntax:

```
IF Boolean_expression { sql_statement | statement_block }
        [ ELSE { sql_statement | statement_block } ]
```

Příklad:
```
DECLARE @Condition int
SET @Condition = 250
IF @Condition > 100
  BEGIN
    SELECT TOP 100 * FROM TABLE
  END
ELSE
  BEGIN
    SELECT TOP @Condition * FROM TABLE
  END
```
Další možností je CASE, což je funkce umožňující vracet hodnotu na základě vyhodnocení jednoho, nebo více výrazů, obdobně jako v programování. Syntax:
```
CASE input_expression
     WHEN when_expression THEN result_expression [ ...n ]
     [ ELSE else_result_expression ]
END
```
Příklad:
```
SELECT firstname, surname,
CASE gender
  WHEN 'male' THEN 'M'
  WHEN 'female' THEN 'F'
ELSE 'UNKNOWN'
age, birthdate FROM peoples
```
20.4.6.2. Řízení toku
T-SQL umožňuje zpracovávat bloky příkazů opakovaně, pohybovat se v nich, nebo je s nějakou podmínkou okamžitě opouštět. Pomocí cyklu WHILE můžeme vykonávat blok příkazů, dokud není splněná námi definovaná podmínka, pomocí příkazu GOTO se můžeme dle uvážení pohybovat v bloku příkazů nebo bloky opouštět pomocí příkazu RETURN.

RETURN umožňuje okamžitě opustit aktuální dotaz, dávku, nebo proceduru a vrátit se do nadřazeného bloku s návratovou hodnotou. Syntax:
```
RETURN [ integer_expression ]
```
Příklad:
```
BEGIN
  SELECT firstname,surname FROM anytable
  IF (@@ROWCOUNT = 0) RETURN
  UPDATE ...
  ...
END
```
WHILE umožňuje v cyklu vykonávat příkaz nebo dávku příkazů. Syntax:
```
WHILE Boolean_expression
     { sql_statement | statement_block | BREAK | CONTINUE }
```
Příkald:
```
DECLARE @nomrows int
WHILE(SELECT @numrows=count(*) FROM anytable WHERE columnA>10)
BEGIN
  IF @numrows=1 BREAK
  UPDATE anytable SET columnA=(columnA+1)
END
```
Příkaz GOTO dokáže provádět skoky v dávkách SQL. Jeho syntaxe je:
```
GOTO label
```
Příklad:
```
...
UPDATE anytable SET columnA=1
IF @@ROWCOUNT = 0 GOTO endblock
UPDATE anytable SET columnB=2
```

```
...
...
endblock:
RETURN
```

20.4.6.3. Kurzory

Kurzory umožňují sekvenčně zpracovávat řádky dotazu. Někdy je zapotřebí zpracovávat řádky jeden po druhém, a ne celou sadu, právě k tomu kurzory slouží. Nicméně pokud to není nezbytně zapotřebí je lépe použít jiné konstrukce. Kurzory mohou totiž způsobit při svém běhu některé problémy, jedním z nich je velká paměťová náročnost, kurzory mohou způsobit i vyčerpání veškeré dostupné paměti, další problém nám může způsobit zamykání prostředků, ale asi největším jejich nedostatkem je rychlost, MSSQL server je založen na práci se sadami řádků, a ne se sekvenčním zpracováním, proto je práce se sadami vždy rychlejší. Někdy však maše aplikace použití kurzorů vyžaduje a vy jejich funkcionalitu velmi oceníme.

Kurzor se definuje SQL dotazem, který vrací sadu řádků. Řádky se pak z něj dají brát postupně po jednom, nebo po blocích, a to v obou směrech podle definice kurzoru. Když skončíme práci s kurzorem, je nutné ho explicitně uzavřít a dealokovat, abychom uvolnili veškeré prostředky, které kurzor alokoval. Pokud tak neučiníme, kurzor zůstane v paměti a bude jí blokovat. Problém může také nastat, pokud bychom alokovali dva stejně pojmenované kurzory (i když nejsou ve stejném bloku!), je lépe tomu předcházet a v každé proceduře pojmenovávat kurzor dle jeho funkce jiným názvem. Životní cyklus kurzoru:

- deklarace kurzoru
- otevření kurzoru
- práce s řádky
- uzavření kurzoru
- dealokace kurzoru

Už při deklaraci kurzoru definujeme, jak s ním budeme pracovat a také se kterými daty, při jeho deklaraci určujeme SQL dotazem, se kterými daty budeme pracovat a dodatečným deklaracemi můžeme určit jestli budeme kurzor otevírat pouze pro čtení, jestli se budeme pohybovat pouze dopředu nebo i zpět, nebo třeba viditelnost kurzoru. Obecná syntaxe T-SQL:

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]
     [ FORWARD_ONLY | SCROLL ]
     [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
     [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
     [ TYPE_WARNING ]
     FOR select_statement
     [ FOR UPDATE [ OF column_name [ ,...n ] ] ]
[;]
```

Příklad:

```
DECLARE cursorABC CURSOR
FOR SELECT columnA,columnB,columnC FROM anytable
```

Příkazem OPEN kurzor naplníme, obecně se příkaz zapisuje:

```
OPEN { { [ GLOBAL ] cursor_name } | cursor_variable_name }
```

Příklad:

```
OPEN cursorABC
```

Pracovat s řádky můžeme pomocí příkazu FETCH kterým načítáme do proměnných hodnoty z jednotlivých řádků, výsledek volání fetch je uložen v proměnné @@FETCH_STATUS (0 úspěch, -1 neúspěch, -2 chybějící řádek). Syntaxe má podobu:

```
FETCH
          [ [ NEXT | PRIOR | FIRST | LAST
                   | ABSOLUTE { n | @nvar }
                   | RELATIVE { n | @nvar }
              ]
              FROM
          ]
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ]
```

Jako příklad si můžeme uvést:

```
FETCH NEXT FROM cursorABC INTO @A,@B,@C
```

Příkazem CLOSE uvolníme paměť, kterou alokoval kurzor pro načtení řádků a příkazem DEALOCATE uvolníme veškeré prostředky, které kurzor alokoval. Synataxe:

```
CLOSE { { [ GLOBAL ] cursor_name } | cursor_variable_name }

DEALLOCATE { { [ GLOBAL ] cursor_name } | @cursor_variable_name }
```

Příklad:

```
CLOSE cursorABC
DEALLOCATE cursorABC
```

Po hrubém popisu jednotlivých položek si můžeme ukázat jeden kurzor, který projde všemi fázemi životního cyklu.

```
-- deklarace proměnných
DECLARE @A int
DECLARE @B varchar(10)
DECLARE @C decimal(10,2)
-- deklarace kurzoru
DECLARE cursorABC CURSOR
FORWRD_ONLY
FOR SELECT columnA,columnB,columnC FROM anytable
-- otevření kurzoru
OPEN cursorABC
-- načte řádek do kurzoru
FETCH NEXT FROM cursorABC INTO @A,@B,@C
-- cyklus pro zpracování všech řádek
WHILE @@FETCH_STATUS = 0
BEGIN
-- zde můžeme zpracovávat řádky
  print @A
  print @B
  print @C
-- načte další řádek
FETCH NEXT
END
-- uzavření kurzoru
CLOSE cursorABC
-- dealokace kurzoru
DEALLOCATE cursorABC
```

Více o kurzorech v samostatné kapitole Kurzory, kde je mimo jiné uveden příklad deklarace kurzoru nad naší modelovou databází.

20.4.6.4. Uložené procedury

Uložená procedura je seskupení příkazů T-SQL do jediného objektu uloženého na SQL serveru. Když se procedura vykonává poprvé po spuštění (za nastartování) instance SQL serveru, uloží se do cache plán optimálního přístupu k datům (Execution plán, budeme se jim věnovat dále), server pak tento plán využívá při každém dalším spuštění, čímž může urychlit vykonání dotazu. To je jedna, a ne jediná, z výhod uložených procedur. Další výhodou je znovu

použitelnost kódu, namísto abychom všude, kde je potřeba definovali jednoúčelový blok příkazů, je lepší využít uloženou proceduru. Pokud by se tento kód měl vykonávat v nějaké z našich aplikací, jejich výhody se znásobí. Pokud by se třeba změnilo uložení dat v tabulkách, musíme naší aplikaci, popřípadě více aplikací na všech místech přepsat a v horším případě i překompilovat, ale pokud používáme uloženou proceduru, stačí ji změnit a pro aplikace se v podstatě nic nestalo. Použití procedur je také z pohledu DB administrátora bezpečnější než přímý přístup do tabulek.

Pro vytvoření uložené procedury použijeme následující syntaxi.

```
CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
    [ { @parameter [ type_schema_name. ] data_type }
        [ VARYING ] [ = DEFAULT ] [ OUT | OUTPUT ] [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { <sql_statement> [;][ ...n ] | <method_specifier> }
[;]
<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]

<sql_statement> ::=
{ [ BEGIN ] statements [ END ] }

<method_specifier> ::=
EXTERNAL NAME assembly_name.class_name.method_name
```

Pro lepší představivost si pojďme ukázat jeden příklad.

```
CREATE procedure [dbo].[MyProcedure]
@pid int,@other int,@sometext varchar(100)
AS
IF ISNULL(@pid,0)=0
  RETURN
EXEC ExecuteAnotherProcedure @pid
IF ISNULL(@other ,0)<>0
 BEGIN
    DECLARE @a int
    SELECT @a=a FROM anytable WHERE a=@pid AND b=@other
    UPDATE sometable SET a=@pid WHERE b=@other
 END
```

Někdy se stane, že potřebujeme uloženou proceduru upravit, nebo úplně smazat. Pokud k této situaci dojde, tak použijeme příkazy popsané následující syntaxí.

```
-- úprava procedury
ALTER { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]
...
same AS CREATE
-- odstranění procedury
DROP { PROC | PROCEDURE } { [ schema_name. ] procedure } [ ,...n ]
```

Někdy je potřeba proceduru zabezpečit, aby nikdo nemohl neoprávněně získat její definici, zejména pokud řešíme část aplikační logiky uloženými procedurami. Řešením tohoto problému je použití šifrování, ale to s sebou nese jedno velmi podstatné omezení, zabraňuje si prohlédnout její obsah. Pokud jsme si správností procedury jistí a máme bezpečně uloženou zálohu její definice, tak můžeme přistoupit k jejímu zašifrování následujícím postupem.

```
CREATE PROCEDURE [dbo].[MyProcedure]
WITH ENCRYPTION
AS
...
```

20.4.6.5. Uživatelsky defniované funkce

Uživatelsky definované funkce mají stejně jako uložené procedury mnoho výhod – ukládání do cache, znovupoužití kódu atd… Na rozdíl od procedur však můžeme funkce použít i tam kde procedury ne (například v klauzuli SELECT). Typy uživatelsky definovaných funkcí:

- Skalární uživatelsky definované funkce
- Přímé tabulkové uživatelsky definované funkce
- Vícepříkazové uživatelsky definované funkce

Skalární uživatelsky definované funkce mohou přebírat nula nebo více argumentů a vrací jedinou hodnotu. Obecná syntaxe:

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
    [ = DEFAULT ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

Příklad:

```
CREATE FUNCTION dbo.udf_DayOfWeek(@dtDate DATETIME)
RETURNS VARCHAR(10)
            AS
    BEGIN
    DECLARE @rtDayofWeek VARCHAR(10)
    SELECT @rtDayofWeek = CASE DATEPART(weekday,@dtDate)
                    WHEN 1 THEN 'Monday'
                    WHEN 2 THEN 'Tuesday'
                    WHEN 3 THEN 'Wednesday'
                    WHEN 4 THEN 'Thursday'
                    WHEN 5 THEN 'Friday'
                    WHEN 6 THEN 'Saturday'
                    WHEN 7 THEN 'Sunday'
        END
    RETURN (@rtDayofWeek)
    END
GO

-- použití funkce
SELECT udf_DayOfWeek(getdate())
```

Přímé tabulkové uživatelsky definované funkce mohou přebírat nula nebo více argumentů a vrací data na základě jediného příkazu SELECT. Odkazujeme se na ně z klauzule FROM. Syntaxe:

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = DEFAULT ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS TABLE
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    RETURN [ ( ] select_stmt [ ) ]
[ ; ]
```

Jako příklad můžeme uvést:

```
CREATE FUNCTION dbo.GetByCity(@city varchar(100))
RETURNS TABLE
AS
RETURN
(
SELECT * FROM peoples WHERE city=@city
(
GO
-- použití funkce
SELECT * FROM GetByCity('Prague')
```

Vícepříkazové uživatelsky definované funkce na rozdíl od přímých tabulkových funkcí, které jsou omezeny na jediný příkaz select, mohou vícepříkazové funkce vrátit výsledek definovaný pomocí více příkazů T-SQL. Zápis má syntaxi:

```
CREATE FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = DEFAULT ] [READONLY] }
    [ ,...n ]
  ]
)
RETURNS @return_variable TABLE <table_type_definition>
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN
    END
[ ; ]
```

20.4.6.6. Odchytávání výjimek

Pokud chceme v našich SQL příkazech nebo procedurách odchytávat výjimky, dává nám k tomu T-SQL nástroj v podobě konstrukce TRY…CATCH. Tato konstrukce je schopná zachytit výjimky, které mohou nastat při vykonávání našeho kódu a vykonat nějakou reakci na vzniklou chybu. Například vygenerovat zprávu o chybě nebo anulovat transakci. Nastane-li v bloku TRY nějaká výjimka, skončí SQL server v místě, kde nastala a vykoná příkazy v bloku catch. Výhoda této konstrukce je, že se do sebe může libovolně vnořovat. Konstrukce TRY nezachytí syntaktické chyby, nebo chybné názvy objektů. Funkce poskytující informace o chybách:

- ERROR_LINE – číslo řádky kde došlo k chybě
- ERROR_MESSAGE – chybová zpráva
- ERROR_NUMBER – číslo chyby
- ERROR_PROCEDURE – název procedury kde došlo k chybě
- ERROR_SEVERITY – závažnost chyby
- ERROR_STATE – stav chyby

Konstrukce TRY...CATCH má syntaxi:

```
BEGIN TRY
     { sql_statement | statement_block }
END TRY
BEGIN CATCH
          [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

Příklad:

```
BEGIN TRY
     INSERT anyTable (columnA,columnB,columnC) VALUES (1,2,3)
END TRY
```

```
BEGIN CATCH
          print 'Chyba při vykonávání příkazu!'
END CATCH
```

Někdy potřebujeme na chyby reagovat vlastními výjimkami, i na to má T-SQL nástroj, vlastní výjimku můžeme přidat pomocí systémové procedury sp_addmessage a výjimku vyvolávat příkazem RAISERROR.

```
sp_addmessage [ @msgnum = ] msg_id , [ @severity = ] severity , [ @msgtext
= ] 'msg'
     [ , [ @lang = ] 'language' ]
     [ , [ @with_log = ] { 'TRUE' | 'FALSE' } ]
     [ , [ @REPLACE = ] 'replace' ]
-- úprava vlastní vyjímky
sp_altermessage [ @message_id = ] message_number   , [ @parameter = ]
'write_to_log'
      , [ @parameter_value = ] 'value'
-- odstraněni vyjímky
sp_dropmessage [ @msgnum = ] message_number
        [ , [ @lang = ] 'language' ]
```

Příklad:

```
EXEC sp_addmessage 123456, 20, N'Došlo k mojí vyjímce!'
```

Vyvolat vlastní výjimku můžeme pomocí RAISERROR:

```
RAISERROR ( { msg_id | msg_str | @local_variable }
    { ,severity ,state }
    [ ,argument [ ,...n ] ] )
    [ WITH OPTION [ ,...n ] ]
```

Příklad:

```
RAISERROR (123456, 20, 1, N'any.anytable' )
```

20.5. Administrace Databází

Administrace dat nejčastěji označuje funkci, která je manažersky orientovaná a týká se celé organizace, zabývá se ochranou, zabezpečením, podnikových dat. Administrace databází se vztahuje na techničtější funkci, která je specifická pro konkrétní databázi a týká se i aplikací na zpracování dané databáze. U malých osobních databází uživatel plní i roli admina, stará se o zálohování a vede minimální dokumentaci, ale u rozsáhlých databází je databázový administrátor vedoucím oddělení zprávy databáze. Administrátor musí v první řadě zajistit vývoj a použitelnost databáze.

Je důležité si uvědomit, že se systémem často pracuje velké množství lidí 24 hodin denně 7 dní v týdnu. Současně s manipulačními operacemi je nutné vymáhat omezení referenční integrity, které by při volném pohybu bylo narušeno. Je nutné kontrolovat souběžné akce uživatelů, aby bylo zajištěno, že výsledky splní očekávání, tedy že uživatelé dostanou správnou podobu dat, o které žádají. Je nutné nastavit a vynucovat bezpečnostní omezení, samozřejmostí je fungování postupů pro zálohy a obnovení.

Kontrola souběžnosti slouží k tomu, aby práce jednoho uživatele neovlivňovala nevhodně práci druhého. Jde o to, aby uživatel vždy obdržel stejné výsledky, jako když by pracoval na systému sám, nebo aby jeho práce byla ovlivněna předvídatelným způsobem. Za velmi vysokou cenu lze dosáhnout přísnou kontrolu souběžnosti uzamčením celé databáze, ta cena je v tom, že do dokončení činnosti nikdo další neprovede žádnou operaci nad DB, ani dotaz nad daty. Je potřeba volit střední cestu, při nastavování přístupu pro více uživatelů, příliš slabé omezení způsobí chybné výsledky dotazů, či narušení referenční integrity.

Ve většině aplikací uživatelé odesílají svou práci ve formě transakcí, které se také označují jako logické pracovní jednotky. Transakce je taková posloupnost databázových akcí, kterou je nutné úspěšně provést jako celek, nebo neprovést žádnou z těchto akcí => atomická transakce. V případě selhání některého kroku v transakci se vše vrací zpět a nedojde ke změně

databáze. Souběžně zadané transakce se zpracovávají prokládaně po menších blocích, zpracování využívá tzv. preemtivní multitasking.

Problém ztracené aktualizace nastane tehdy, pokud dva uživatelé získají data, která v danou chvíli jsou správná a jeden z uživatelů aktualizuje položku, ale druhý má ve své paměti původní její původní stav. Pokud provedou oba uživatelé aktualizaci dané položky, tak databáze obsahuje chybný údaj. O ochraně proti tomuto problému se budeme bavit dále.

Nečisté čtení nastává, když jedna transakce přečte změněný záznam, který nebyl v databázi potvrzen. Nastává například, když jedna transakce změní hodnotu, druhá ji načte a první následně zruší změnu hodnoty.

Neopakovatelné čtení se vyskytuje tehdy, když transakce znovu načte dříve přečtená data a zjistí úpravy nebo odstranění způsobené jinými transakcemi.

Fantomní čtení popisuje situaci, kdy transakce znovu načte data a nalezne nové řádky, které po předchozím čtení vložila jiná transakce.

20.5.1. Uzamykání prostředků

Předchází problémům se souběžným zpracováním tak, že zabraňuje sdílení prostředků. Užívá se přitom uzamykání dat, která jsou načtena za účelem aktualizace. Funguje tak, že jedna transkace musí zpracovát data a pokud přijde druhá, tak musí vyčkat, nežli první transakce dokončí zpracování dat, následně si data uzamkne a pracuje druhá transakce. Zámky může umístit databázový systém (implicitní zámky), nebo aplikace či uživatel pomocí příkazu (tzv. explicitní zámky). Zámky mohou být na úrovni řádků, stránky, tabulek nebo celé databáze, v literatuře se velikost zámku se označuje jako granularita zámku. Zámky s větší granularitou se snáze administrují, ale často způsobují konflikty a snižují celkový výkon databázového systému. Zámky s malou granularitou se obtížně administrují (systém řízení databáze musí sledovat a kontrolovat mnohem více detailů), ale snižuje se počet konfliktů a zvyšuje propustnost databázového řešení. Zámky máme dvou druhů, výhradní zámek uzamyká položku před vstupem libovolného typu, příslušná data nemůže načíst ani měnit žádná jiná transakce, ale sdílený zámek uzamkne položku tak, aby ji nebylo možné měnit, ale zůstává přístupná pro čtení.

Když dochází k souběžnému zpracování dvou nebo více transakcí, měly by být výsledky v databázi logicky konzistentní s výsledky, kterých bychom dosáhli při sériovém zpracování transakcí v libovolném pořadí. Jedna z metod, jak dosáhnout serializace je dvoufázové uzamykání, při kterém mohou transakce získávat zámky podle potřeby, ale po uvolnění prvního zámku již nemohou získat žádný další zámek. Transakce procházejí rostoucí fází, kdy získávají zámky a klesající fází, kdy jsou zámky postupně uvolňovány. V některých databázových systémech to pracuje malinko odlišně, transakce získává zámky, ale žádné nejsou uvolněny do zadání příkazu COMMIT nebo ROLLBACK (je restriktivnější, ale jednodušší na implementaci), následně jsou všechny zámky uvolněny naráz.

Deadlock nastává, pokud dvě transakce čekají vzájemně na data, která si uzamkla ta druhá. Řešením je předcházení, nebo při jeho nastání přerušení výkonu obou transakcí. Jedním z možných řešení je, aby si uživatel uzamkl všechny prostředky hned na začátku najednou a následně provedl manipulaci s daty. Další možností, jak zabránit uváznutí je, že všechny aplikační programy musí prostředky uzamykat ve stejném pořadí, což vede k zastavení výkonu jedné z transakcí do dokončení té druhé. Prakticky všechny databázové systémy jsou vybaveny algoritmy detekce uváznutí, běžným řešením je přerušení jedné z transakcí a vrácení všech změn, které provedla, tedy voláním ROLLBACK.

Optimistické a pesimistické uzamykání jsou dva různé přístupy k uzamykání položek. Optimistické uzamykání předpokládá, že nedojde k žádnému konfliktu, proto načteme data, zpracujeme transakci a poté zkontrolujeme, zda nedošlo ke konfliktu. Jestliže, se konflikt neobjevil, tak se transakce dokončí, v případě konfliktu se transakce zruší voláním ROLLBACK a okamžitě opakuje, dokud není zpracována bez konfliktu. Celou fintou je na začátku uložení

dat do proměnných a do WHERE klauzule příkazu update se uvede podmínka, že hodnota dat v proměnné a tabulce se musí rovnat. Nevýhodou je, že při vysoké aktivitě na řádku je nutné transakci mnohokrát opakovat, což vede k nárůstu času i spotřeby systémových prostředků potřebných pro zpracování.

Pesimistické uzamykání, předpokládá, že ke konfliktu dojde. Požádáme systém o zámky, zpracujeme transakci a následně zámky uvolníme. Nevýhodou je, že transakce drží poměrně dlouho zámek, což je znatelné hlavně při velké granularitě.

Deklarace vlastního zámku je dnes již možná ve všech databázových systémech. Databázové systémy nezadávají zámky explicitně. Programy vyznačují hranice transakcí a následně deklarují typ uzamykání, který od databázového systému požadují. Pokud je potřeba změnit zamykací chování, tak se změní deklarace zámků, která se provádí pomocí systémového parametru (viz dokumentace daného databázového systému).

20.5.2. Úroveň izolace transakcí

Norma ANSI SQL z roku 1992 definuje 4 úrovně izolace, na nichž závisí, které problémy s kontrolou souběžností smějí nastat. Izolační úrovně slouží k automatizované správě zámků databázovým systémem.

	Úroveň izolace			
Typ problému	**Nepotvrzené čtení**	**Potvrzené čtení**	**Opakovatelné čtení**	**Serializovatelné**
Nečinné čtení	ANO	NE	NE	NE
Neopakovatelné čtení	ANO	ANO	NE	NE
Fantomní čtení	ANO	ANO	ANO	NE

Nápisy ANO a NE v tabulce znamenají, co je v dané úrovni izolace povoleno. Obecně platí, že čím je úroveň izolace restriktivnější, tím je menší propustnost celého řešení. Ne všechny databázové systémy podporují všechny 4 uvedené úrovně izolace, proto je při volbě databázového systému nutné číst dokumentaci. Nejčastěji se v praxi používá úroveň Opakovatelného čtení.

20.5.3. Kurzory

Kurzor je ukazatel na sadu řádků, které představují množinu výsledků příkazu SELECT. Deklarace kurzoru je jednoduchá: DECLARE CURSOR *název* AS SELECT… Jakmile aplikační program otevře kurzor, může jej umístit na určené místo sady výsledků (tradičně na začátek nebo konec). Transakce může otevřít několik kursorů (postupně, nebo současně) a zároveň je možné v jedné tabulce otevřít dva či více kurzorů buď přímo, nebo v pohledu. Kurzory jsou značně paměťově náročné, proto je potřeba jejich užívání opravdu přemýšlet. Základní druhy kurzorů jsou:

- Dopředné – aplikace může procházet záznamy pouze směrem vpřed a změny provedené jinými kurzory jsou vidět jen tehdy, pokud jsou provedeny před kurzorem.
- Navigační – aplikace může procházet záznamy dopředu i zpět.

Rozlišujeme několik základních typů kurzorů.

Statický kurzor pořizuje snímek relace a následně jej zpracuje. Viditelné jsou změny provedené tímto kurzorem, ale nikoliv změny mající jinou příčinu. Statický kurzor řadíme mezi navigační kurzory.

Dynamický kurzor poskytuje všechny možnosti. Dovoluje sledovat všechny vložení, aktualizace, odstranění a změny pořadí řádků. Pokud transakce nepovoluje nečisté čtení, jsou vidět jen potvrzené změny.

Klíčový kurzor kombinuje funkce statických a dynamických kurzorů. Po otevření jsou pro každý řádek uloženy hodnoty primárních klíčů, a když aplikace umístí kurzor na řádek, tak jsou pomocí primárního klíče načteny hodnoty. Nejsou viditelné řádky nově vložené jinými kurzory, a pokud transakce nepovoluje nečisté čtení, jsou vidět jen potvrzené změny.

Obecně platí, že náklady na zpracování rostou dle následujícího typu kurzorů: statický, klíčový a dynamický. Pro optimalizaci výkonu by měl vývojář vymýšlet takové kurzory, které přesně provedou jen požadované. Je důležité znát dokumentaci kolem kurzorů od konkrétního systému a je potřeba pro optimalizaci důsledně uvádět typ kurzoru i izolaci transakce. Příkladem může být následující deklarace.

```
DECLARE soucet CURSOR FOR
       SELECT Zakaznik_ID, SUM(Cena_za_kus)
       FROM dbo.Objednavky JOIN dbo.Polozky_objednavky
                           ON dbo.Objednavky.ID = dbo.Polozky_objednavky.Objednavka_D
                           JOIN dbo.Vyrobky
                           ON Vyrobky.ID = Polozky_objednavky.Vyrobek_ID
       GROUP BY Zakaznik_ID
FOR READ ONLY
```

20.5.4. Zabezpečení databáze

Přístup uživatele dělíme na dvě části, ověřování (zjištění, zda má uživatel právo k přístupu) a autorizaci (přidělení oprávnění). Přihlášení k systému nestačí k použití DB, uživatel musí mít přidělena práva na práci s danou databází. K vynucování práv je možné užít funkce databázového systému, nebo doplňky v aplikační vrstvě.

20.5.4.1. Uživatelské účty

Pro MS SQL je možné nechat ověření uživatele provést OS (pokud je databázový server instalovaný na Windows), nebo je možné, aby k databázi přistupoval pomocí samostatného uživatelského jména a hesla, které je pracováváno přímo databázový serverem nezávisle na systému. Uživatele lze přiřadit jedné, nebo více rolím, zároveň role může zahrnovat jednoho, nebo více uživatelů. Uživatelské role a objekty mají mnoho oprávnění, každé se přiřazuje jedné roli či jednomu uživateli a jednomu objektu. Administrátor systému obecně není uživatel a neměl by mít přístup k uživatelským datům. Oprávnění se vždy přidělují uživatelským rolím (skupinám), nikoliv přímo jednotlivým uživatelům. Uživatel je nejprve vytvořen na úrovni systému řízení databáze a následně je pro danou databázi "znovu" vytvořen se stejným uživatelským jménem (heslo se již nezadává). V rámci konkrétní databáze uživateli přidělíme role.

Pevná databázová role	Oprávnění specifické pro DB	Oprávnění k serveru
db_accessadmin	Alter any user, create schema, conenect	View any database
db_backupoperator	Backup database, backup log, checkkpoint	View any database
db_datareader	select	View any database
db_datawriter	Delete, insert, update	View any database
db_dlladmin	Viz dokumentace	View any database
db_denydatareadr	select	View any database

Pevná databázová role	Oprávnění specifické pro DB	Oprávnění k serveru
db_denydatawriter	Delete, insert, update	View any database
db_owner	control	View any database
db_seccurityadmin	Alter any application role, alter any role, create schema, view definition	View any database

Povinnosti uživatelů nelze vymáhat databázovým systémem, ale musí jim být vyloženy na školení. Jejich vymáhání je čistě v rovině administrativní. Za zprávu práv a dodržování povinností s nimi spojenými je odpovědný administrátor databáze.

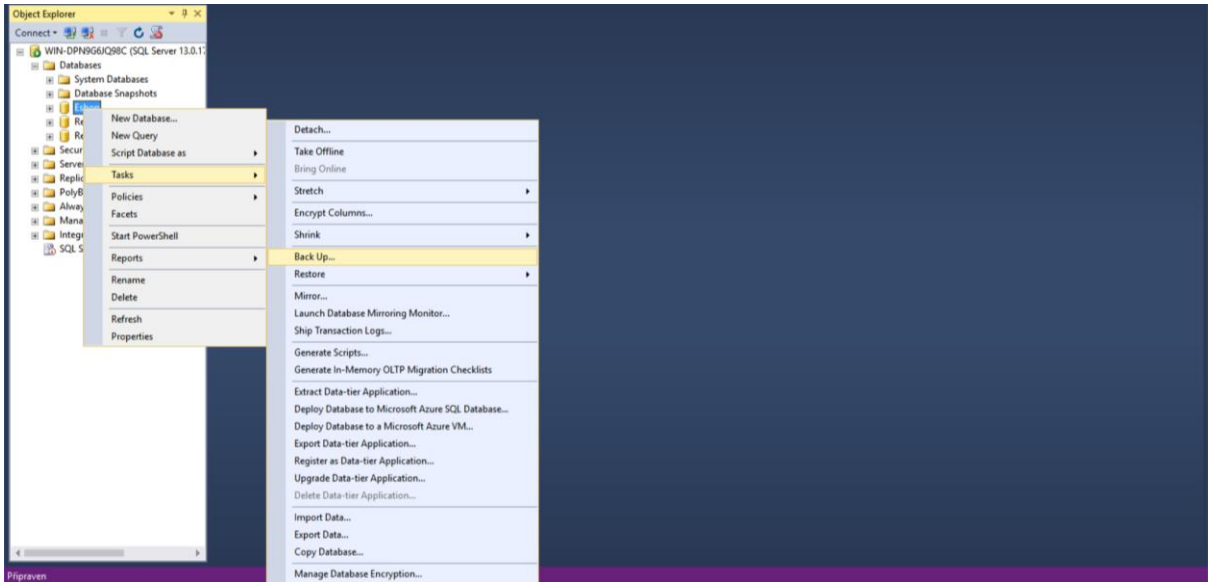
20.5.4.2. Zabezpečení na úrovni databázového systému

Je nutné systém provozovat za firewallem, protože jako takový obsahuje velmi citlivá data a žádný program není bez chyby. S chybami SW souvisí i nutnost vždy bez odkladů aplikovat bezpečnostní aktualizace pro OS a databázový systém. Nepovolovat, aby bylo možné s databází manipulovat mimo firemní síť (vyjímaje e-shopy a další webové aplikace). Protože webové aplikace manipulují s databázemi mimo podnikovou síť, je potřeba aby podnikový webserver fungoval na samostatném stroji a měl k sobě vlastní databázový server. Další důležitý bezpečnostní prvek je povolení jen skutečně aktivně využívané protokoly pro komunikaci s DB. Systémové uložené procedury, které nejsou využívány, je vhodné odebrat z provozní databáze, aby nemohlo dojít ke zneužití případných chyb v nich obsažených. Jedním z velkých bezpečnostních rizik je uživatel, proto je potřeba při zabezpečení databází pomatovat. Počítač určený ke konfiguraci databázového systému by měl být uložen samostatné místnosti a přístup k němu by měl být protokolován, samozřejmostí je kontrola vzdáleného přístupu a pravomoci jej udělit.

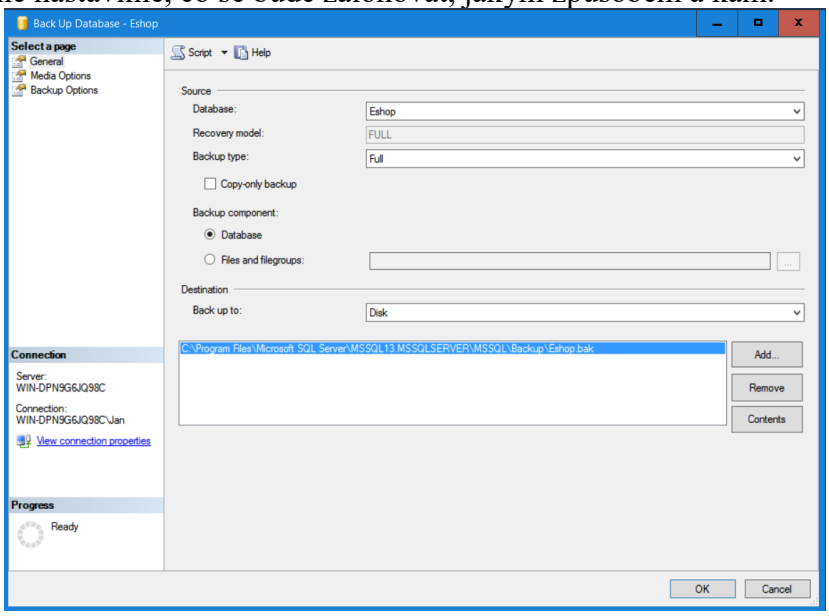
Zabezpečení na úrovni aplikace slouží především ke scénářům, kde obecné zabezpečení na úrovni DB systému již nestačí. Aplikační zabezpečení webových aplikací řeší webserver, i u něj nesmíme opomínat instalovat bezpečnostní aktualizace a dodržovat bezpečnostní doporučení pro provozování webových aplikací. Zabezpečení aplikačním kódem je pouze doplňkové ke službám DB serveru, nejde o primární způsob zabezpečování.

20.5.4.3. Záloha a obnovení

Zálohování databázových strojů není jednoduše přímočaré. Databázové systémy jde zálohovat spolu s mateřským OS, ale zde nastává problém s integritou dat. Jednotlivé databáze jde zálohovat samostatně, buď manuálně, nebo pomocí plánované úlohy. Manuální zálohu popisují následující snímky. V MS Management studiu v Object Exploreru klikneme pravým tlačítkem myši na databázi.



Následně nastavíme, co se bude zálohovat, jakým způsobem a kam.



Záloha má podobu souboru formátu bak v případě MS SQL Servereu.

Obnovení databáze se standardně děje pomocí dvou postupů, jedním je obnovení založené na opakovaném zpracování a druhým obnovení pomocí vrácení zpět a postupu vpřed. Obnovení založené na opakovaném zpracování pracuje následovně. Vrátíme se k poslednímu potvrzenému zpracování, které se uložilo na disk a zátěž provedeme znovu od tohoto místa. Využívá se transakční žurnál, který ukládá záznam o všech transakcích zpracovaných od posledního uložení na disk. Při obnovení dojde ke znovu zpracování všech commit transakcí provedených po posledním uložení dat na disk. Nevýhodou je doba zpracování, systém by nemusel stihnout zpoždění dohnat. Při souběžném zpracování transakcí probíhají události asynchronně, což komplikuje použitelnost tohoto typu obnovy u více uživatelských systémů. Z popisu je zjevné, že se jedná o postup obnovy databáze po výpadku databázového serveru, respektive HW. Transakční log se ukládá na disk velmi rychle, a ne v tak komplikované struktuře na disk, aby nedošlo při výpadku HW k jeho ztrátě.

Obnovení pomocí vrácení zpět a postupu vpřed také využívá transakční žurnál. Uchovává se i protokol změn, které transakce učinily od posledního uložení databáze. V případě postupu vpřed se databáze obnoví z uloženého stavu a znovu se aplikují všechny změny pro-

vedené potvrzenými transakcemi z dat uložených v protokolu (transakce se znovu neprovádějí). V případě postupu vrácení zpět napravujeme chyby způsobené chybnými nebo částečně zpracovanými transakcemi tak, že se provede ROLLBACK na změny v databázi, které tyto transakce provedly a následně se provedou znovu spuštěny platné transakce, které probíhaly v okamžiku selhání. Obě metody vyžadují vedená protokolu s výsledky transakcí. Protokol obsahuje záznamy o změnách dat v chronologickém pořadí. Transakce se do protokolu zapíše před aplikováním na DB, což je klíčové pro její případné obnovení. Předchozí snímky jsou záznamy dat před provedením transakce uložené v protokolu (používají se pro metodu vrácení zpět), zatím co následné snímky jsou kopie dat po provedení změn uložené v protokolu (pro metodu postup vpřed). Protokol typicky obsahuje:

- Čas akce
- Typ operace
- Zpracovaný objekt, např.: typ a identifikátor záznamu
- Předchozí a následné snímky

Kontrolní body umožňují urychlení zpracování tím, že je to synchronizační bod mezi uloženou databází a protokolem. Při vytváření kontrolního bodu systém odmítá nové požadavky, dokončí zpracování běžících transakcí a zapíše vyrovnávací paměť na disk, po jeho dokončení je do protokolu zapsán kontrolní bod a systém pokračuje v činnosti. Vytvoření kontrolního bodu je poměrně nenáročné a provádí se obvykle v intervalu 10 až 20 minut. Většina DB systémů vytváří kontrolní body automaticky.

Ze záložního souboru bak je možné pomocí grafického rozhraní v MS Mamanagement Studiu obnovit databázi obdobně, jako jsme vytvářeli zálohu, průvodce je velmi jednoduchý a přehledný.

20.5.4.4. SQL server hacking

Chování instance SQL serveru je dáno účtem, pod kterým běží, pro běh SQL stačí obyčejný doménový účet Windows. SQL server využívá virtuální účty v OS pro automanagement a umí si registrovat SPN. Pro nabourání je většinou je potřeba lokální administrátorský přístup k Windows, kde běží SQL. K nastavení je Psexec slouží k přepnutí příkazového řádku v kontextu SQL serveru.

Služba SQLWriter má sys admin oprávnění a pomocí regedit lze upravit cesta k binárce této služby. Parametr -E pro přihlášení se k SQL serveru je ověřování systém Windows účtem. Ve verzi 2016 je v základu nastaveno ověřování pouze pomocí Windows účtu, je potřeba přepnout na kombinaci Windows SQL server účtů (pomocí registru (změnit z 1 na 2)), nebo toto nastavení zvolit při instalaci.

Pro spouštění SQL serveru máme startup parametr -m je single user režim a parametr -F je minimální spuštění. Pokud SQL je spuštěn s parametrem -m stačí sqlcmd zadat do příkazového řádku v režimu lokálního administrátora a je k dispozici příkazový řádka sysadmin. Při spuštění s parametrem -F je vynechán login ochrana a spouštěcí konfigurace, tato konfigurace naběhne i po kolapsu databáze nebo s nedostatkem paměti.

Je možné nastavit, aby v případě problému se zápisem do auditu došlo ke zrušení dané operace, nebo se server natvrdo vypíná, nebo může danou situaci ignorovat. Pokud je šifrovaná libovolná uživatelská databáze, tak se šifruje temp a backup a bez certifikátu není možné zobrazit ani náhled zálohy. Šifrování a dešifrování běží na úrovni IO operací.

Pomocí příkazové řádky v SQL 2016 je možné DB komprimovat i šifrovat v záloze. Row Level Security je novinkou ve verzi 2016 a umožňuje nastavit, kterému uživateli se zobrazí, které řádky a od verze 2016 Enterprise lze data šifrovat pomocí Advance Data Encription.

Pokud používáme uživatelské účty SQL serveru, tak pomocí pohledu Sp_help_revlogin se dostaneme k záloze a obnově uživatelských účtů s původními hesly.

20.5.5. Execution plan

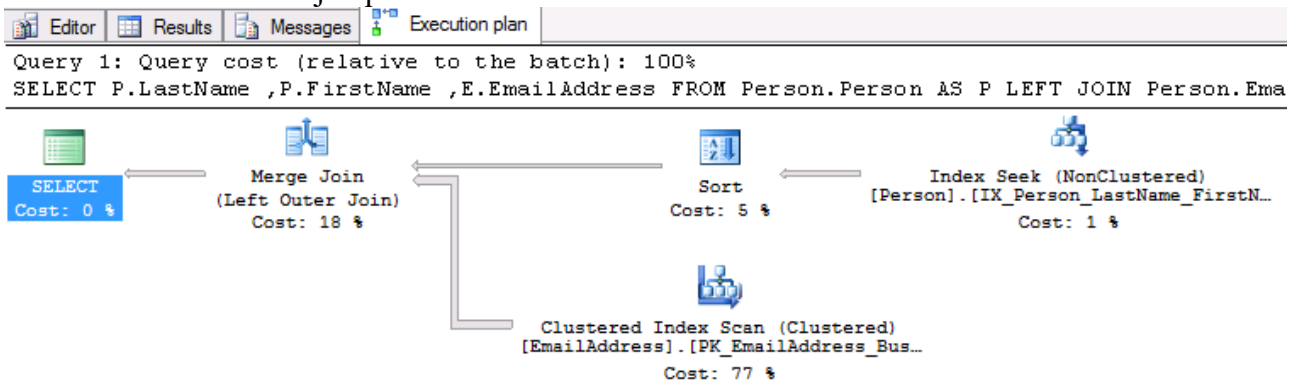
Jak jsme si již řekli, jazykem SQL nedefinujeme serveru, co má přesně krok za krokem provést, abychom získali naše data, ale SQL dotazem pouze popíšeme, co chceme od serveru získat. SQL server musí náš dotaz pochopit a najít algoritmus, jehož provedením nám poskytne požadovaná data. Právě v postupu hledání algoritmu se hodně liší Oracle a Microsoft servery, protože Oracle se snaž za každou cenu nalézt ten nejlepší algoritmus, zatím co Microsoft se spokojí s nalezením uspokojivého algoritmu. To, jak MS SQL server pozná, že má již uspokojivý algoritmus, je knowhow výrobce, ale z těchto rozdílů je jasné, že pro méně často opakované dotazy je Microsoft rychlejší, nežli Oracle, kterému jdou lépe shodné dotazy, které nemají podobu uložené procedury, v případě uložených procedur jsou na tom oba výrobci prakticky shodně. Execution plán je zobrazení algoritmu, kterým se na konec server rozhodl vytvořit odpověď na náš dotaz.

Estimated Execution Plan je seznam operací, které je potřeba vykonat. Tento plán je celý postaven na odhadech z dostupných metadat tabulek. Udávané počty řádků jsou pouze vypočteny na základě dostupných statistik. Plán fakticky není proveden, proto se počty odhadnutých řádků mohou lišit od počtů skutečných řádků, které dotaz vrátí při skutečném spuštění.

Actual Execution Plan je naopak plán, který byl získán při spuštění dotazu. Počty řádků nejsou vypočteny, ale změřeny při provádění operací. Pro zobrazení aktuálního exekučního plánu v SQL Management Studio stiskněte CTRL+M nebo menu Query volba Include Actual Execution Plan a spustíme dotaz. Následující příklad využívá testovací databázi AdventureWorks2016CTP3.

```
SELECT P.LastName, P.FirstName,E.EmailAddress
FROM
Person.Person AS P LEFT JOIN Person.EmailAddress AS E
                        ON E.BusinessEntityID = P.BusinessEntityID
WHERE P.LastName = 'Smith'
```

Uvidíme následující plán.



První operací je vyfiltrování řádků v tabulce *[Person].[Person]* podle zadané podmínky *LastName = ‚Smith‘*. Pro vyhledání řádků se použije index *IX_Person_LastName_FirstName_MiddleName*, který obsahuje sloupec *LastName*. Obdržená data je potřeba setřídit podle pole *BusinessEntityID*, proto je druhou operací *Sort*. SQL Server si tedy připravil první sadu dat, která obsahuje pole *LastName* a *FirstName*. Jelikož se v dotazu vyskytuje pole *EmailAddress*, je potřeba projít tabulku *[Person].[EmailAddress]*. Tato operace je v plánu označena jako *Clustered Index Scan* podle primárního klíče, který se skládá z polí *BusinessEntityID* a *EmailAddressID*. Máme tedy druhou sadu dat, která je shodou okolností již seřazena podle *BusinessEntityID*. Předposlední operací tohoto plánu je *Merge Join*, který má na vstupu dvě sady stejně seřazených dat a provádí jejich sloučení. V případě, že k záznamu z tabulky *[Person].[Person]* neexistuje záznam v tabulce *[Person].[EmailAddress]*, zobrazí se na výstupu ve sloupci *EmailAddress* hodnota *NULL*. Poslední operací toho plánu je zobrazení dat na výstup. Spojnice mezi jednotlivými operacemi značí počet řádků, které jsou přečteny z tabulek, popř.

počet řádků, které vyprodukuje konkrétní operace na výstupu. Procentní hodnota nad každou operací značí její náročnost z hlediska CPU+I/O.

Po ukázání myší na jednotlivé položky plánu v Management studiu můžeme vyčíst předpokládané vs. aktuální počty řádků a jejich náročnost z hlediska CPU (Est CPU Cost) nebo IO (Est IO Cost). Obě jsou bezrozměrné veličiny. Obecně *Cost* je bezrozměrná veličina, která slouží k relativnímu srovnání jednotlivých operátorů jak v rámci jednoho plánu, tak i různých plánů navzájem.

Exekuční plán je velmi užitečným pomocníkem pro ladění dotazů a zkoumání, jak SQL Server získává data z tabulek. Ladění dotazů je důležité především pro častěji se opakující dotazy a ostré provozní databáze, rozhodně se nejedná o problematiku úplných začátečníků v jazyce SQL, to stejné, jako když bychom chtěli kvalitní HW optimalizaci zdrojového kódu od začínajícího programátora.

20.5.5.1. Statistiky MS SQL serveru

V úvodu do Execution plánů zaznělo, že k jejich vytvoření se používají statistiky, proto si pojďme něco málo o nich povědět. Statistika je interní tabulkou serveru, která má maximálně 200 řádek, kdy každý řádek obsahuje:

- RANGE_HI_KEY – Horní mez intervalu kroku histogramu.
- RANGE_ROWS – počet řádků v intervalu, mimo horní mez
- EQ_ROWS – počet řádků odpovídající horní mezi intervalu
- DISTINCT_RANGE_ROWS – počet jedinečných hodnot v intervalu, mimo horní mez
- VG_RANGE_ROWS – průměrná hodnota počtu jedinečných hodnot, mimo horní mez. Tedy RANGE_ROWS / DISTINCT_RANGE_ROWS

Pokud v exekučním plánu vidíme velmi rozdílnou hodnotu v údaji Estimated Number of Rows a Actual Number of Rows, pak máme problém se statistikami.

Pokud je na úrovni SQL serveru zapnuto Auto Create Statistics a Auto Update Statistics (a tyto hodnoty by měly být zapnuty), pak pokud server při vytváření exekučního plánu zjistí chybějící statistiky, respektive zastaralé statistiky, tak dojde k vytvoření potřebné statistiky, respektive k aktualizaci zastaralé statistiky. To může zdržovat při vykonávání dotazu. Odchytit update statistik při vykonání dotazu je možné Profilerem – událost Performance/Auto Stat.

Při rebuildu indexu dojde k aktualizaci statistik s ním spojených. V tomto případě se provede fullscan, tedy statistiky se vypočtou ze všech řádků a dostaneme tak nejpřesnější možné statistiky. Po tomto není vhodné aktualizovat statistiky s implicitním nastavením, výsledek by byl horší než při použití fullscan.

Server automaticky (pokud je zapnuto Auto Create Statistics) vyrábí statistiky, které potřebuje pro vytvoření plánu vykonávání dotazu. Statistiky vyrábí jen pro jednotlivé sloupce. Automaticky není schopen vyrobit statistiky pro kombinací sloupců. V určitých případech může být vhodné takovéto statistiky vytvořit manuálně, nebo za použití Database Tuning Advisor. Nyní je jen otázka, jak zjistíme informace o statistikách přímo pro naši databázi, respektive tabulku. Mám v zásadě 3 možnosti, jak informace o statistikách dostat:

- SSMS – Object Explorer – statistics – properties – Details.
- Systémových pohledech **sys.stats** a **sys.stats_columns**
- DBCC SHOW_STATISTICS(<table_name>, <stats_name>) [WITH HISTOGRAM]

S vytvořením filtrovaného indexu se vytvoří i filtrovaná statistika. Filtrované statistiky je možné vytvářet i manuálně. Výhodou je možnost obejít limit 200 řádků statistiky tak, že vytvoříme dvě statistiky v různých intervalech hodnot. Například jednu pro 90 % historických dat a jednu podrobnější pro 10 % aktuálních a často používaných dat. Nevýhodou pak je, že pokud používáme filtrované statistiky nad sloupcem a pokud server zjistí, že chybí nefiltrovaná

statistika, pak nefiltrovanou statistiku automaticky vytvoří. S tím souvisí režie s udržováním dvojích statistik. Doporučuje se ponechat zapnuté automatické tvoření statistik a u nefiltrované statistiky vypnout automatický přepočet (nastavit NORECOMPUTE) a nechat ji zestárnout, server pak bude používat aktuální filtrované statistiky.

Automatický update statistik se řídí nastavením serveru:

- Auto Update Statistics – zapíná automatickou aktualizaci
- Auto Update Statistics Asynchronously – zapíná asynchronní aktualizaci statistik, implicitně vypnuto

Synchronní aktualizace – Pokud se při sestavování plánu zjistí zastaralá statistika, pak se okamžitě provede její aktualizace. Výhodou je plán na základě vždy aktuálních statistik. Nevýhodou je, že aktualizace statistik zdržuje sestavení plánu. Toto nastavení se doporučuje. Asynchronní aktualizace – Plán se sestaví na základě dostupných statistik. Pokud se zjistí zastaralá statistika, pak je asynchronně odpálena její aktualizace. Výhodou je, že sestavení plánu není zdržováno výpočtem statistik. Nevýhodou je, že plán může být postaven na základě neaktuálních statistik. Aktualizace statistik se pak provede při snížené zátěži serveru.

Statistiky jsou zastaralé když:

1. Tabulka má více než 500 řádků a je modifikováno více než 20 % řádků + další 500 řádků
2. Tabulka má méně nebo rovno 500 řádků a je modifikováno 500 řádků
3. Pokud se počet řádků tabulky změní z 0 na nenulový počet
4. U temporarních tabulek po každých změněných 6 záznamech

Filtrované statistky se vyhodnocují bohužel podle počtu modifikací všech řádků, a nejen těch řádků zahrnutých do filtru. Z toho vyplývá nutnost manuální aktualizace filtrovaných statistik. Zda jsou statistiky zastaralé, se vyhodnocuje při sestavování plánu vykonání dotazu.

Chybějící statistiky je možné odhalit profilerem zachycením události Errors and Warnings/Missing Column Statistics. Příčinou může být vypnutá volba serveru Auto Create Statistics. Druhou příčinou může být používání tabulkových proměnných, pro které server statistiky nevyrábí (odhadovaný počet řádků je vždy 1 respektive 0 při nesplnitelné podmínce). Mělo by být pravidlem, že tabulkové proměnné se používají jen pro tabulky do 100 řádků. Pro větší tabulky je lépe využívat temporární tabulky (s ‚#' na začátku názvu).

Nemožnost použít statistiky, se vyskytuje, při použití proměnných v podmínkách výběru nemůže server vyžít statistiky, protože nezná hodnotu proměnné v době sestavování plánu. Pro odhadovaný počet řádků tak server použije průměrnou hodnotu z celé tabulky, která se může diametrálně lišit od hodnoty odpovídající hodnotě proměnné, můžeme tak dostat chybný plán vykonání.

Pro 4 000 000 záznamů dostaneme 200 řádků statistiky po 20 000 záznamech, pokud k této, nebo podobné, situaci dojde, hovoříme o příliš hrubých statistikách. Z toho může být 90% historických dat, pro které se statistiky nemění. Řešením je udělat dvě statistiky. Jednu pro historická data, tu není potřeba často udržovat. A jednu pro aktuální živá data. Statistika pro 10 % živých dat tak bude jemněji dělena a bude muset být manuálně častěji udržována aktuální.

Nedostatečné automatické aktualizace nastávají při 20 % modifikovaných řádků, kdy je málo pro odpálení automatické aktualizace u tabulek s velkým počtem řádků. Nejhorší je to u primárních klíčů, které přibývají na konci řady, tedy mimo statistku a server je tak nemůže vyhodnotit. Zde je nutná častější manuální aktualizace statistiky.

Statistiky jsou používány pro odhad velikosti potřebné paměti pro některé druhy joinu. Pokud je odhad podhodnocen, pak je místo chybějící paměti využívána tempdb a dochází k cca desetinásobnému zpomalení. Tento problém je možné zachytit profilerem jako události Errors and Warnings/Sort Warnings a Errors and Warnings/Hash Warnings.

20.5.5.2. Trubleshooting SQL pomocí Extended Events

SQL Trace a SQL Profiler jsou hodně zastaralé a nerozvíjené nástroje, proto je velkou chybou spuštění Profileru na produkčním serveru při hledání problému. Rozvíjeným nástrojem vhodným právě pro tyto situace jsou Extended Events, které jsou dostupné od verze SQL serveru 2008. Oproti Profileru je tento nástroj méně zatěžující pro HW a umožňuje jednodušší zpracování. Extended Events dokáží zachytit execution plan a patřičný dotaz, oproti tomu profiler sbírá úplně všechno a umožňuje nastavit jen co se má zobrazit, proto je mnohem náročnější na spotřebu systémových prostředků.

Všechny novinky dostupné od SQL serveru 2012 se nezobrazují v Profileru, proto je nutné při užívání nových funkcí SQL serverů zapomenout na Profiler a naučit se pracovat s Extended Events. Od SQL serveru 2012 má tento nástroj dostupné také UI.

Existuje volně dostupný zdrojový kód, který z definice Trace na XEvents. Získaná data jsou vidět s drobný spožděním (cca 3 až 5 sekund), audity sbírané v moderních MS SQL serverech jsou sbírány pomocí XEvents. Pojmem Ring buffer rozumíme uložení všech dat do paměti a Event file je uložení těchto dat do souboru. Bezpečnostní informace jsou uloženy samostatně na stejném místě, jako Error log. Ve složce Temp má obsahovat, pokud je 8 a méně procesorových jader, stejný počet souborů XEvents jako je celkový počet jader procesorů. Soubory XEvents v tempu mají mít stejnou velikost.

20.5.6. SQL Server Data Tools

Je integrován ve Visual studiu od verze 2015 do všech edic. Pokud je neinstalovaný, je možné jej doinstalovat pomocí instalačního souboru. Pro starší verze Visual studia je ke stažení zdarma formou doplňku. Hlavní výhodou je offline vývoj, tedy možnost vyvíjet bez nutnosti připojení k serveru a umožňuje porovnat schéma dvojice databází a také migrovat změny schémat databází a porovnávat data. Často slouží k vývoji BI řešení. Nikdy nepracují přímo s databází, sestaví si model a pracuje s ním, po dokončení a finalizaci práce se aplikuje model na databázi, se kterou chce vývojář pracovat. Umožňuje pokročilý refaktoring databáze a volit verzi SQL Serveru, na které databáze poběží.

DACPAC je soubor popisující schéma databáze a související objekty, jde o automatický upgrade schématu a nepodporuje data. BACPAC soubor je DACPAC nesoucí v sobě data exportované pomocí BCP, jde mimo jiné o cestu migrace z a do Azure databáze.

Power Buffer sleduje všechny změny, které provádíme nad otevřenými objekty a generuje skript pro propsání změn. Intelisence funguje proti modelu v Power Buffer. Tlačítko update slouží k vytvoření skriptu a je možné změny rovnou propsat do cílové databáze.

Pro offline vývoj je potřeba vytvořit SQL Server Database Project, pro přímou práci s databází je možné pracovat přímo bez nutnosti vytváření projektu. Offline vývoj automaticky verzuje každou změnu ve schématu, čímž je snazší se o nějaký krok vrátit v případě nutnosti.

Nástroj Schema Copare identifikuje a přenáší změny mezi databázemi, nebo databází a projektem. Po porovnání je možné vybrat změny, které se mají prosynchronizovat, proto při pozornosti vývojáře je minimalizováno riziko chyby.

Umožňuje přesun tabulky do jiného schématu. Import dat se řeší pomocí SQL skriptů, které se spouští jako post-build a pro vygenerování je potřeba použít nástroj Data Compare.

Unit testy vyžadují známá testovací data v databázi. Jednotlivé Unit testy běží v transakci, která se rolbackne po dokončení testu. Po testu je potřeba důkladně databázi uklidit, není vzácné dropnutí databáze a její obnovení ze stavu před spuštěním testu.

20.5.7. Columnstore indexy

Rowstore je tradiční uložení dat po řádcích na datových stránkách, zatím co Columnstore je uložení tabulky po sloupcích, což umožňuje dosáhnout efektivní komprese dat a snížení

nároků na IO a paměť, cože je velkou výhodou, protože IO operace je úzké hrdlo všech DB řešení. Cenou za tyto plusy je zvýšená zátěž CPU.

Využití těchto indexů vede k řádovému zrychlení dotazů a řádovému snížení nároků na úložný prostor. V datových skladech jsou dotazy většinou směřované na určité sloupce faktové tabulky, proto je velmi výhodné data ukládat po sloupcích. Umožňuje SQL serveru zpracovávat data z více řádků najednou, což umožňuje dávkové zpracování výsledků.

Je dostupný pouze v Enterprise edici a každá tabulka je dělena na rowgroup a nad tou se provádí komprese podle sloupců. Od verze SQL 2016 je možné mít klastrovaný i neklastrovaný index pro čtení i zápis. Zásadní výhodou ale je, že lze libovolně kombinovat s běžnými indexy a od verze SQL 2016 může být filtrovaný a umožňuje zvolit konkrétní sloupce pro práci. K dispozici je řada katalogových pohledů, takže i vývojářům ušetří nějakou práci.

20.5.8. Pár typů pro administrátory SQL serveru 2016

- Po služby startu si automaticky registruje SPN, což je nutné pro kerberos protokol
- Pro cluster je nutné, aby SQL běžel pod doménovým účtem
- Pokud služba běží pod managed service account dokáže si sama vyměnit hesla, dle politiky
- IFI je oprávnění na serveru, kde běží SQL, pro účet SQL s názvem Perform Volume Maintenance task
- IFI je služba zajišťující postupný nárůst velikosti databázového souboru
- Instalátor SQL 2016 umožňuje nastavit IFI
- Program Files => SQL Server => 130 => Setup Bootstrap => Log => * => ConfigurationFile.ini
- Quietsimple = "True" a IACCEPTSQLSERVERLICENSETERMS="True" se musí doplnit a upravit v konfiguráku
- Do konfiguračního souboru se musí uvést produktový klíč, v klasické instalaci je předvyplněný
- Pokud se zapomene vyplnit klíč, instaluje Developer verzi, nebo Evolutions verzi při instalaci z Enterprise
- Je potřeba změnit výchozí velikostí DB
- MDOP je nastavení paralyzace, ve výchozím nastavení je 0, tedy neomezená a 1 je sériový běh
- Samostatná instance má smysl, kdy je potřeba oddělit od sebe sysadminy a v případě Sharepointu
- Samostatná instance má smysl, když je potřeba vyřešit nedostatečnou dostupnost TempDB
- Contained DB je autorizace na úrovni databáze
- Ms_foreachdb je procedura, která umí iterovat přes všechny databáze
- K hromadné instalaci je možné použít Systém Center nebo MS Deployment Toolkit
- Image SQL serveru
    - Image preparation - install binárek
    - Sysprep
    - Nalití
    - Dokončení instalace na další stanici
- Agent umožňuje pro každý Job step volit oprávnění, se kterými běží
- Do Job operatoru jde zadat jen jeden email, proto je potřeba mít email skupiny
- Není přímá možnost, jak pustit SQL kód pod jiným uživatelem

20.6. MS Access 2016 pokračování

Na tomto místě si ukážeme, jak využít další základní funkce MS Access 2016, protože jsme se věnovali především MS SQL serveru, i když i z něj jsme se dozvěděli jen zlomek základů.

20.6.1. Šipky a jejich význam

V Accessu se při tvorbě dotazů můžeme potkat s tím, že tabulky jsou propojené šipkami. A podle toho, jestli tam šipky jsou, a podle toho, kterým směrem ukazují, se mění výsledek dotazu. Vše si ukážeme na příkladu:

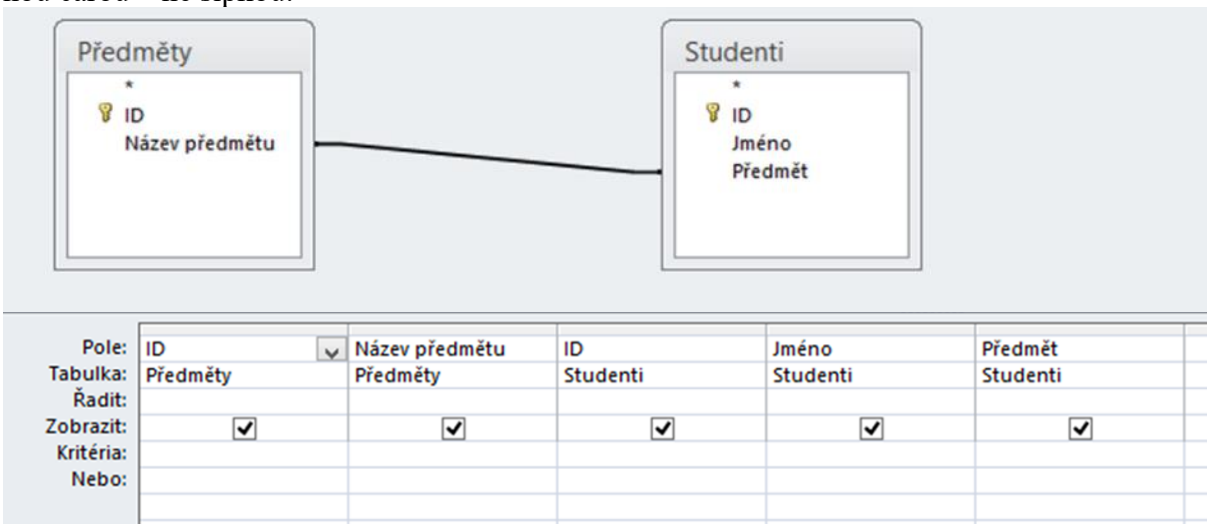
Máme dvě tabulky, nad kterými chceme dělat dotaz. Jedná se o tabulku předmětů ve škole a o tabulku studentů, z nichž někteří na některé předměty chodí. Tabulky vypadají takto:

ID	Název předı	Kliknutím přidat
1	Angličtina	
2	Němčina	
3	Matematika	
4	Fyzika	
5	Špalělština	
(Nové)		

a

ID	Jméno	Předmět
1	Novák	
2	Nováček	Němčina
3	Nováková	Angličtina
4	Nováčková	
5	Novotná	Matematika
6	Novotný	Němčina

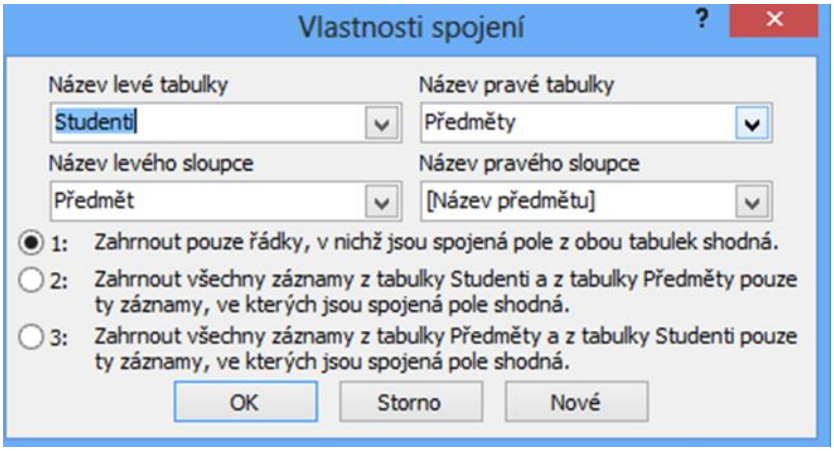
Nad tabulkami chceme vytvořit dotaz, který bude zobrazovat, kteří studenti chodí na které předměty. Základní dotaz vypadá takto. Tabulky v návrhu dotazu jsou propojené obyčejnou čárou – ne šipkou.



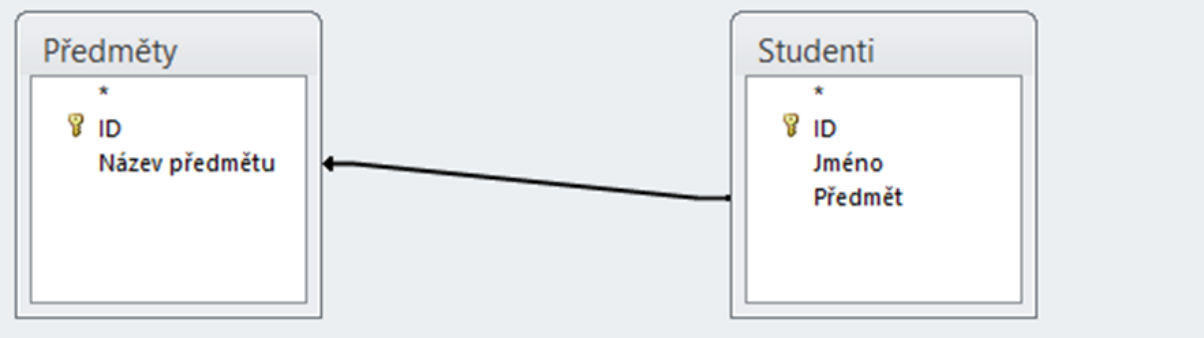
Výsledek vypadá takto:

Předměty.ID	Název předı	Studenti.ID	Jméno	Předmět
1	Angličtina	3	Nováková	Angličtina
2	Němčina	6	Novotný	Němčina
2	Němčina	2	Nováček	Němčina
3	Matematika	5	Novotná	Matematika

Vidíme, že v tabulce nejsou zahrnutí všichni studenti, ani všechny předměty. Jsou tam jen ti studenti, kteří chodí na některý předmět, a ty předměty, na které chodí některý student. To ale můžeme změnit. Klikneme-li na vazbu tabulek pravým tlačítkem a vybereme-li vlastnosti spojení, otevře se následující dialog.



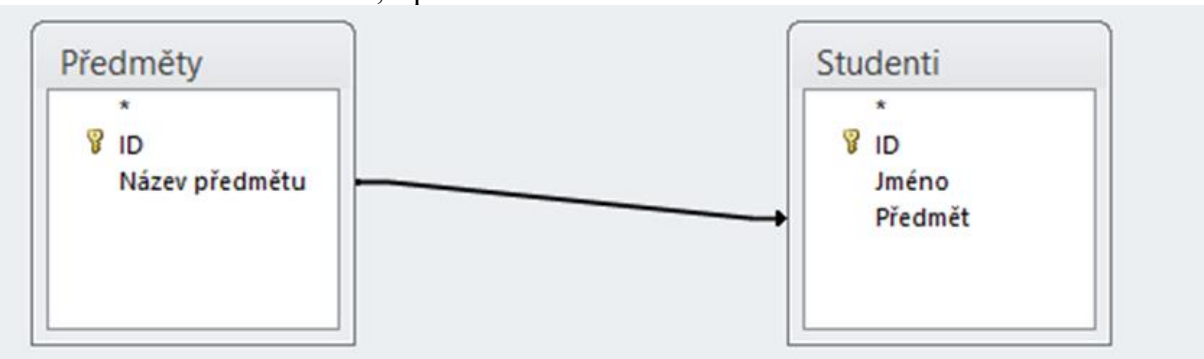
V dialogu jsou velmi dobře popsány všechny tři volby. První volbu už známe. Pokud zaškrtneme druhou volbu, změní se spojení tabulek takto v šipku:



Výsledek dotazu pak vypadá takto:

ID	Jméno	Předmět	*Kliknutím přidat*
1	Novák		
2	Nováček	Němčina	
3	Nováková	Angličtina	
4	Nováčková		
5	Novotná	Matematika	
6	Novotný	Němčina	
(Nové)			

Zobrazili se všichni studenti, bez ohledu na to, jestli chodí na nějaký předmět nebo ne. U studentů, kteří na žádný předmět nechodí, se zobrazilo ve sloupci s předměty prázdné místo. Pokud zaškrtneme třetí volbu, šipka se otočí:



A výsledek vypadá takto:

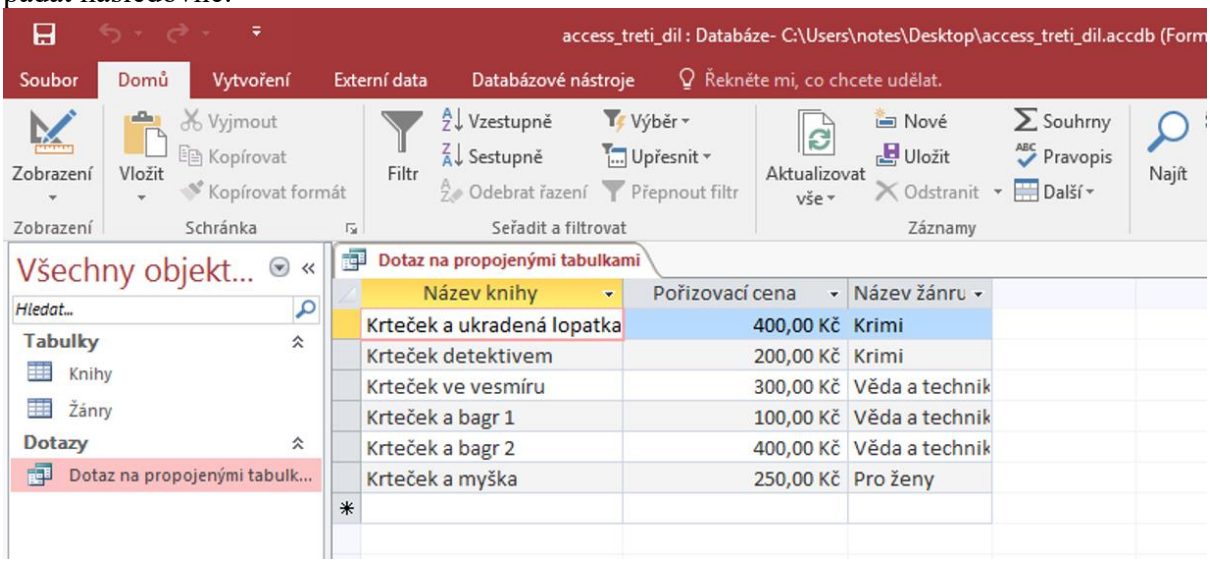
1	Angličtina	3	Nováková	Angličtina
2	Němčina	6	Novotný	Němčina
2	Němčina	2	Nováček	Němčina
3	Matematika	5	Novotná	Matematika
4	Fyzika			
5	Špalělština			

Zobrazily se naopak všechny předměty, bez ohledu na to, jestli je na ně někdo zapsaný.

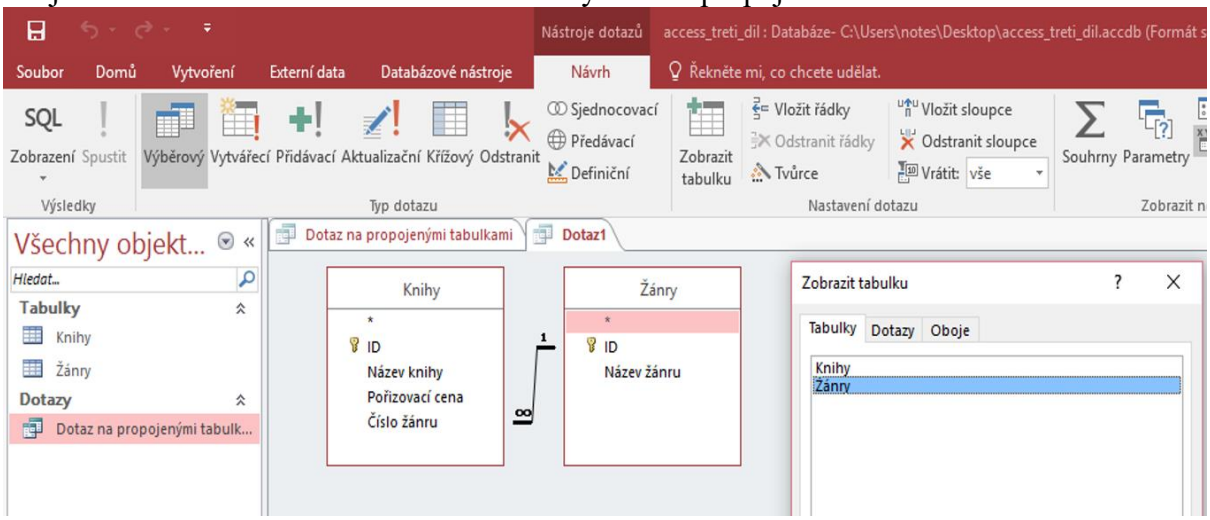
20.6.2. Tvorba dotazu

V dotazu se data neukládají, v dotazu se pouze zobrazují. Ve chvíli spuštění dotazu se databáze "podívá" do tabulek a ukáže aktuální data. Jinými slovy, dotaz vždy potřebuje nějakou tabulku (nebo tabulky), ve které jsou uvedena data dotazem zobrazovaná.

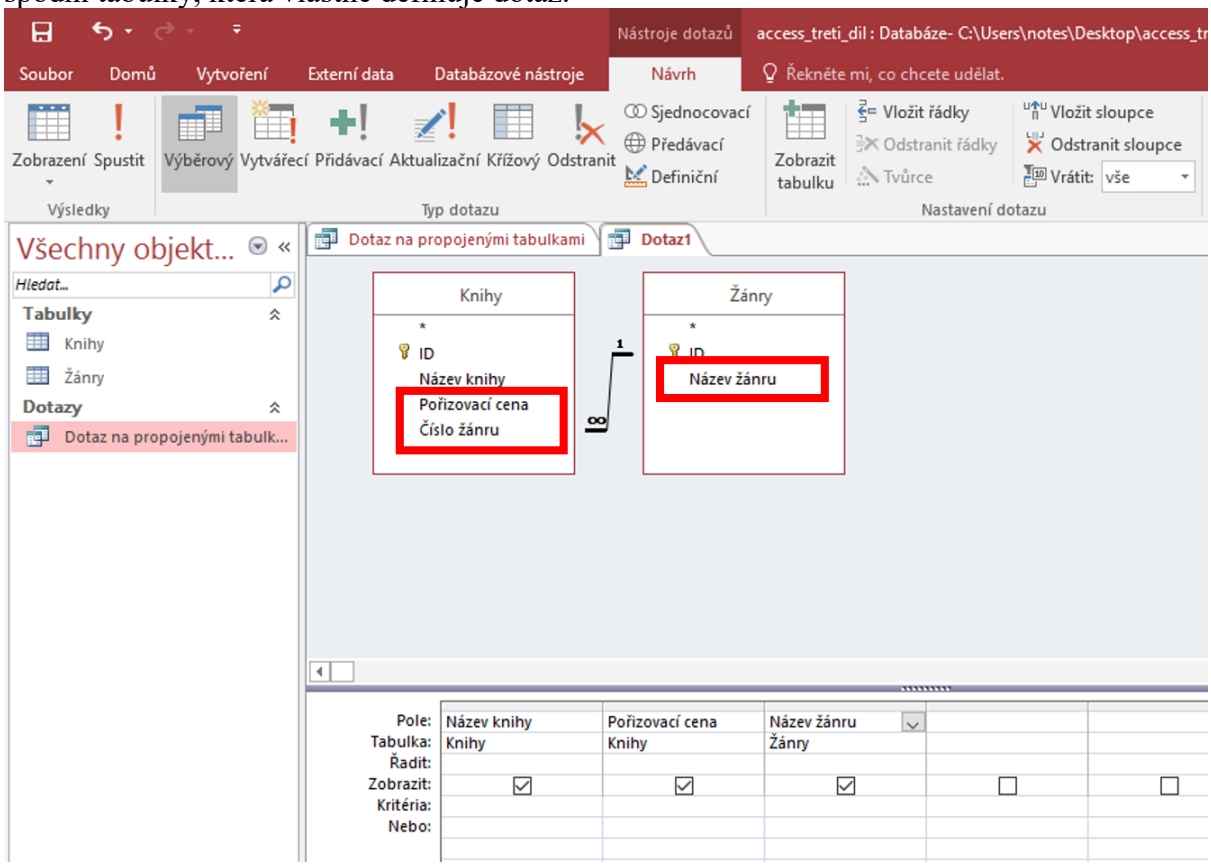
V našem příkladu máme k dispozici dvě propojené tabulky. Naším úkolem bude vytvořit dotaz, který bude zobrazovat data z obou těchto tabulek. Konkrétně "Název knihy" a "Cena" z tabulky "Knihy" a "Název žánru" z propojené tabulky "Žánry". Dotaz bude ve výsledku vypadat následovně:



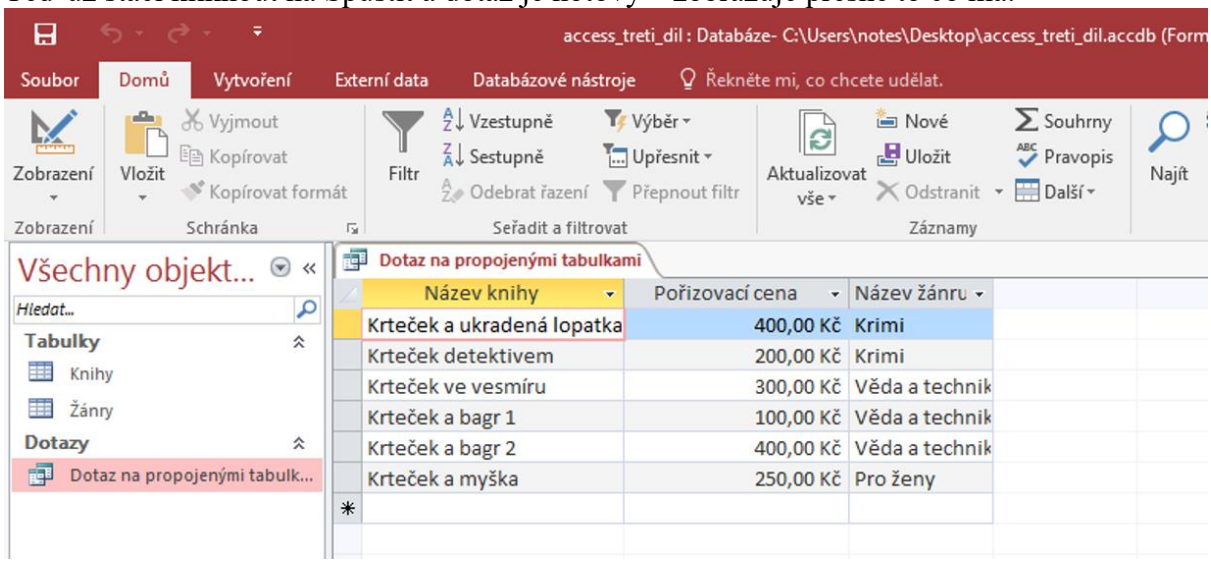
Ve Vytvoření klikneme na Návrh dotazu. Budeme pracovat s oběma tabulkami, proto na obě dvojkliknu. Ve schématu se ukázalo i dříve vytvořené propojení.



Zavřeme okno s tabulkami a klikneme na pole, která se mají ve výsledném dotazu objevit. Konkrétně je to „Název knihy“, „Pořizovací cena“ a „Název žánru“. Pole se tím naskládají do spodní tabulky, která vlastně definuje dotaz.

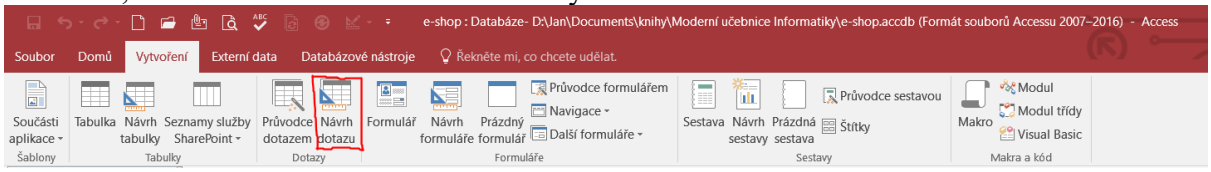


Teď už stačí kliknout na Spustit a dotaz je hotový – zobrazuje přesně to co má.

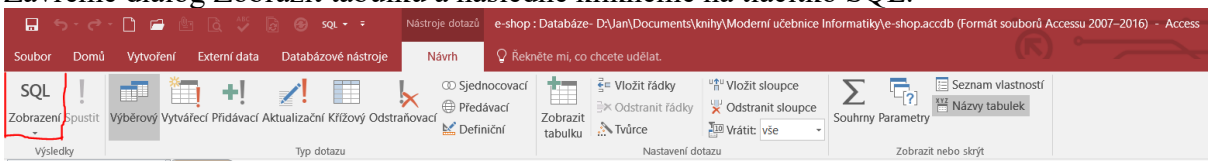


20.6.2.1. SQL dotaz

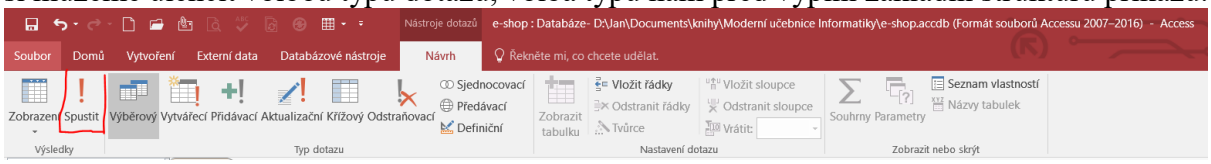
Pokud chceme psát dotaz pomocí jazyka SQL stejně, jako v případě práce s MS SQL serverem, tak tu možnost máme. Na kartě Vytvoření zvolíme Návrh dotazu.



Zavřeme dialog Zobrazit tabulku a následně klikneme na tlačítko SQL.



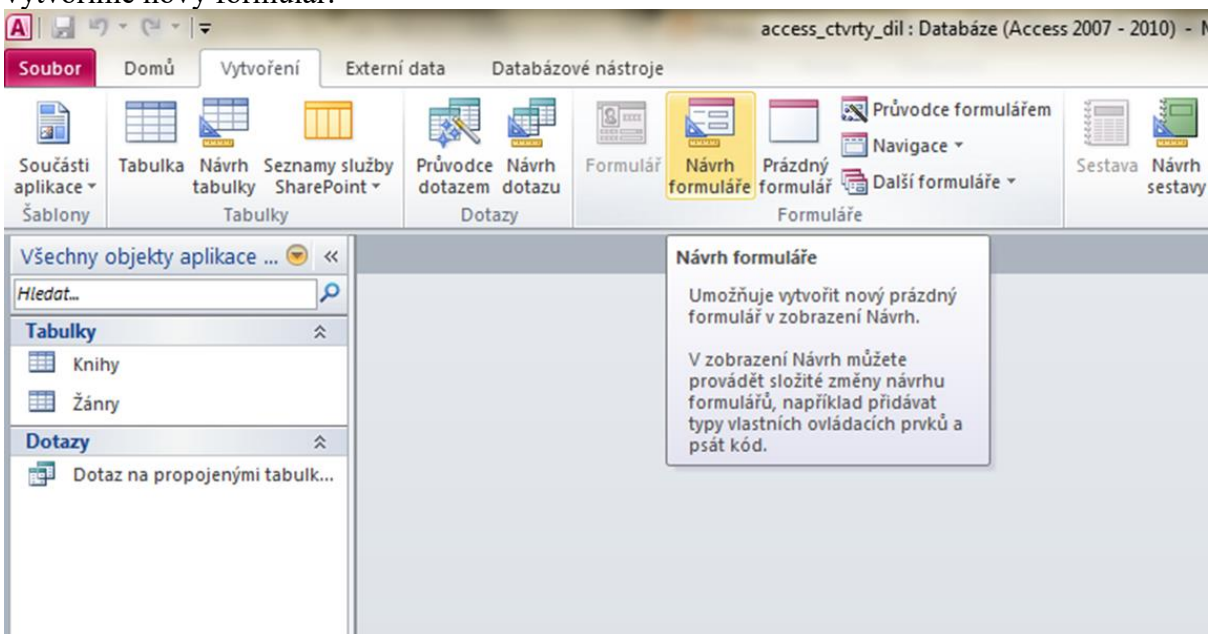
Nyní již můžeme psát vlastní SQL dotaz, ale na rozdíl od nástrojů k SQL serveru nemáme k dispozici nápovědu Intelisence. Kliknutím na červený vykřičník náš dotaz provedeme. Práci si můžeme ulehčit volbou typu dotazu, volba typu nám před vyplní základní strukturu příkazu.



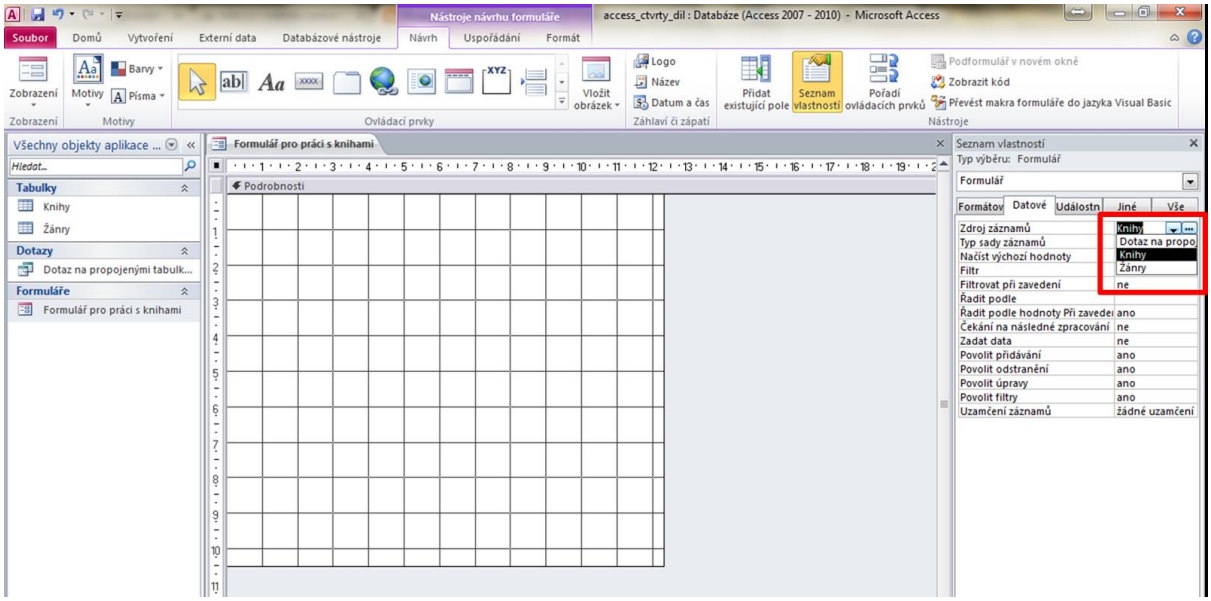
20.6.3. Formuláře

20.6.3.1. Tvorba

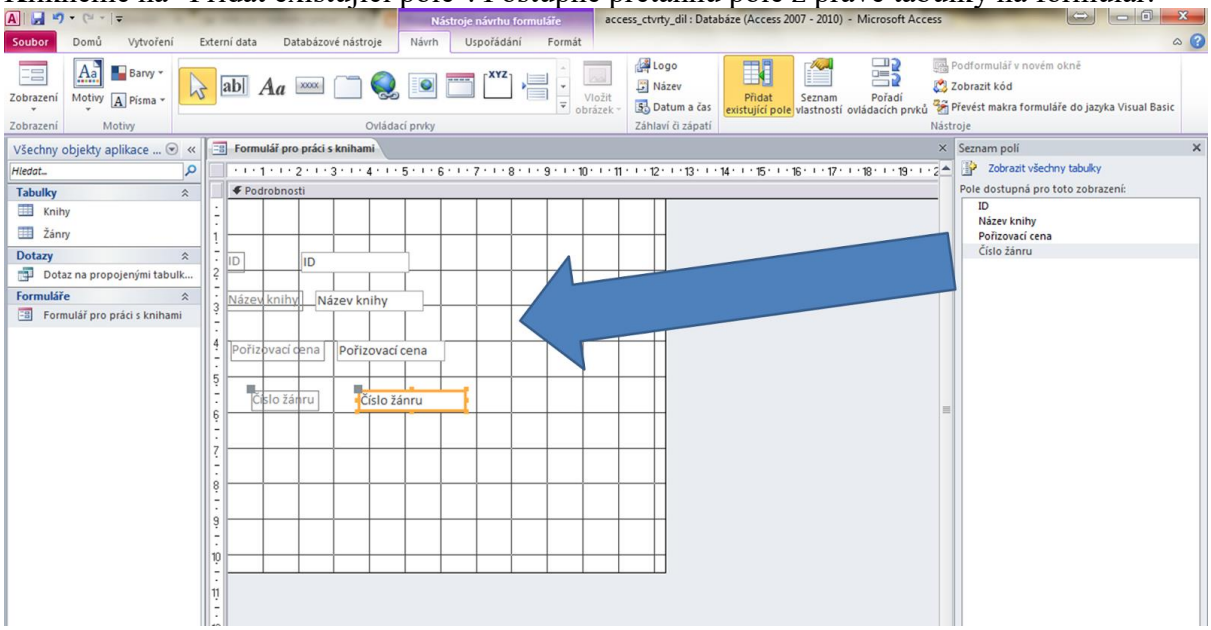
Máme tedy už jednoduchou databázi, ale zatím nemáme žádné uživatelské rozhraní, které by umožnilo snadno přidávat seznam knih nebo přidávat nové knihy. Uživatelské rozhraní vytvoříme pomocí takzvaného formuláře. Klikneme na Rozložení / Návrh formuláře. Tím vytvoříme nový formulář.



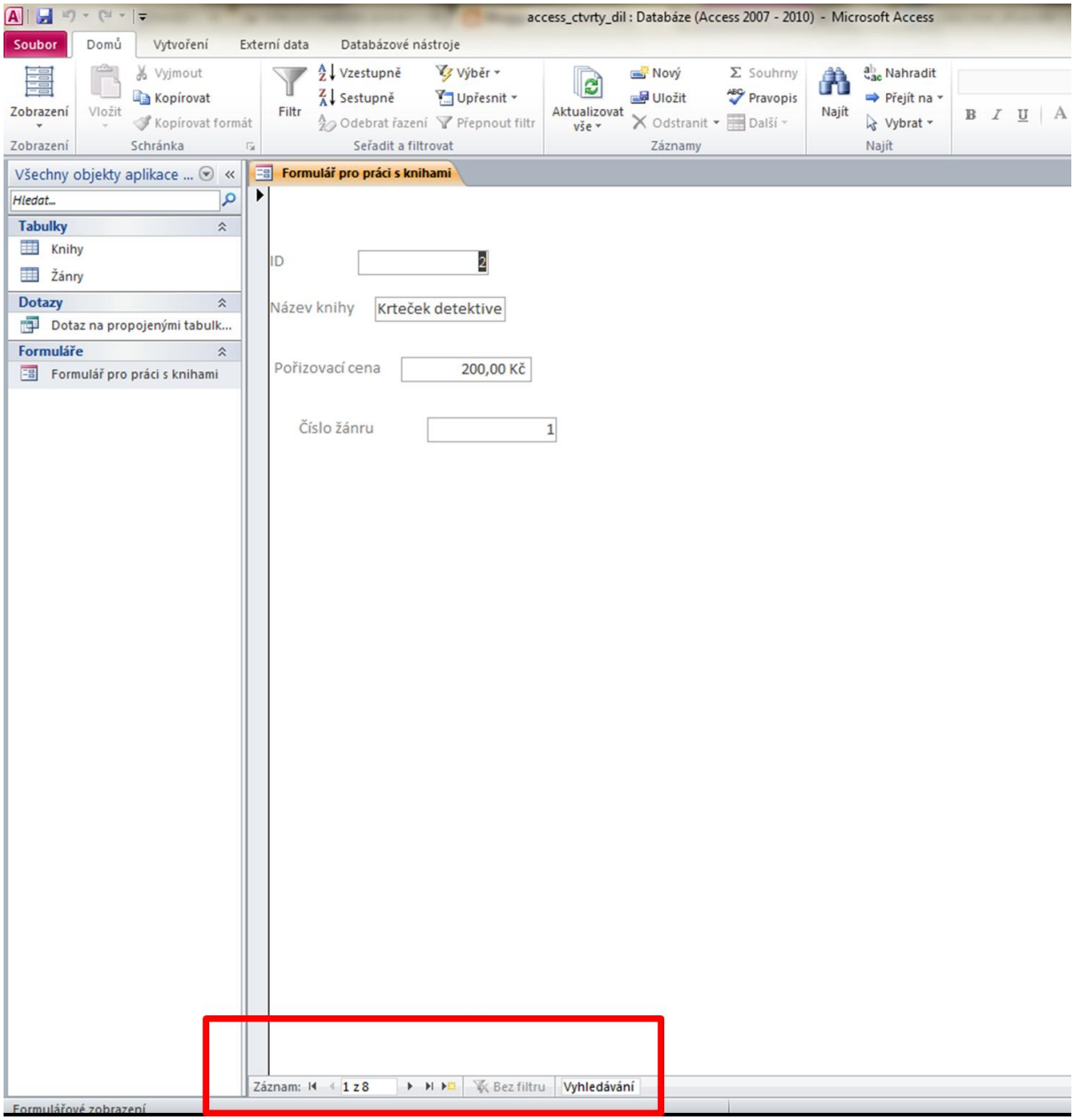
Musíme zvolit, která data se budou v tomto formuláře evidovat. Klikneme do formuláře pravým tlačítkem a vybereme Vlastnosti formuláře. V tabulce vpravo v Datové v položce Zdroj záznamů vybereme "Knihy". Tím říkám, že formulář bude prezentovat data z tabulky "Knihy".



Klikneme na "Přidat existující pole". Postupně přetáhnu pole z pravé tabulky na formulář.



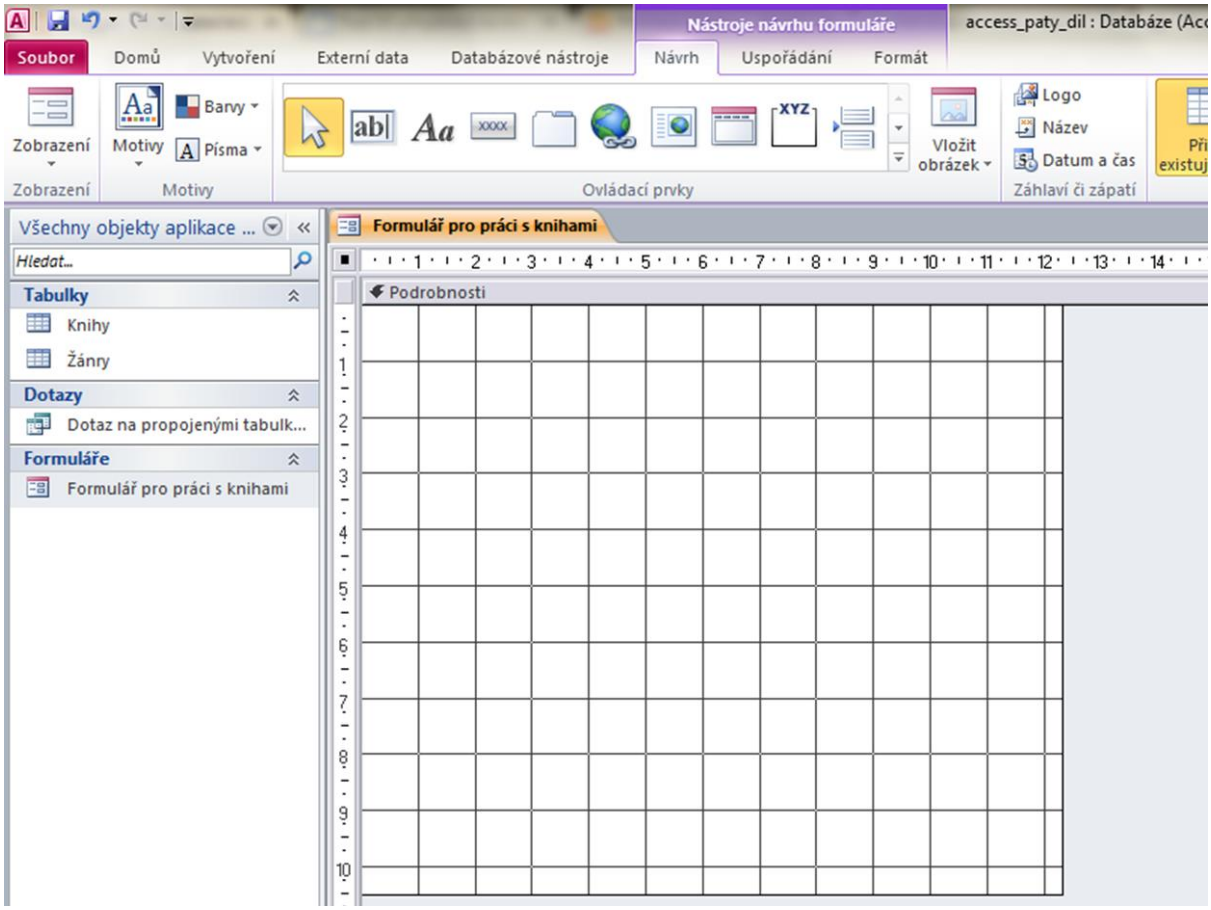
A to je všechno. Teď už jen změníme zobrazení formuláře z Návrhového na Formulářové. A máme hotový spuštěný formulář, ve kterém můžeme procházet knihy a přidávat nové – pomocí tlačítek dole.



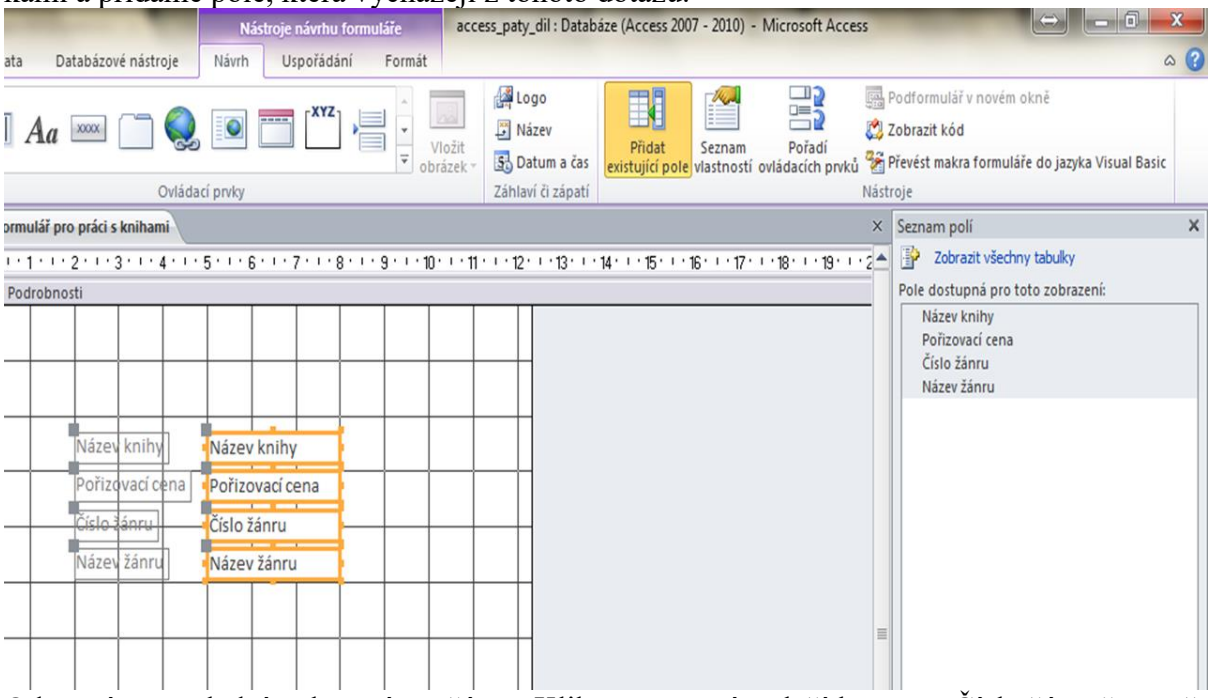
Na formuláři nám zatím jistě vadí, že musíme ručně zapisovat čísla žánrů – a přitom ani nevíme, co se, pod kterým číslem skrývá.

20.6.3.2. Dropdown box ve formuláři

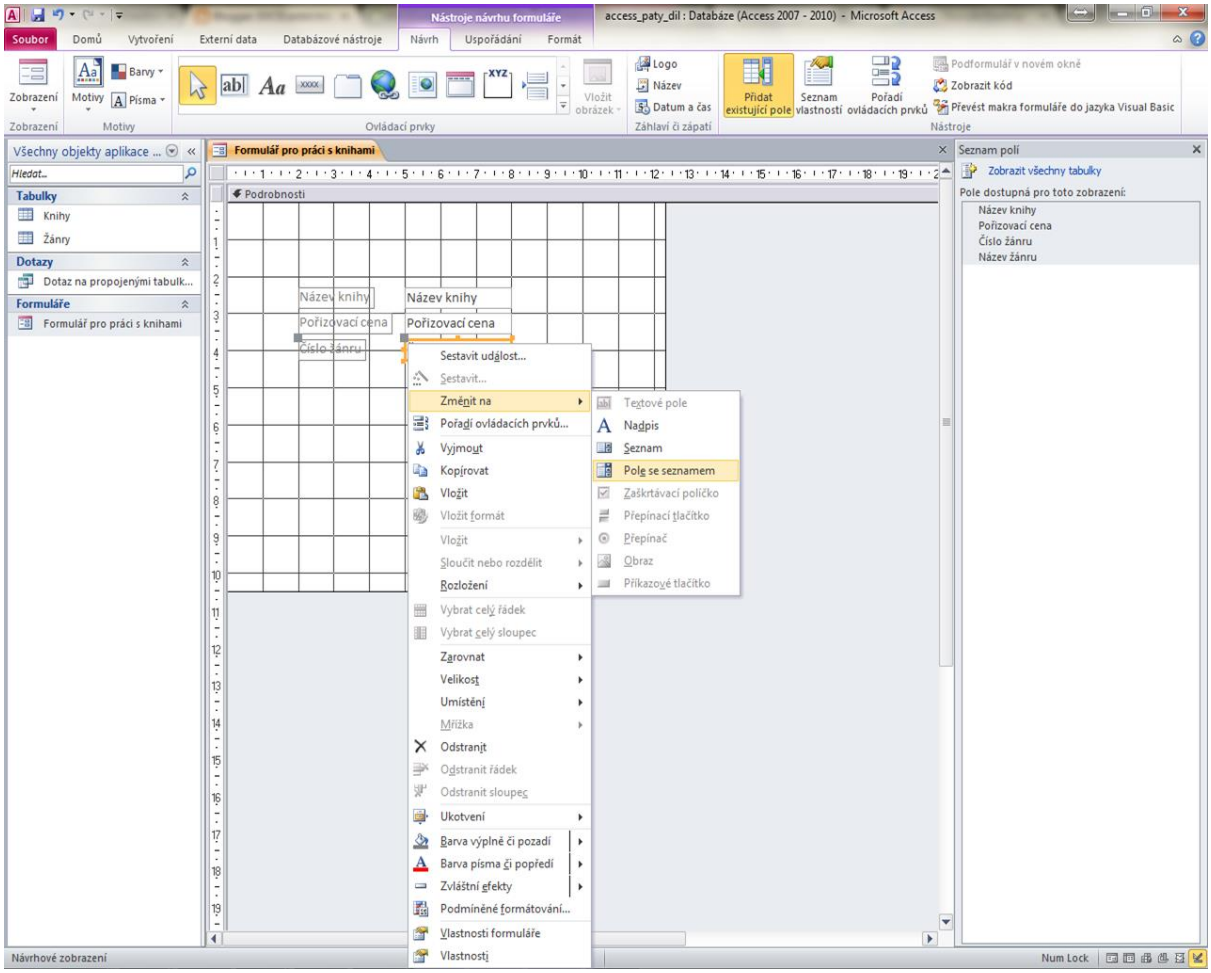
Na formuláři se nám nelíbí to, že jsou rozházená tlačítka, ale ještě jedna věc. Když chci přidat novou knihu nebo upravit stávající, nemohu rozumně vybrat žánr knihy. Teď musím ručně napsat číslo žánru. Dalece pohodlnější je vybrat si jednu z možností, a to z názvů žánrů, nikoliv z jejich čísel. To je problém, protože v tabulce s knihami jsou pouze čísla – názvy jsou v jiné tabulce, která je s tabulkou knih propojená v pohledu, který jsme už dříve vytvořili. Nejprve změníme data, na kterých je formulář postavený. Místo tabulky „Knihy“ budou zdrojová data brána z dotazu „Dotaz“ nad propojenými tabulkami. Otevřeme formulář a klikneme do Návrhového zobrazení. Smažeme pole, která máme od minule.



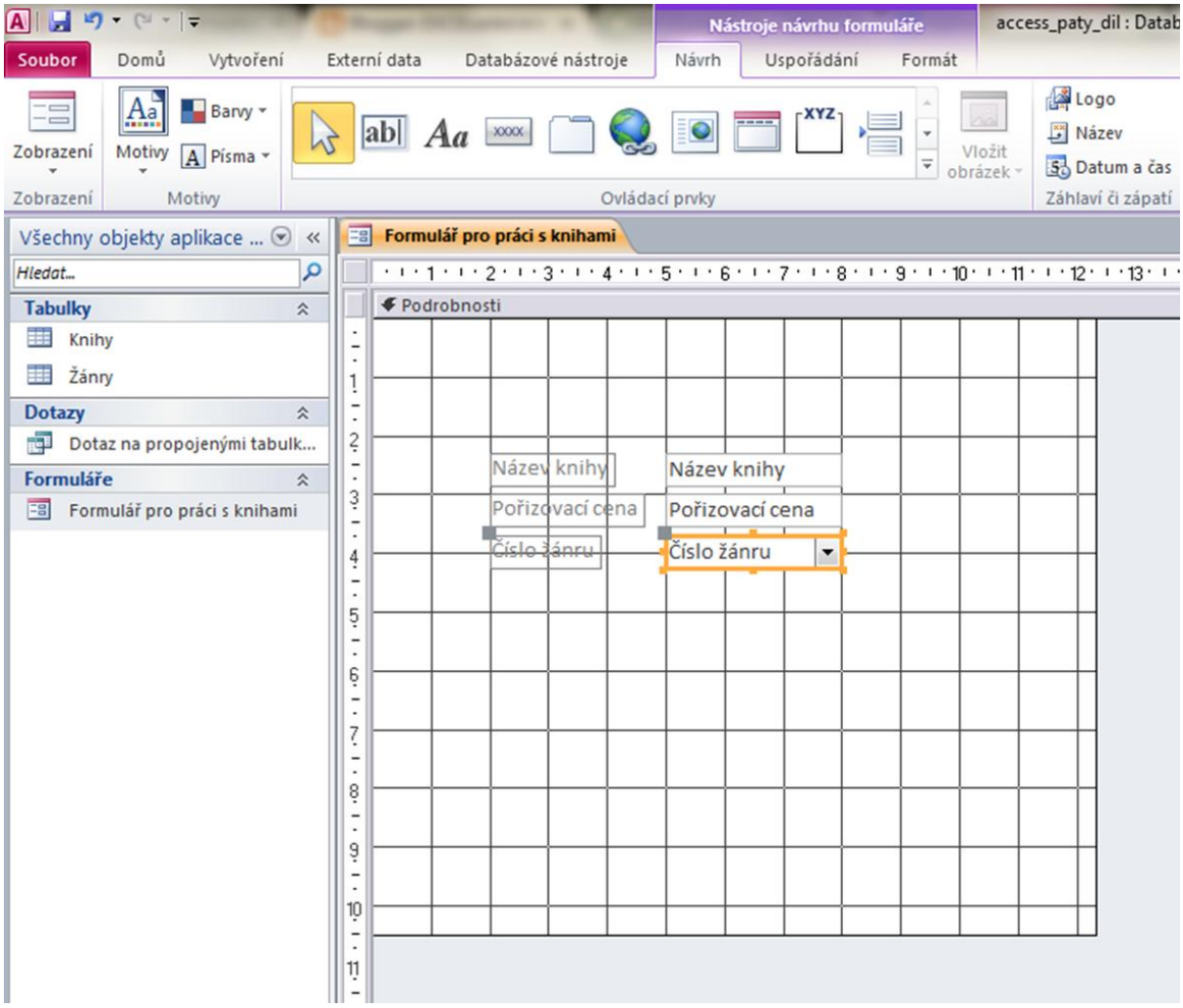
Klikneme pravým tlačítkem do formuláře a vybereme „Vlastnosti“ formuláře. V Datových vlastnostech formuláře vybereme jako zdroj záznamů dotaz „Dotaz“ nad propojenými tabulkami a přidáme pole, která vycházejí z tohoto dotazu.



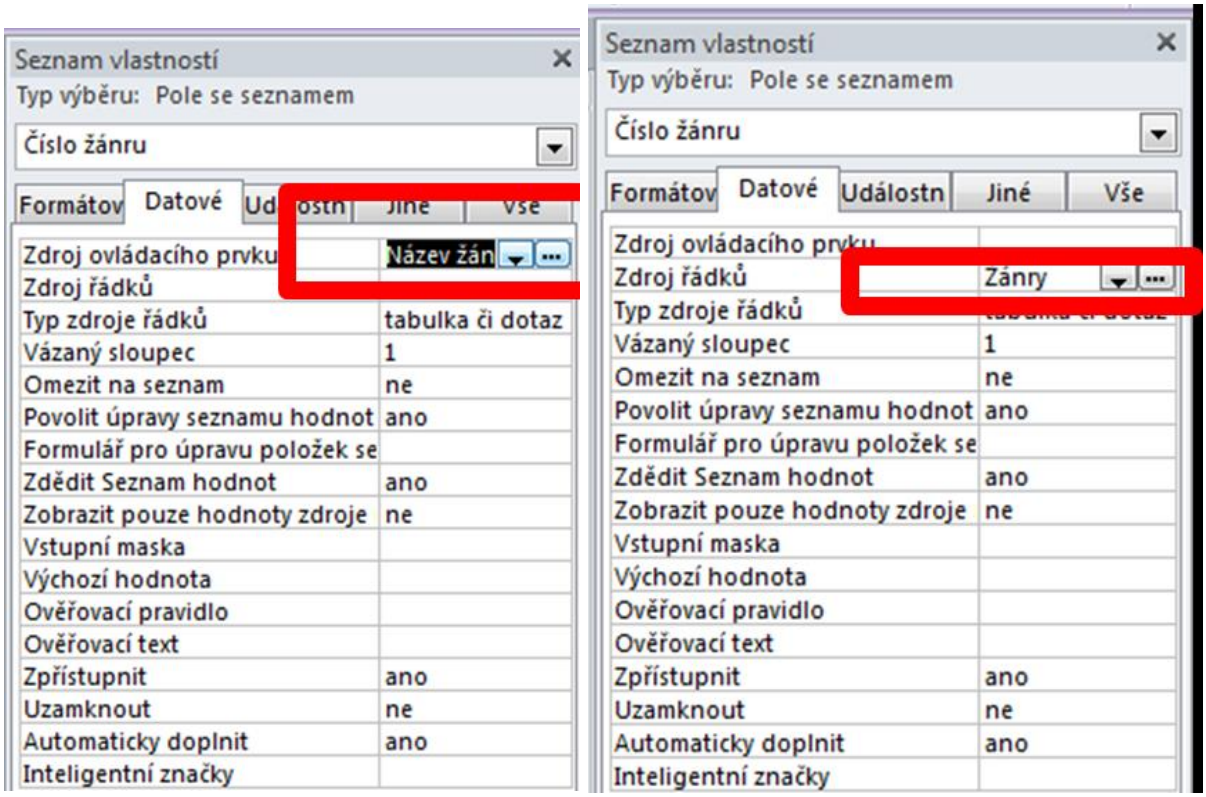
Odstraníme poslední pole – název žánru. Klikneme pravým tlačítkem na „Číslo žánru“ a změníme jej na Pole se seznamem:



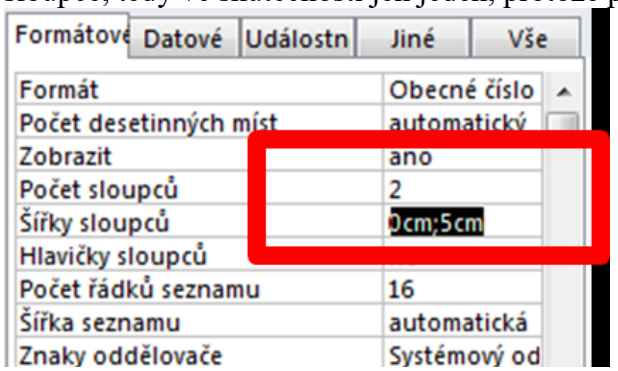
Takto vypadá výsledek:



Už se nám podařilo změnit textové políčko na výklopník, ale ten ještě nefunguje – ničím se neplní. V jeho Vlastnostech na kartě „Data“ smažeme Zdroj ovládacího prvku.



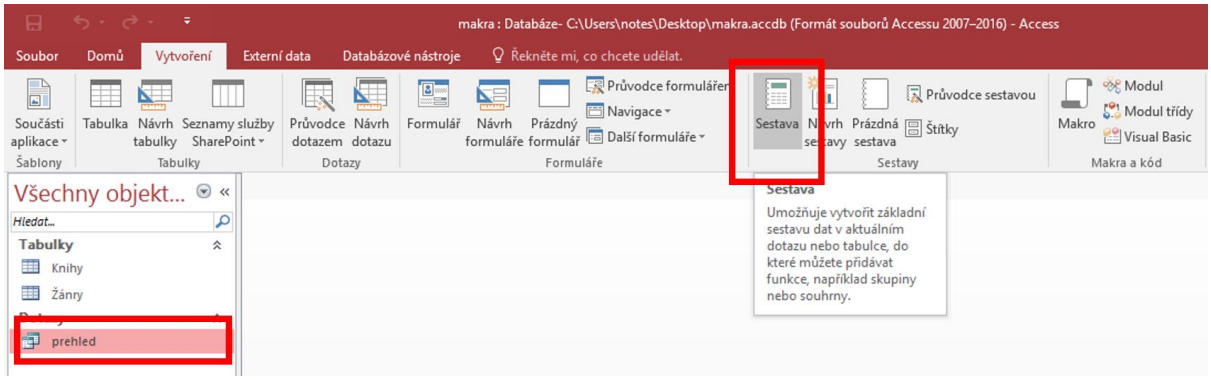
Teď už se výklopník plní – ale ne tak, jak chceme – zobrazuje se pouze první sloupec tabulky „Žánry“. Musíme ještě zařídit, aby se uživateli zobrazovaly názvy žánrů, které jsou ve druhém sloupci. To uděláme poněkud dřevním způsobem ve Formátových vlastnostech. Nastavíme, že první sloupec (to je ten s čísly žánrů) bude mít šířku nula, zatímco druhý sloupec (to je ten s názvy žánrů) bude mít šířku třeba pět centimetrů. Také nastavíme, že se budou zobrazovat dva sloupce, tedy ve skutečnosti jen jeden, protože první bude mít nulovou šířku.



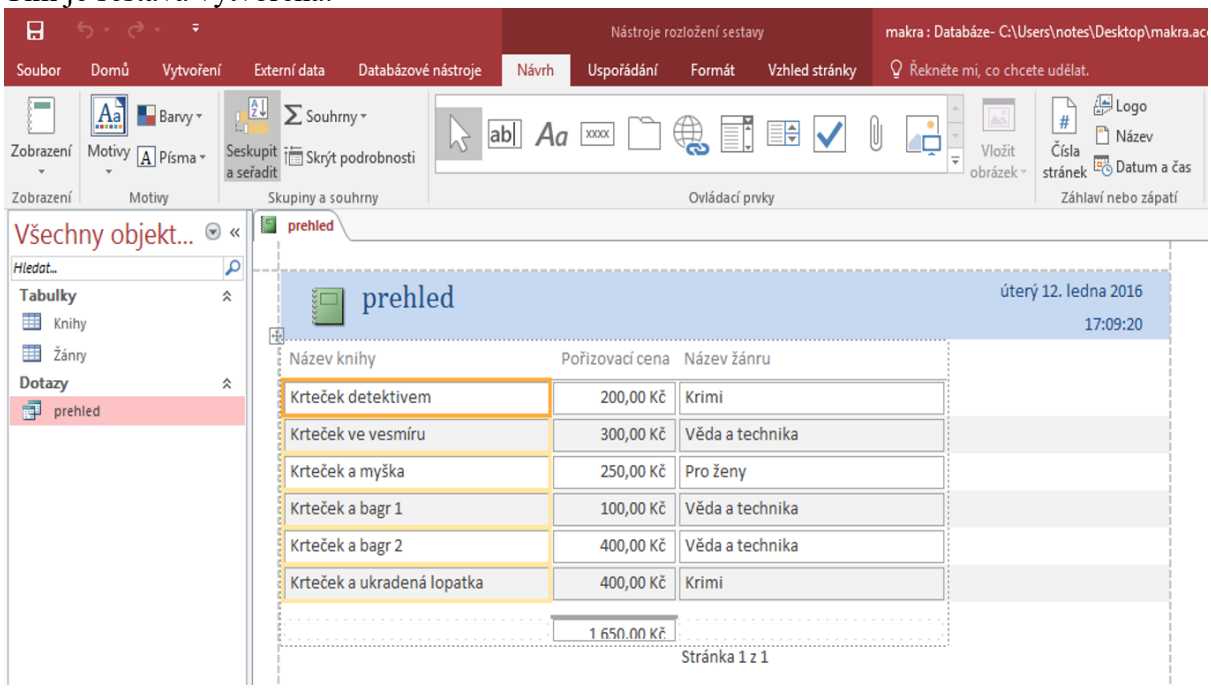
A to je vše. Formulář funguje, jak má. Je důležité si uvědomit, že ačkoliv uživatel vidí názvy žánrů, do příslušného pole v tabulce „Knihy“ se ukládá číslo žánru.

20.6.4. Vytvoření sestavy

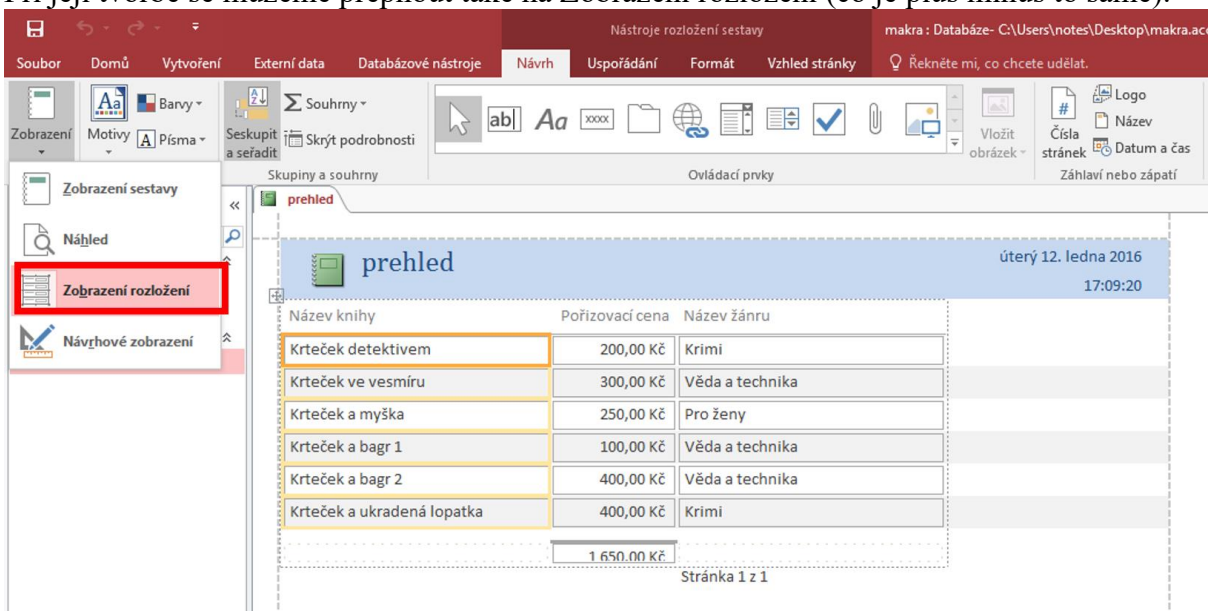
Sestavy je možné vytvářet nad dotazy nebo nad tabulkami. V Accessu klikneme myší na příslušnou tabulku nebo dotaz, a pak na Vytvoření / Sestava.



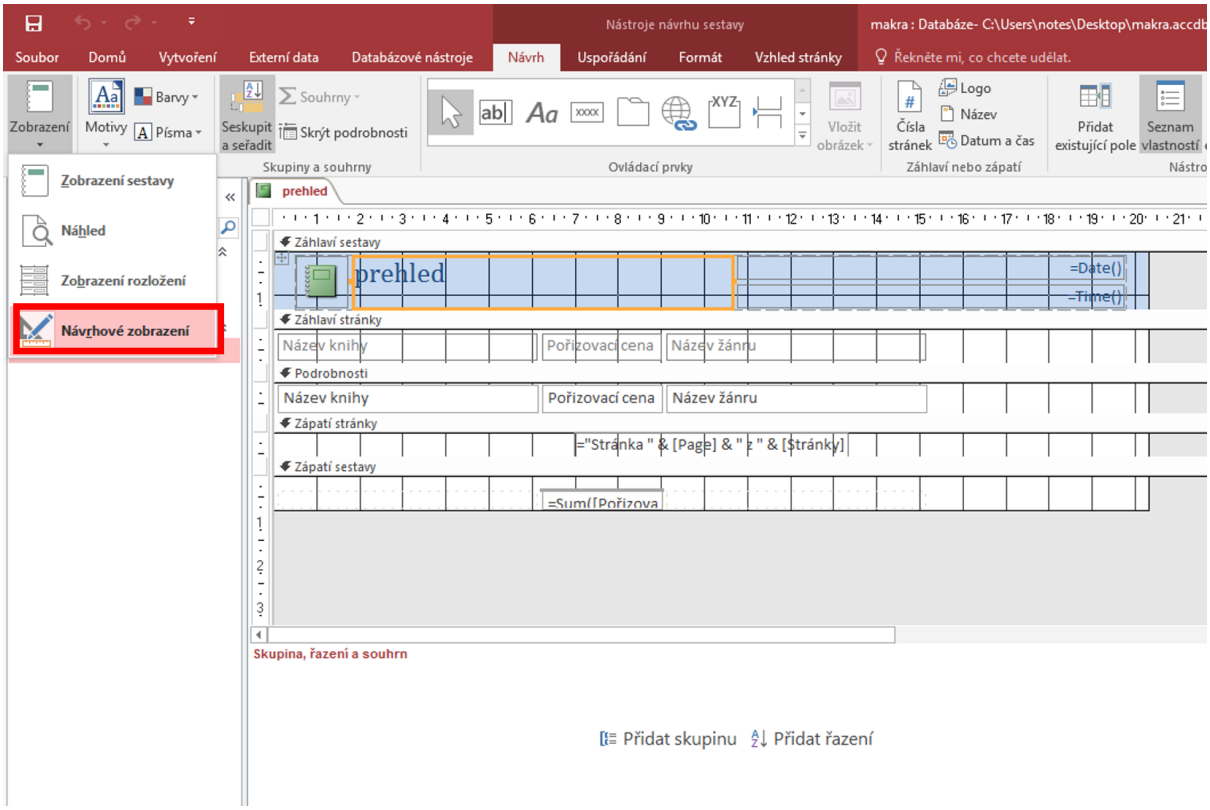
Tím je sestava vytvořená.



Při její tvorbě se můžeme přepnout také na Zobrazení rozložení (co je plus mínus to samé):

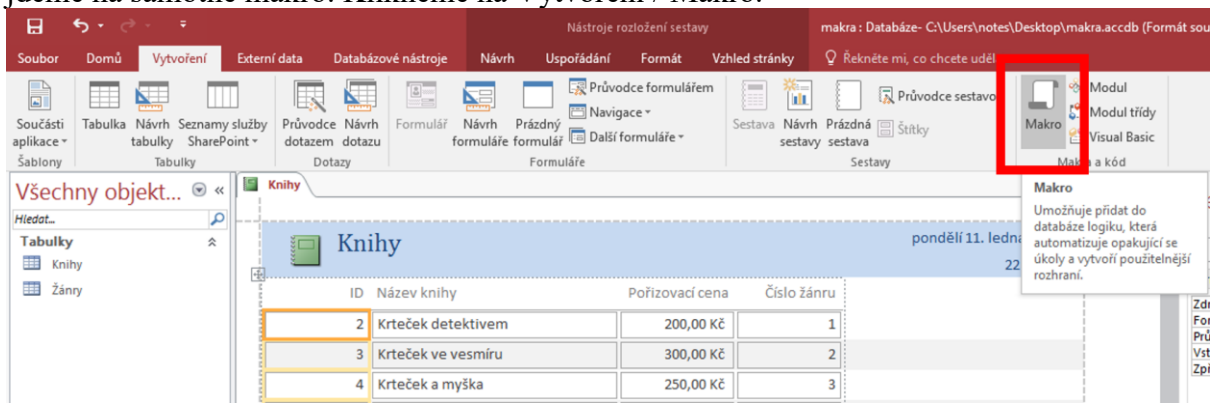


Nebo se můžeme přepnout na Návrhové zobrazení, které umožní přesné nastavení jednotlivých částí sestavy.

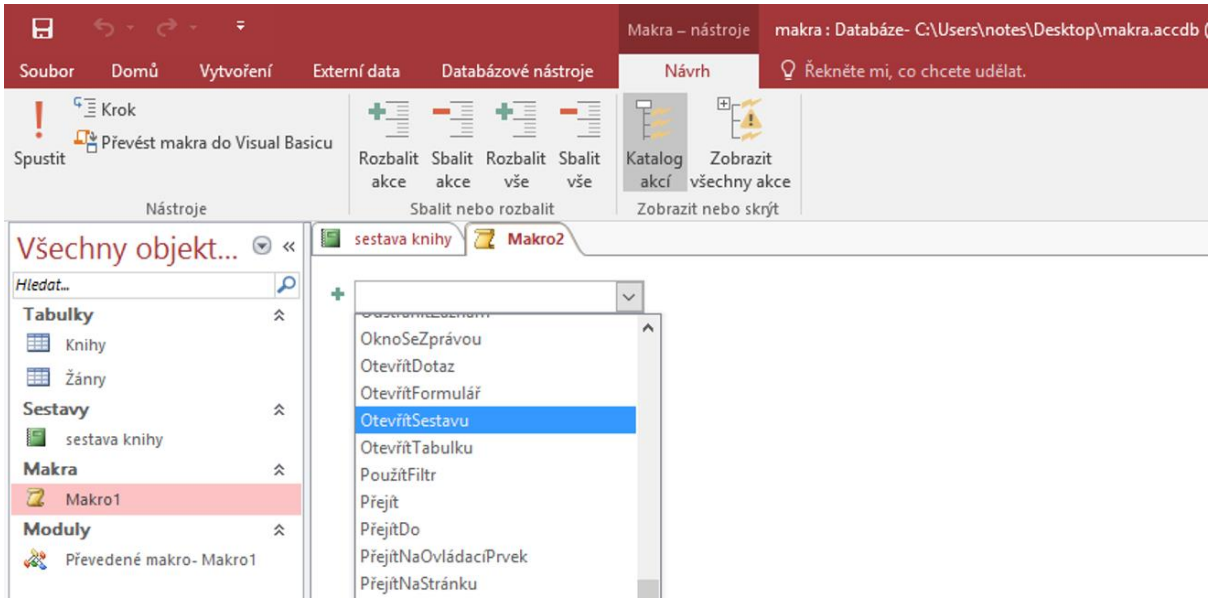


20.6.5. Vytvoření makra

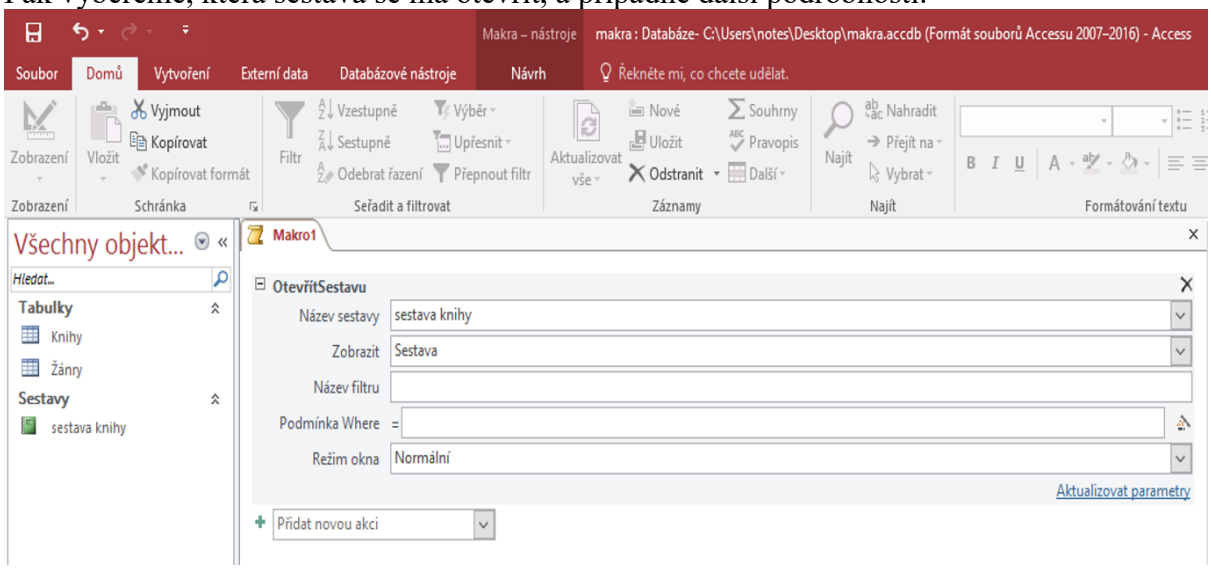
Makra v Accessu fungují trochu podobně, jako v Excelu, je v tom ale trochu terminologický zmatek. V Accessu jsou dva typy objektů použitelných pro programování – makra a moduly. Slovem "Makro" se v Accessu myslí sled přednastavených naklikaných kroků, čemuž se budeme teď věnovat. Moduly Visual Basic jsou pak v zásadě Accessovským protějškem maker v Excelu. V této ukázce vytvoříme jednoduché makro, které při stisknutí tlačítka otevře tiskovou sestavu a pak pípne. Nejprve vytvoříme sestavu, která se bude makrem otevírat. Teď jdeme na samotné makro. Klikneme na Vytvoření / Makro.



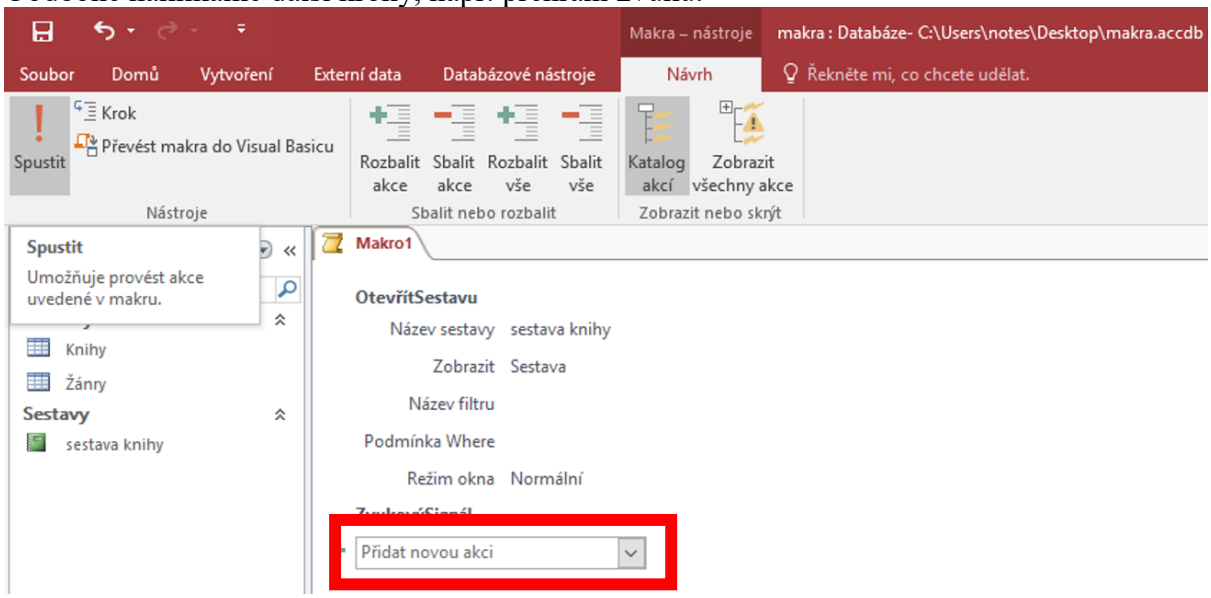
Otevřelo se okno, kde budeme makro vytvářet. Makro je v tomto kontextu sekvencí "naklikaných" kroků. Pro jeho vytvoření tedy nepotřebujeme psát VBA, prostě přidáváme další kroky. Prvním krokem má být otevření sestavy. Vybereme tedy OtevřítSestavu:



Pak vybereme, která sestava se má otevřít, a případně další podrobnosti.

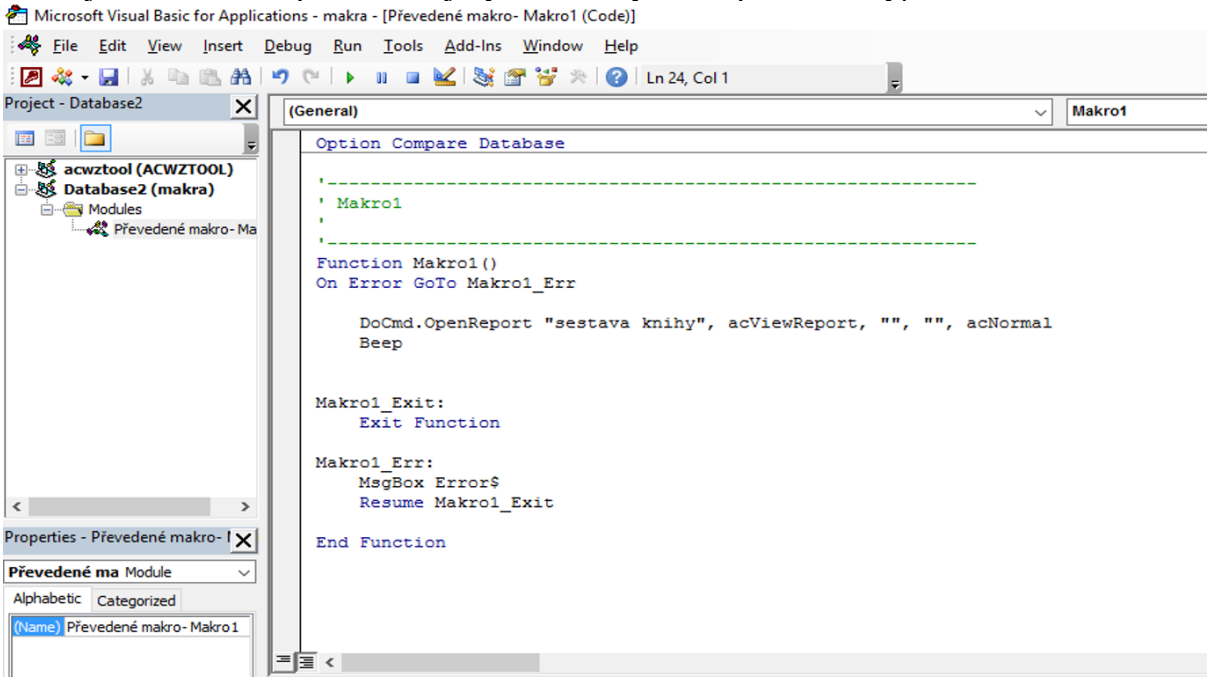


Obdobně naklikáme další kroky, např. přehrání zvuku.



Tlačítkem s vykřičníkem vlevo nahoře je možné makro spustit a vyzkoušet jeho fungování.

Vedle je také možnost převodu na jazyk VBA. Výsledek pak může vypadat i takto:



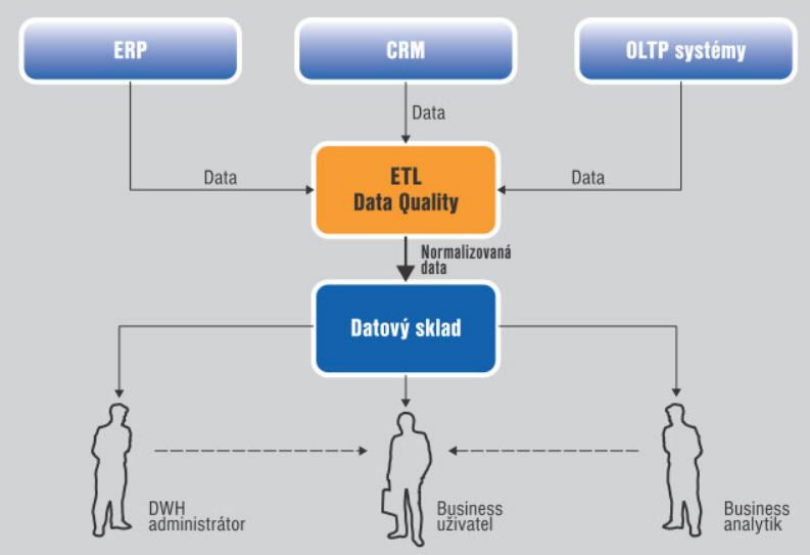
20.7. Big data

Pokud si člověk přečte deset prezentací dodavatelů technologií, přijde na zhruba patnáct různých definic. Každá z nich má podle očekávání tendenci podporovat produkty či služby toho daného dodavatele, nicméně většina vyhovuje definici: „Big data je termín aplikovaný na soubory dat, jejichž velikost je mimo schopnosti zachycovat, spravovat a zpracovávat data běžně používanými softwarovými nástroji v rozumném čase.“

Pojem „velikost“ dat je chápán nejen z hlediska objemu dat měřeného giga-, tera- či petabyty, ale i z hlediska rychlosti jejich tvorby a přenosu a z hlediska různorodosti jejich typů. Jako příklad je často citováno množství údajů o počasí, které získává každý den Národní úřad pro oceán a atmosféru (NOAA) nebo NASA. I komerční sektor má své premianty, jako například energetické, telekomunikační nebo farmaceutické společnosti shromažďující obrovská množství dat. Velké organizace čelí stálé potřebě udržovat rozsáhlé soubory strukturovaných i nestrukturovaných dat.

20.7.1. Klasický způsob využití dat z datového skladu

Až do nedávné doby bylo zpracování dat pro analytické účely poměrně statickou úlohou. Konkrétně podniky produkují zejména strukturovaná data ze stabilních datových modelů prostřednictvím podnikových aplikací, jako je CRM, ERP a finanční aplikace. Pomocí ETL nástrojů se data z těchto systémů přenáší do pracovní oblasti, kde se kontroluje kvalita údajů a provádí se jejich normalizace, finálně se pak ukládají do datového skladu. Tento proces obvykle běží v pravidelném cyklu – zpravidla na denní nebo týdenní bázi. Správci datového skladu pak vytvářejí reporty, které běží nad daty uloženými ve skladu. Datoví analytici používají analytické nástroje pro provádění složitých výpočtů nad daty ze skladu, anebo z důvodu omezení velikosti častěji nad vzorky dat z data martu. Běžným uživatelům je většinou povoleno použití BI nástrojů pro základní vizualizaci dat a omezené analytické výpočty. Objemy dat z tradičních datových skladů zřídka překračují několik terabytů.



20.7.2. Změna podstaty big data

Nástup webu, mobilních zařízení a dalších technologií zapříčinil zásadní změnu charakteru dat a způsobu jejich využití. Již nejsou centralizovaná, vysoce strukturovaná a snadno zvládnutelná, ale více než dříve jsou volně strukturovaná (pokud mají vůbec nějakou strukturu), vysoce distribuovaná a mají vzrůstající objem. Často se v této souvislosti hovoří o trojrozměrnosti velikosti a růstu dat (zkráceně také jako 3V):

- objem (volume) – množství dat vznikajících v rámci provozu firem roste exponenciálně každý rok
- typ (variety) – různorodost typů dat vzrůstá, například nestrukturované textové soubory, semi-strukturovaná data (XML), data o geografické poloze, data z logů,
- rychlost (velocity) – rychlost s jakou data vznikají a potřeba jejich analýzy v reálném čase vzrůstá díky pokračující digitalizaci většiny transakcí, mobilním zařízením a vzrůstajícímu počtu internetových uživatelů.

Big data mají odlišné vlastnosti, které je odlišují od „tradičních“ firemních dat. Tradiční datové sklady a nástroje pro správu dat nejsou připraveny na zpracování a analýzy velkých objemů dat ve velmi krátkém čase (někdy real-time) nebo nákladově efektivním způsobem. Proto je třeba hledat nové způsoby zpracování a analýzy velkých objemů dat.

20.7.3. Nové přístupy k analýze a zpracování big data

Jedním z takovýchto nových nástrojů je Hadoop. Hadoop je open source framework pro zpracování, ukládání a analýzu velkého množství distribuovaných, nestrukturovaných dat. Původně byl vytvořen ve společnosti Yahoo!, jako inspirace byla použita MapReduce, uživatelsky definovaná funkce vyvinutá společností Google pro indexování webu. Hadoop je stavěn pro zvládání petabytů a exabytů dat distribuovaných přes více uzlů současně.

MapReduce je výpočetní vrstva v rámci Hadoopu. Úlohy MapReduce přistupují k datům, která jsou distribuována na webu nebo v datových centrech, rozdělují je do více replikovaných dílů a jejich zpracování pošlou na jednotlivé uzly. Dotazy a další zpracování pak probíhá v každém uzlu paralelně. Výsledky jsou agregovány a ukládány do úložné vrstvy, jako například Hadoop Distributed File System (HDFS). Odtud jsou data načtena do jednoho z několika analytických prostředí pro analýzu. Ekosystém Hadoop se dále skládá z dalších vzájemně se doplňujících projektů. Mezi ně, kromě výše uvedených HDFS a MapReduce, patří NoSQL datová úložiště, jako Cassandra nebo HBase.

Hlavní výhodou Hadoopu je, že umožňuje analyzovat úplné datové soubory údajů, včetně nestrukturovaných a částečně strukturovaných dat, a to z hlediska nákladů i času efektivním způsobem. Mezi nevýhody Hadoopu patří částečná nezralost a hektický vývoj. Kromě toho, zavádění a řízení Hadoop clusterů a provádění pokročilé analýzy na velké objemy dat vyžaduje značné odborné znalosti. Pro firmy je takový model vesměs nepřijatelný, a proto v rámci ekosystému vznikla řada firem, které staví komerční řešení na bázi Hadoopu tak, aby se nasazení a správa technologie stala praktickou realitou tradičního enterprise odvětví.

Řešit problémy s big data znamená zasahovat do mnoha komponent IT architektury od hardwaru po optimalizaci vzorkování dat. Proto i trh s řešeními orientovanými na big data je velmi pestrý. Jednoduchý přehled obsahuje tabulka.

20.7.4. Vybavení pro práci s Big data

V rámci big data je velké zaměření na hardwarové konsolidace. Většina velkých dodavatelů nabízí integrovaná řešení včetně specializovaného hardwaru s důrazem na výkonnost při snížení celkových nákladů na správu a provoz. Na druhou stranu je zde i druhá skupina dodavatelů, kteří upřednostňují hardwarovou nezávislost, nebo přímo podporují běh svých systémů na komoditním hardwaru.

Množství dodavatelů řešení specializovaných na big data se rychle zvětšuje. Řada z dodavatelů vyvinula své vlastní Hadoop distribuce s různým stupněm úprav. Patří mezi ně jak řada firem vzniklých právě na základě potřeby řešit „velká data“, tak dnes i většina velkých hráčů na IT trhu.

V rámci data managementu hrají prim především NoSQL databáze jako prostředek pro obsluhu požadavků na čtení a zápis velkých objemů dat. Z pohledu integrace je zde patrná snaha o napojení big data technologií do stávajících nástrojů a zároveň jejich těsnou integraci se stávajícími technologiemi, především RDBMS.

Obecně platí, že čím větší vzorek dat, tím přesnější výsledek analýzy. To samozřejmě zvyšuje tlak na zvětšování objemu analyzovaných dat. Dodavatelé analytických nástrojů se snaží vylepšit své produkty, aby velké objemy dat zvládly a ulehčily uživatelům od vymýšlení a tvorby různých „náhradních řešení“. Trendem je využívat vlastní, vestavěné databáze jako součást analytických nástrojů, maximum dat nahrávat do operační paměti a pracovat s nimi tzv. in-memory, využívat nové principy uložení dat v databázi (sloupcově orientované databáze) či využívat masivně paralelní systémy. Velký rozvoj se očekává v následujících letech v oblasti zobrazování výsledků analýz (vizualizace dat). Bude se zvyšovat množství zobrazitelných bodů (hodnot), očekávat lze vylepšování animačních schopností analytických nástrojů.

20.7.5. Big data znamenají budoucí příležitost

Více než osmdesát procent všech dat v podniku má nestrukturovanou formu. Nejtěžší je nalézt v nich informace podstatné pro daný byznys. Informační pracovníci tráví dnes téměř čtvrtinu svého času právě vyhledáváním informací, přičemž doba získání těchto informací může mít vliv nejen na vnitřní produktivitu firmy, ale také například na spokojenost a loajalitu zákazníků. Big data přinášejí nový pohled i na samotné projekty datových skladů. Tradiční projekty budování datových skladů trvají i roky, od formulace zadání až po provedení samotných změn na základě výsledků analýz mnohdy uplyne dlouhá doba, což může způsobit značné finanční ztráty. Při použití big data prostředků lze tyto projekty zásadně urychlit a současně dospět k přesnějším výsledkům. Rychlost dosažení přínosů (time-to-value) bude klíčovým ukazatelem úspěšnosti těchto projektů. To bude vyžadovat změnu i na straně dodavatelů, kteří budou akceptovat krátké, intenzivní projekty.

21. Prostředky identifikace uživatele

Uživatele je často potřeba identifikovat s vekou mírou jistoty, protože dnes řada systémů obsahuje velmi citlivá data, například internetové bankovnictví, proto jsem se rozhodl do našeho výkladu zařadit toto téma. Dnes již i mobilní telefony obsahují prostředky pro vysokou míru identifikace majitele, protože telefony dnes o nás ví velké množství informací. Telefony dnes mají poměrně bežně čtečky otisků prstů, nebo realsence kameru pro Windows Hallow. Identifikace/Autentizace v osobním kontaktu spoléhá na biometrii – opakovaní rozpoznání tváře, nebo otisku prstu. Identifikace/Autentizaci využíváme pro ověření osoby pro přístup k aktivum, tedy informacím uloženým v systému. Identifikace je určení identity, následná autentizace je ověření identity. Autorizace je odvození oprávnění ze známé ověřené identity. V praxi dochází ke splývání uvedených pojmů dochází z objektivních důvodů ve chvíli, kdy v závislosti na kontextu může jeden pojem splynout s druhým. Identifikaci a autentizaci provádíme z důvodu možnosti provedení autorizace. Důvěrnost vyžaduje autentizaci, protože informace lze vydávat pouze oprávněným subjektům. Integrita vyžaduje autentizaci a lze provádět pouze povolené změny aktiva. Zodpovědnost vyžaduje autentizaci, za každý přístup k aktivu musí být zodpovědný konkrétní subjekt. Dostupnost je zpravidla ohrožena porušením integrity nebo přetížením systému.

21.1. Identifikace

Je proces určení identity zájmové osoby (systému), jde o sadu vlastností vázaných k určitému subjektu, na základě, kterých subjekt jedná či s ním jednají ostatní subjekty. Identita je různá dle kontextu, smysl ji dává paměť, kontext a stopa historie. Identita je většinou reprezentována jednoznačným identifikátorem, ovšem chybná identifikace může mít fatální následky. Volba identifikátoru:

- Identifikátor musí být jednoznačný v rámci kontextu
- Původ identifikátoru: přirozený (jméno, přezdívka), nebo umělý (identifikační číslo)
- Vypovídající hodnota: významový (nese dostatečnou informaci), nebo bezvýznamový (nenese jinou informaci, nežli unikátnost)

Jedna osoba může mít více identit v závislosti na kontextu. Propojování jednotlivých atributů identity může být žádoucí (vyšetřování zločinu) nebo nežádoucí (poškození soukromí). Online svět generuje množství různých pohledů a virtuálních identit pro jednu fyzickou osobní identitu. Vědomá identifikace nastává ve chvíli, kdy uživatel vědomě sdělujete svou identitu ve formě identifikátoru, naproti tomu nevědomá identifikace nastává ve chvíli, kdy je uživatel identifikován bez toho, aby vědomě sdělil svůj identifikátor (poznání člověka na ulici). Nevědomá identifikace probíhá i v online světě (automatické rozpoznání po otevření webové stránky).

21.2. Autentizace

Je proces potvrzení identity subjektu na základě důkazů. Máme 3 základní způsoby provedení:

- Znalost – něco vím
- Něco mám – vlastnictví
- Něčím jsem – rysy osoby (biometrie)

21.2.1. Autentizace znalostí

Znalost, která je známa jen osobě s danou identitou – heslo, postup apod. Při ověření subjekt prokazuje znalost hesla. Většina autentizací je založena na znalosti hesla k danému

identifikátoru. Autentizaci heslem vždy předchází identifikace identifikátorem, příklad známe všichni, typickým příkladem je uživatelské jméno a heslo. U hesel je riziko vytvoření kopie (zcizení) bez možnosti detekce. Při provádění autentizace je z pravidla nutnost odhalit tajemství, máme možnost využít tzv. zero knowledge postupy. Zero knowledge postupy jsou založeny na matematických postupech, kdy se neposílá přímo heslo, ale výsledek funkce spočítané z hesla a výzvy, server s originálem hesla provede stejný výpočet a porovná výsledek s tím, který mu přišel. Pro přenos hesel je nutné využívat šifrované protokoly. Program na sledování hesel: zed attack proxy project.

21.2.2. Autentizace vlastnictvím

Jde o autentizaci vlastnictvím konkrétního předmětu, příkladem může být čipová karta, mobilní telefon, OTP generátor, autorizační kalkulátor aj. Subjekt se prokáže vlastnictvím předmětu, ověření může být bez využití speciálního HW, nebo se specializovaným HW. Důkazy jsou založeny na kriptografii. Nejběžnější jsou PKI v podobě SW certifikátu nebo HW implementace. Velmi časté jsou také statické OTP, které pracují principiálně obdobně jako hra lodě. Dynamické generátory OTP jsou typicky mobilní telefon (SMS kódy) a tokeny, jde o ověřovací SMS kódy, které chodí při potvrzování transakcí internetového bankovnictví. V bankách a firmách pracujících s citlivými údaji se požívají autentizační kalkulátory (potvrzování transakcí v rámci bank), EMV CAP/DPA a kombinace (karta s klávesnicí/displejem apod.) - karta + pin.

Fyzický předmět s sebou nese ryziko scizení, ale toto scizení je snadno detekovatelné. Není možné vytvořit kopii, s vyjímkou SW PKI. Bez kombinace s autentizací znalostí snadno zneužitelné při zcizení, protože již není žádná ochrana. Nevýhodou je rovněž nutnost nošení dalšího předmětu, což není pro uživatele zrovna pohodlné. Z pohledu administrátora je další nevýhodou nutnost někdy připojení specializovaného HW do systému.

21.2.3. Autentizace založená na subjektu

Spoléhá na biometrické charakteristiky (otisk prstu, obraz duhovky, hlasová analýza…). Pracuje na principu srovnání vzorku získaného v reálném čase se vzorkem uloženým. Není schopna rozhodovat se 100% úspěšností, vždy se nastavuje určitá hranice, kdy je uživatel přijat a kdy ne, vždy s sebou nese určité procento chybných přijetí, či odmítnutí. Je založeno na statistických metodách. Výhodou je, že uživatel má vše potřebné vždy u sebe a je zde velmi malé riziko nezištného zcizení. Záskání kopije je obtížné, ale ne nemožné, ale vždy je nutné využívat specializovaný HW. Nevýhodou je nekompatibilita a špatná porovnatelnost různých implementací.

21.2.4. Vícefaktorová autentizace

Snižuje komfort uživatele, ale zvyšuje bezpečnost systému. Typické kombinace:

- znalost + vlastnictví – čipovka s pinem
- vlastnost + vlastnictví – čipovka s biometrií

21.2.5. Autentizace systému

Systém, kam vkládáme své údaje musí být důvěryhodný. Fyzická kontrola systému (můj počítač, firemní počítač) je jedním z bezpečnostních opatření, kterými můžeme zvýšit bezpečnost svých údajů. Internet: ověřená adresa, známá stránka, správný protokol. Systém by měl předložit certifikát serveru pomocí HTTPS vydaný důvěryhodnou certifikační autoritou.

21.2.6. Autentizace osob mimo informační systémy

Identifikační doklad (tzv. ID) slouží v zemi vydání k autentizaci držitele. Nesou základní údaje o osobě využívané většinou pro prokázání identity vůči státní moci. V ČR jde o

občanský průkaz (OP). Občanský průkaz je plastový, respektive papírový, doklad pro vizuální inspekci. Nese jednoznačný identifikátor (číslo OP, rodné číslo), dále pak základní údaje (jméno, místo narození, bydliště), fotografii a také tiskové a materiálové ochranné prvky pro prokázání pravosti dokladu. Dnešní podoba ID je nástroj vícefaktorové autentizace.

Cestovní pas slouží pro určení totožnosti při cestách do zahraničí. Nese základní údaje stejné jako ID a k tomu ještě zemi, která jej vystavila. Cestovní pasy jsou definované v ICAO (mezinárodní organizace civilního letectví) Doc 9303.

eID je jako běžné ID avšak s elektronickým rozšířením a je platné i v případě poškození čipové části. V EU je zajištěna standartizace prostřednictvím CEN a následně ETSI norem. Formát čipové karty je dán normou ISO 786. Běžně mají čipy průkazů kontaktní a bezkontaktní rozhraní a čip nese základní identifikace údaje v el. Formě a může nést biometrické údaje. Možné jsou i rozšířené funkce pro přímé užití na počítači a je možné poskytování určitých typů informací při zachování soukromí – potvrzení dosažení věku 18 let, ale utajuje datum narození. Poskytnutí citlivých údajů pouze terminálům s CAI.

ePassport je knížka vybavena bezkontaktním čipem. Čip nese osobní údaje – povinné, biometrickou fotografii obličeje – povinná, obrazy otisků prstů od roku 2009 v EU povinné a jsou přístupné pouze pověřeným terminálům. Obrazy otisků prstů může číst pouze stát, který pas vystavil a státy, které od něj dostali ertifikát, který je platný po dobu 6 měsíců. Dnes nedokáže ani česká celní správa vždy otisky číst. Čip také nese bezpečnostní informace (el. Podpis). Doklad je v současné době platný i bez funkční el. části.

elDAS je definováno nařízením vydaným v roce 2014. Stanovuje, že stát může zřídit elektronickou identifikaci a definuje, že elektronická identita je rovnocenná s fyzickou částí. V ČR vešla v platnost v červnu 2015 a nahrazuje zákon o elektronickém podpisu. Stát prostředky elektronické identifikace:

- Vydává
- Pověřuje vydáním
- Uznává vydané

Úroveň záruky:

- Nízká (jméno a heslo)
- Značná
- Vysoká (přísně chráněná data)

Nové občanské průkazy pro roce 2018 budou vyhovovat standardům elDAS na úrovni vysoké záruky. Úroveň se vyhodnocuje v průběhu celého životního cyklu prostředku, sleduje se úroveň zabezpečení předání prostředku. Prvek s vysokou úrovní záruky musí zaručit ochranu proti provedení autentizace po odposlechnutí údajů.

21.2.7. Útoky v prostředí internetu

Krádež identity souvisí s plnohodnotným černým trhem s identitami (platební karet, bankovní účty, účty platebních systémů, online hry, prodej bezpečnostních chyb). Plnohodnotný černý trh je také s nástroji na krádeže či zneužití identit. Cílem je finanční zisk, získání informací a neoprávněná modifikace informací.

Vývoj útoků na hesla neprodělal tolik změn, jako jiné oblasti informatiky. Útočník se vydává za autoritu, která má právo kontrolovat heslo, čímž oklame uživatele, který mu heslo poskytne. Další možností, jak hesla získat je odposlech klávesnice, nebo získání uložených hesel z počítače. Hesla je potřeba hashovat, neukládat v otevřeném tvaru, ačkoliv hashování je samotné je často slabou ochranou, ale stále lepší nežli nic. Mechanismy ochrany toho, co může být na systému spuštěno je dnes již nezbytnou ochranou. Obecně je snaha eliminovat hesla. Šikovná je webová stránka haveibeenpwned.com slouží ke zjištění, zda je náš účet mezi těmi, od kterých uteklo heslo na základě emailu.

Základní prostředky krádeže identity jsou:

- Slovníkový útok
- Útok hrubou silou (brue force)
- Odposlech hesel – man in the middle (odposlech na síti) a man in the browser (odchycení přímo v počítači)
- Sociální inženýrství – phishing, pharming
- Využití chybného návrhu aplikací – Cross Site Scripting (XSS)

Hesla na uložišti se mohou chránit především fyzickou ochhranou uložiště, použití šifrování (symetrická šifra klíčem systému) a použití jednosměrné hash funkce. Ukládá se SALT a HASH (SALT+Hodnota hessla), hodnota SALT eliminuje detekci stejného hesla porovnáním uložených HASH.

Hesla v OS nebo IS nejsou uložená v čitelné podobě. Systémy z rodiny UNIX využívají pro ulokládání soubory /etc/passwd – hash hesla veřejně dostupný a novější verze /etc/shadows – hash hesla dostupný jen privilegovanému ulivateli. OS Microsoft Windows využíá pro ukládání hesel SAM databázi – obsahuje hash (různě kvalitní dle verze systému) z hesla. Systém Windows 10 v edici EDU a Enterprise disponuje tzv. virtual secure mode, který mění uložení SAM databáze a také její strukturu.

Uhádnutí hesla je jednou z nejčastějších metod zlomení hesla. Jde aplikovat na systémy, kde není omezeno "zkoušení" hesel. Řešení tzv. lockout politikou – po X nesprávných pokusech se účet uzamkne.

Slovníkový útok je překvapivě účinný. Jde o zkoušení známých slov (jména dětí, filmů, zvířat, data narození, obvyklá hesla apod.). Brute force je proti předchozím metodám systematické procházení prostoru hesel. Brute force je, pokud chybí lockout politika, úspěšné prakticky vždy, ale je časově i výkonově náročnou metodou a také je takto vedený útok je hodně nápadný.

Politika silného hesla ztěžuje útok hrubou silou. Snižuje uživatelský komfort, protože je heslo hůře zapamatovatelné. Pro zapamatování silných hesel je vhodné používat mnemotechnických pomůcek. Zásady:

- Eliminovat slovníková hesla
- Maximalizovat velikost prostoru hesel
    - Minimální délka hesla 6
    - Vynucení použití širší rodiny znaků (a-z, A-Z, 0-9, spec. Znaky)

Nežádoucí je zmenšování prostoru hesel (eliminace zdánlivě zlepšujících pravidel – nesmí být více jak 2 číslice vedle sebe apod.), tato pravidla umožňují parametrizovat bruteforce útok tak, aby odhalil správné heslo rychleji. Dnes se již pomalu upouští se od politiky nutnosti výměny hesla po krátkých časových intervalech, protože často vede k nechráněnému ukládání hesla uživatelem. Důležité je doplnit heslo o další faktor, například OTP generátor.

Obrana proti kompromitaci přenášeného hesla:

- Šifrování – Heslo je přenášeno v šifrovaném tvaru (TLS)
- Challenge/Response protokoly – promocí hesla je potvrzována výzva a odpověď je možno konstruovat pouze se znalostí hesla
- Password Autehenticated key exchange protokoly – Do obecného algoritmu pro výměnu klíčů je zahrnuta hodtnota

XSS je útok, kdy kvůli chybě ve webové stránce útočník spustí svůj aktivní kód ve webovém prohlížeči uživatele s nadměrnými právy (prolomení "same origin" politiky). Útočník získává absolutní moc nad vším, co je zobrazeno i zadáno.

Man In The Browser využívá instalaci cizího SW. Při zadání transakce útočník přebírá kontrolu a modifikuje na pozadí příkaz transakce vedené vůči informačnímu systému, nejčastěji internetovému bankovnictví. Útočník má pod kontrolou i to, co se zobrazí a pamatuje si, co bylo zadáno.

Více faktorová autentizace kombinuje několik autentizačních metod. Většinou něco vím a něco mám (čipová karta a pin, OTP generátor a heslo). Výhodu je zvýšení bezpečnosti, ale snižuje komfort užovatele.

Vícekanálová autentizace eliminuje diskreditaci jednoho kanálu, např. PC napadené malwarem. Doplňuje druhý kanál komunikace, buď fyzický (SMS či poštovní zásilka), nebo kryptografický (autentizační/autorizační kalkulátor).

Praktické aspekty autentizačních metod, podívejme se na to s ohledem na skupiny uživatelů. Osoby se běžně autentizují v následujících podmínkách:

- Běžný uživatel IS
- Uživatelé jsou laici bez hlubších znalostí
- Špatná správa počítače, většině užovatelů stačí, že počítač nějak funguje
- Uživatelé nejsou neproškolení k řádnému používání IS a zprávě počítače

Pro firmy jsou pak tipické následující podmínky:

- Opět osoby, ale v jiném prostředí
- Proškolení uživatelé
- Profesionálně zpracovaný počítač
- Zodpovědnost za svěřená aktiva

Ekonomické požadavky na zavedení autentizačních metod dělíme do dvou kategorií. Investiční náklady jsou fixní (pořízení HW a SW licence) a na uživatele, protože počet užoovatelů souvisí s cenou licence. Druhou velkou skupinou jsou provozní náklady, někdy systém dražší na pořízení, může ve výsledku být levnější, díky výrazně nižším provozním nákladům. Při výběru řešení musíme zvážit možné metody úteku, které na systém hrozí.

21.2.8. Adaptivní autentizace

Požadovaná síla závisí na rizikovosti trasakce. Standardní úroveň pro retail: OTP autentizace, SW PKI pro autorizaci jednotlivých transakci. Zesílená autentizace: SMS, EMV CAP podpis, zpětné volání klientovi pro potrvrzrní transakce. Již dnes částečně zavedeno v mojebanka.cz, kde se využívá identifikace stanice, ze které do systému přistupujeme.

Možnosti implementace jsou komplexní FDS – pravidla a postupy tvořené dodavatelem FDS. Vhodné jsou implementace spíše pro backend, kde pokryje všechny transakce, nejen přímé bankovnictví.

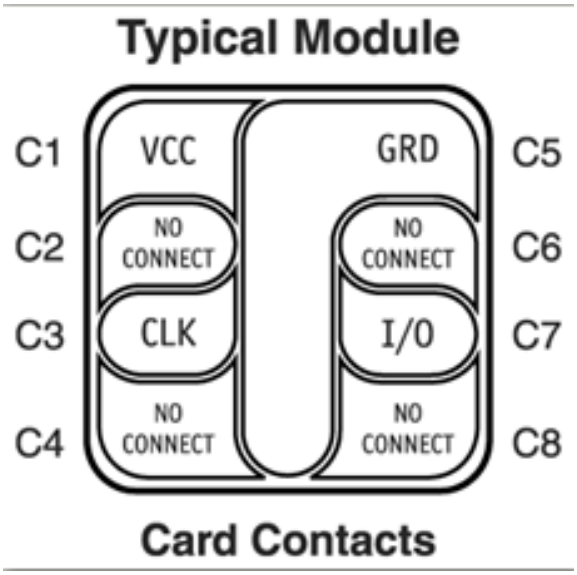
21.2.9. Čipové karty

Je karta vybavená čipem a komunikačním rozhraním, která je nezávislým prvkem mimo IS. Omezené možnosti interakce vedou ke zmenšení vektorů útoku a vyšší bezpečnost. Výhodou je mobilita totohot identifikačního prvku, což je umožněno vysokou mírou standardizace těchto prvků. Značná míra penetrace do různých systémů, příkladem může být SIM, platební karta, městské karty, InKarta apod. Základem karty je čip a podle něj dělíme karty do dvou velkých kategoorií, paměťové a procesorové karty.

Paměťová karta umožňuje pouze zápisy a čtení paměťových míst, jich příklady jsou Telefonní karty z konce minulého století.

Procesorová karta obsahuje jednočipový procesor s komunikačními obvody. Má oproti paměťové kartě pokročilou logiku a vyšší míru ochrany. Aby výpočty prováděné na kartě byly možné, je využívána HW akcelerace kryptografie a biometrie. Příklady jsou: SIM, EMV platební karta, InKarta.

Komunikační rozhraní, kterými jsou vybaveny procesoroové karty jsou dvou typů, kontaktní a bezkontaktní. Kontaktní rozhraní je většinou založeno na sériové komunikaci, kdy je čip uvnitř karty napájen skrz komunikační rozhraní, které je ukázáno na následujícím obrázku.

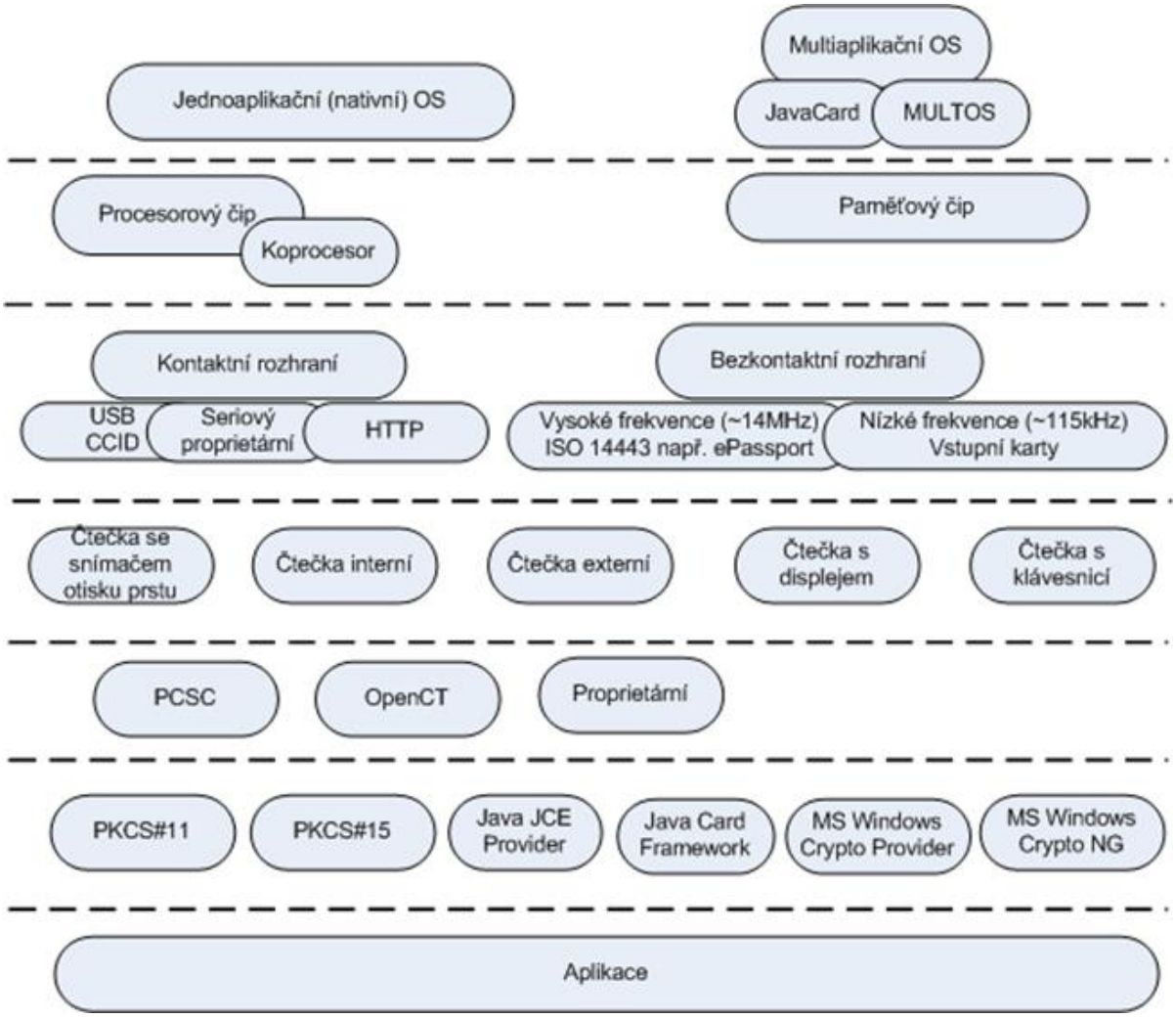


Bezkontaktní rozhraní je obecně ne zcela správně shrnována pod kategorii RFID. Napájení čipu je zajištěno anténou pomocí elektromagnetické indukce. Zde bohužel nalezneme velké množství různých standardů a proprietárních řešení, které nejsou vždy vzájemně kompatibilní. Určité informace je možné číst ze všech typů karet pomocí NFC a open SW.

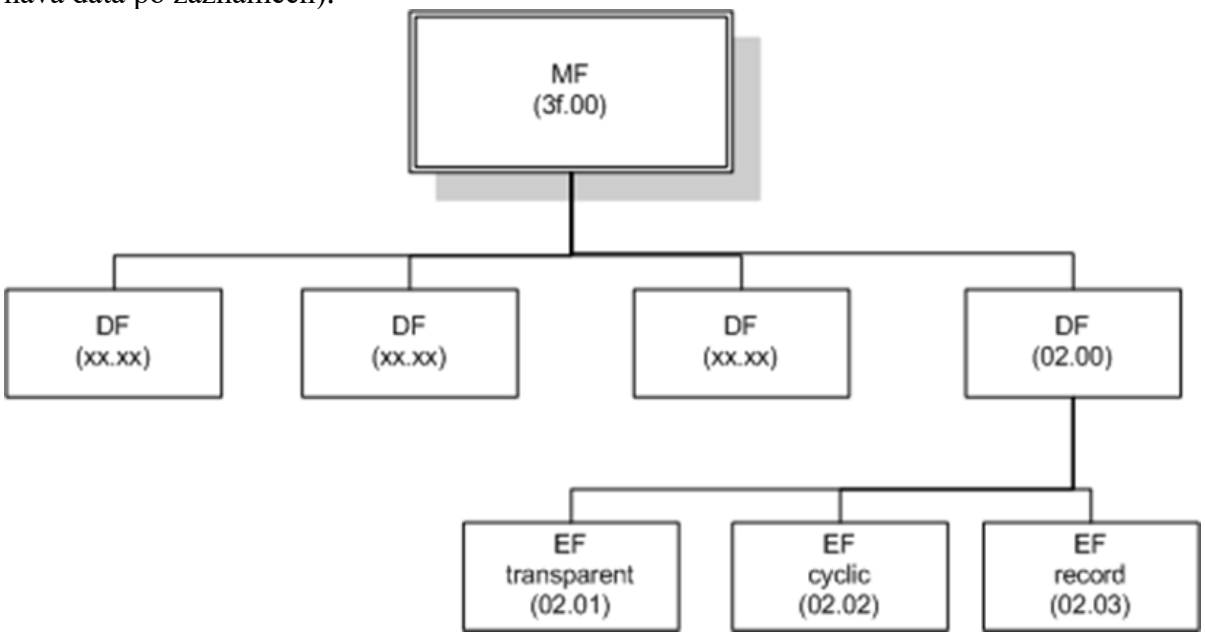
Kombinované rozhraní máme dvojího typu, duální – jeden čip s dvojím rozhraním, nebo hybridní – dvojice čipů na jedné kartě.

Standardy rodiny ISO 7816 specifikují kartu od fyzických parametrů po syntaxi a sémantiku komunikačních protokolů a chování karty. Specifikuje rozměry a mechanickou odolnost karty, definuje polohu a rozměry kontaktů i elektrické charakteristiky a základní protokoly. Z SW stránky je definována struktura souborů a základní příkazy (např.: ověř pin), stejně jako registrační mechanismus pro identifikátory aplikací a datové struktury. Pro komunikaci s kartou se používá jazyk SCQL – card query language, jehož příkazy souvisí mimo jiné s bezpečností a správou životního cyklu karty. Některé karty umožňují biometrickou identifikaci majitele a disponují USB rozhraní (T=USB). Moderní karty umí zpracovat příkazy pro správu aplikací ve víceaplikačním prostředí, ač se to většinou nepoužívá. Nejbezpečnější je, aby jedna karta měla pouze jednu aplikaci.

Nejvíce se prosadily javakarty, kteréíé se programují v Javě, karta má nahraný aplet. Nejčastější varianta karty: JavaCard na kartě s Procesorovým čipem s koprocesorem a komunikuje na rozhraní USB CCID. Je doporučeno používat čtečku s klávesnicí (platební terminály, kopírky) a v OS je karta reprezentována rozhraním PCSC. Pro kryptografii se ustanovily dva základní přístupy: PKCS#11 a MS Windows Crypto Provider, případně MS Windows Crypto NG, který je běžný v novějších systémech s jádrem NT. Následující obrázek ukazuje, jak vypadají obě běžně požívané skupiny karet od fyzické vrstvy, přes komunikační protokol až po aplikaci.



Uložení dat na kartě je vždy ve stromové struktuře. Každý soubor či adresář má označení 2B číslem, které slouží jako „název“. Kořenový adresář značíme MF a podřízený adresář je DF a slouží vždy pro jednu aplikaci. Datové záznamy (soubory) jsou označeny EF. Datové soubory jsou 3 typů: transparentní (binární kód), cyclic (cyklický buffer) a record (zaznamenává data po záznamech).



Postup práce s kartou:

- Reset karty
- Nastavení globální bezpečnostní kontext
- Nastavit DF (slect DF)
- Nastavit EF (select EF)
- Nastavit per EF/DF bezpečnostní kontext
- Provést operaci s aktuální aplikaci

Bezpečnost:

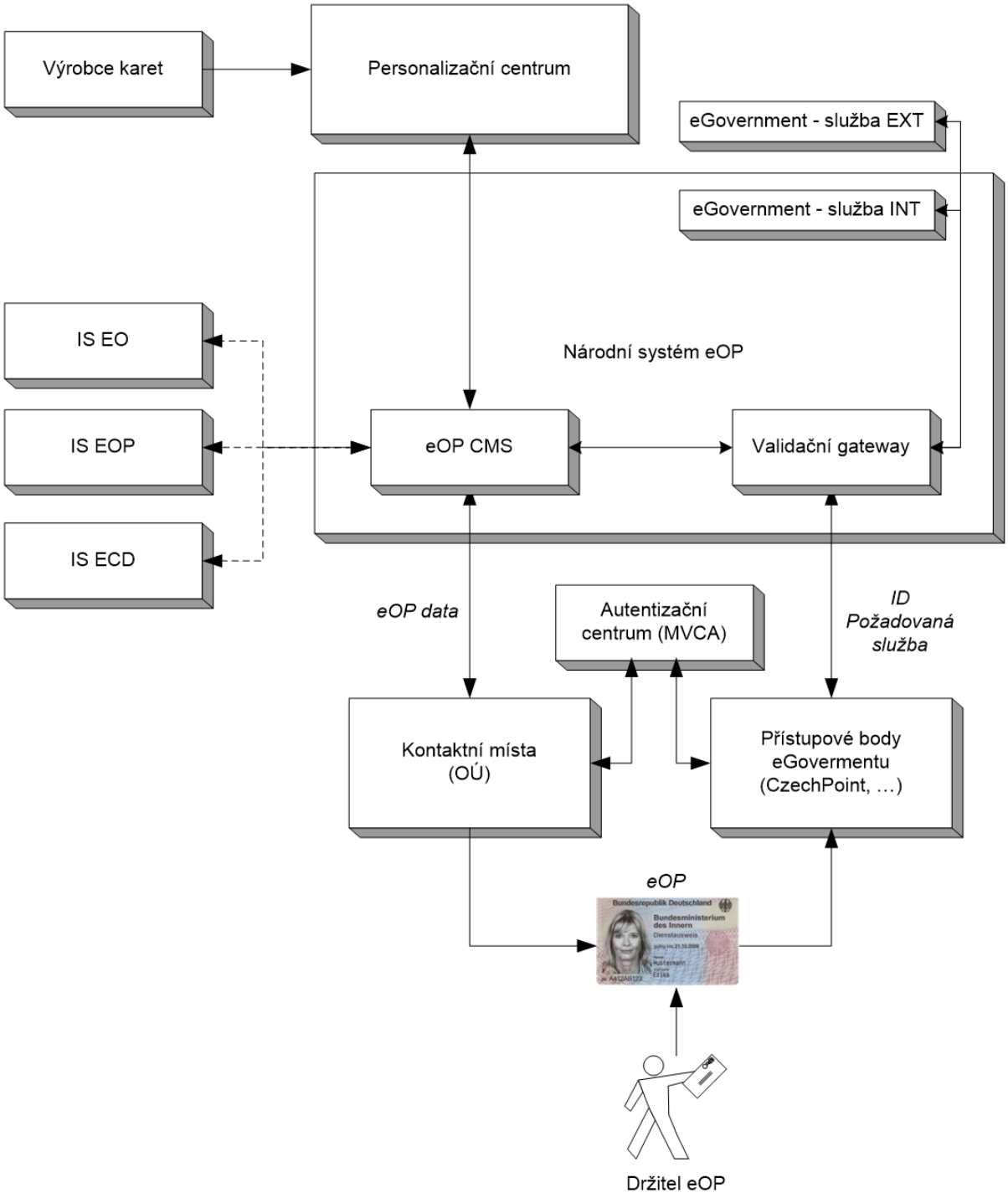
- Ověření pinu
- Ověření klíče (symetrického/asymetrického)
- Bezpečností správy jsou zasílány mezi kartou a domovským serverem. Mezi terminálem a kartou je navázán bezpečný tunel. Šifruje procházející zprávy a připojuje MAC k procházejícím zprávám. Umožňuje vytvořit bezpečný tunel mezi kartou a centrem skrze zcela nedůvěryhodné prostředí (např.: nabíjení e-peněženky).

Ověření původu karty využívá statické autentizace – interní datové struktury jsou el. podepsány. Možná je také dynamická autentizace, kdy karta generuje unikátní podpis (např. včetně výzvy z terminálu). Internal authenticate použije interní privátní/soukromý klíč k tvorbě důkazu o původu karty.

Applikacion protocol data unit (APDU) slouží pro komunikaci s kartou a skládá se z 5 až X B, naproti tou PC/SC standard definuje nízko úrovňový standard této komunikace. Využití karty na PC – SW:

- Middleware – SW modul, který je často zamčený
- Zpřístupňuje služby karty na vyšší úrovni
- Různě zaměřené middleware – kriptografie a eID
- Standardy: PKCS#11, PKCS#15, MS Crypto API Provider / NG
- PKCS#11 je dostupné v jazyce C
- PKCS#11 rozlišuje uživatele (běžný a administrátor), ověření pomocí PIN/hesla (možnost alternativní cesty mimo knihovnu)

Karta je jen malou součástí infrastruktury, která je mnohem složitějš a lehce popsána následujícím obrázkem pro elektronickou OP. Je nutné si uvědomit, že karta musí být začleněna do celého systému, který se skládá z Card management systém (CMS) - vydání, předání, stažení karty, čtecí zařízení (terminály) i SW, který pracuje s kartou.



21.2.10. OTP generátory

HW generátory jednorázových hesel, nebo SW aplikace se stejným účelem. Častým příkladem jsou SMS OPT a OATH, které je sdružení firem (http://www.openauthentication.org). OATH definuje standard pro jednotnou implementaci vícefaktorové autentizace. HOTP-MHAC-Based One Time Password je založeno na čítači a TOTP-Time-Based One Time Password je založeno na aktuálním čase (rozlišní 30 sekund).

HOTP-RFC4226 je založen na MHAC-Based One Time Password algorithm. Na základě sdíleného tajemství za využití HMAC algoritmu je generován kód. V tokenu i ve validačním modulu existuje sdílený klíč (pro každý token). Centrálně je generován kód a zaslán do mobilu. TOTP(KC) = Truncate(HMAC-SHA-1(K,C)), kdy čítač je samostatný pro každého

uživatele a nikdy se nesnižuje. Zasynchronizování pomocí rozsynchronizováni klienta a serveru o více jak 3 kódy se děje pomocí dvojice kódů od uživatele najednou.

TOTP-RFC6238 je založen na Time-Based One-Time Password Algorithm. Na základě sdíleného tajemnství za využití HMAC algoritmu a času je generován kód, který je platný po 30 sekund. V tokenu i ve validačním modulu existuje sdílený klíč (per token) a musí být synchronní čas. TOTP=HOTP(k,T), kde T=(Current Unix time - T0)/X. Při každé autentizaci dojde k synchronizaci času, kdy si server uloží rozdíl zadaného času a serverového času.

OCRA-RFC6287 stojí na OATH Challenge-Response Algorithm. Je založeno na HOTP, ale jako vstup se používají data a čítač.

Praktické aspekty používání OTP s sebou nesou dva hlavní pohledy, prvním je licencování, které se dělí na licencování tokenů a licencování validačního modulu, které se děje per uživatel. Druhým pohldem jsou processy, které se skládají ze zavedení a předání tokenu, správu tokenu a vyřazení tokenu.

Příkladem je Autorizační kalkulátor, který je nejméně pohodlný pro uživatele, ale je nejvíce zabezpečený. Uživatel zadává detaily požadavku do zařízení, které vrátí pvěřovací kód. Data jsou symetricky "podepsaná".

21.2.11. FIDO aliance

Aliance sdružuje všechny stakeholdety autentizace. Cílem aliance je autentizace bez hesel, založená na elektronickém podpisu eliptickými křivkami, jehož základem je asymetrická kriptografie. Privátní klíč je chráněný v certifikovaném autentikátoru, který splňuje požadavky pro udělení atestace. Pro každou službu (stránku) je generován samostatný klíč, kterým se uživatel přihlašuje ke službě, díky čemuž je zaručena unikátnost hesel uživatele k různým službám. Cílí na snadnost použití, ochrany soukromí a bezpečnost. Autentikátor se nejčastěji připojuje k počítači pomocí USB portu, ovšem jedno připojení autentikátoru provede jen jedno přihlášení ke službě. Autentikátor nepotřebuje instalovat žádné ovladače, v systému se tváří jako klávesnice. V případě přihlašování k mobilnímu zařízení se autentizuje člověk tím samým, jako k zařízení (otisk prstu, nebo Windows Hallo). Pokud na mobilním zařízení chybí biometrie, je možné použít heslo (pouze na certifikovaném zařízení). Certifikované zařízení má HW čip, který funguje jako TPM v počítačích. Mezi serverem autentizační služby a stránkou probíhají dvě akce, registruj a autentizuj se.

FIDO 1.0 definuje Universal 2nd Factor (U2F) a Universal Autehntication Factor (UAF). Novější verze standardu FIDO 2.0 definuje W3C Web Authentication Draft a sjednocení U2F, UAF pod jedno API, které je např. implantováno ve Windows 10 jako Windows Hallo.

21.2.12. Biometrie

Fido staví na tom, že UAF používá pro odemčení Biometrii, kterou se všeobecně rozumí měření fyzických charakteristik (statických a dynamických). Biometrická identifikace 1:N je vyhledání shody v N šablonách, které je výpočetně náročné a praxi se příliš nepoužívá. Biometrická verifikace 1:1 je porovnání šablony s jedním vzorkem, což je v praxi dnes již běžné. Biometrie zakládá své závěry na statistice a výsledkem je míra pravděpodobnosti shody.

Netechnická biometrie se používá od pravěku, např. jde o rozeznání známé tváře v místnosti či na chodníku. V historii se používal i otisk prstu (starověká Čína nebo Babylon). V průběhu 19. století byly stanoveny první formální postupy: bertillonáž a daktyloskopie. Základní vlastnosti ve vazbě na osobu jsou:

- Nelze zapomenout či ztratit
- Je těžké je falzifikovat
- Je prakticky nepřenositelná na jinou osobu

Rozlišujeme dva základní typy biometrie, **statickou** – oční duhovka (duhovka není statický objekt, je stále pulzující objekt a pokud není pozorován pohyb, jde o podvrh), oční sítnice, tvář, tvar vnějšího ucha, otisky prstů, geometrie prstů a ruky, topografie žil dlaně, rozměry, váha, DNA a **dynamická** – podpis tužkou, hlas…

Předpoklady použití charakteristiky pro identifikaci:

- Měřitelné
- Jedinečné
- Neměnné
- Technicky realizovatelné
- Automatizovatelné
- Musí být příjemné snímání pro člověka

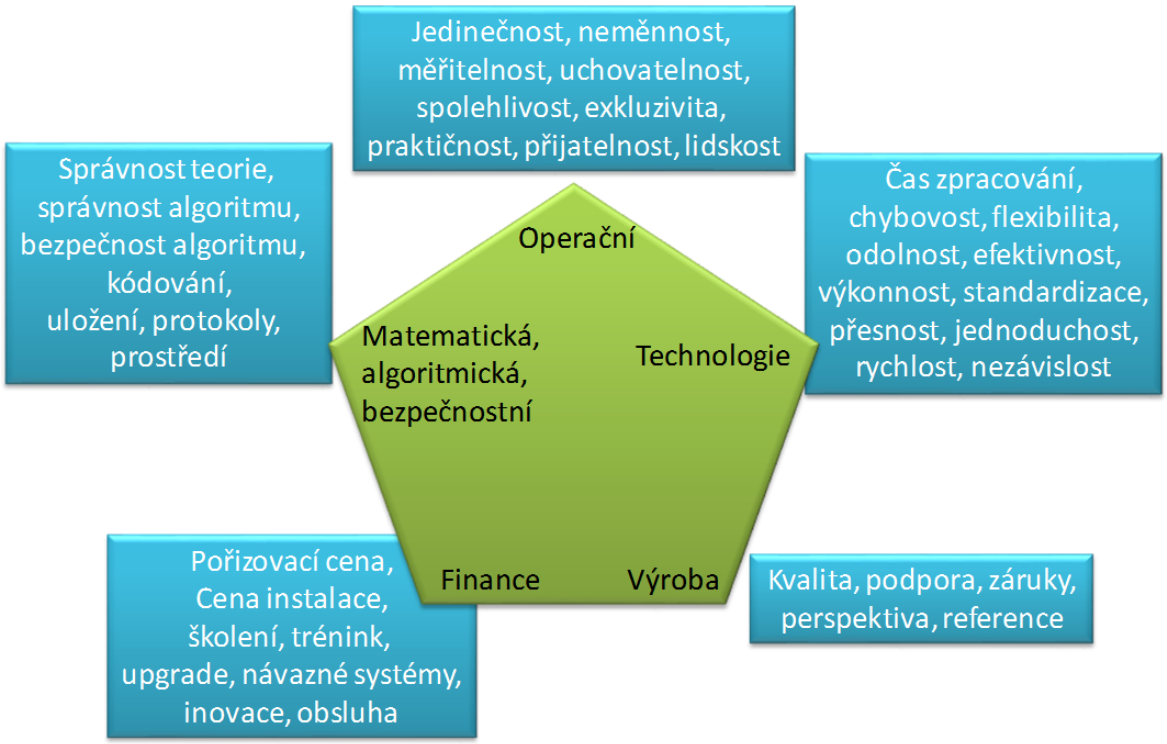
Klasifikace charakteristik dle použití:

- Bezpečnostně komerční – otisky rstů, duhovka/sítnice, tvář, hlas, podpis...
- Policejně soudní – otisky prstů, DNA, hlas, písmo (podpis)

Požadavky na charakteristiky dle jejich použití:

Charakteristika	Bezpečnostně komerční	Policejně soudní
Doba zpracování	Řádově sekundy	Nehraje roli
Regulace	Smluvní	Zákonná
Oblast použití	Široká	Úzce zaměřená
Dostupnost	Obecná	Pouze pro státní složky
Cena	Rozhoduje, únosná	Nehraje takovou roli, zpravidla vysoká

Kritéria pro nasazení biometrické technologie:



Operační kritéria obsahují vlastnosti, které souvisí přímo s biometrickými charakteristikami a jejich propojení s konkrétním uživatelem. Od biometrické charakteristiky očekáváme:

- Jedinečnost – schopnost jednoznačně identifikovat/verifikovat
- Neměnnost – měřené prvky se nemění po dobu potřebnou (produktivní věk)
- Měřitelnost – spolehlivé měření
- Uchovatelnost – schopnost uchovat naměřené (od čipu po db)
- Exkluzivita – sama metoda musí identifikovat dostatečně
- Praktičnost – je možno používat s rozumným úsilím
- Přijatelnost – sociálně, osobně, lidsky přijatelné

Při návrhu systému musíme také zohledinit, v jakém prostředí a s jakými uživately bude systém fungovat, proto musíme zvážit následující:

- Spolupracující vs. Nespolupracující – uživatel (ne)spolupracuje při identifikaci
- Zjevné vs. Skryté – povaha snímání charakteristiky
- Aktivní vs. Pasivní – systém aktivně cíleně rozpoznává
- Obvyklé vs. Neobvyklé – užovatelé se s metodou identifikace setkávají na více místech
- Samoobslužné vs. S obsluhou – systém pracuje automaticky, nebo vyžaduje lidskou obsluhu
- Standardní vs. Nestandardní
- Veřejné vs. Privátní – cílová skupina prostředí pro nasazení
- Otevřené vs. Uzavřené – proprietární vs. Opensorce řešení

Je důležité prověření, zda vložená data jsou skutečně ověřovaného člověka. Z toho důvodu je většinou zavedení uživatele do systému prováděno pomocí obsluhy.

Zcela běžně se v problematice biometrie setkáváme s následujícími pojmy

- Biometrický vzorek – např.: otisk prstu, obraz obličeje
- Biometrická charakteristika – určující informace pro zpracování vzorku (křížení linií, vzdálenost bodů…)
- Biometrické markanty – charakteristiky rozhodující pro identifikaci
- Biometrická šablona – výsledek zpracování vzorku pro uložení identifikaci při porovnání s budoucími vzorky
- Šablony se optimalizují dle požadovaného úkonu (verifikace, autentizace)

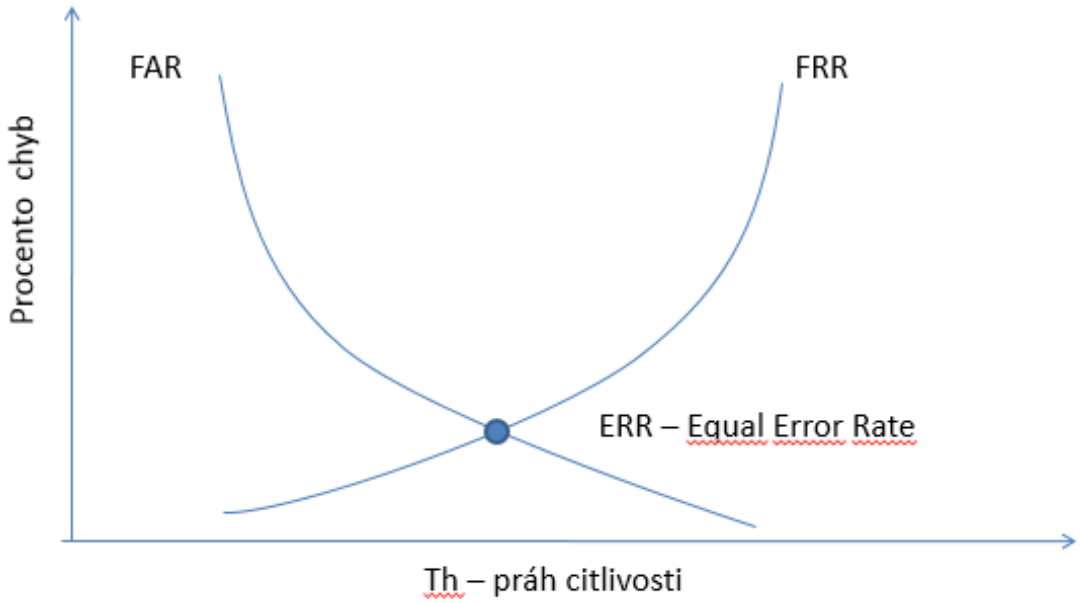
Jako u každého programu i u SW pro biometrickou autentizace měříme jeho výkon, který nění nikdy 100%. Máme dva základní ukazatele, FAR a FRR. FAR - False Accept Rate – je pravděpodobnost chybného přijetí (neoprávněnému je umožněn přístup), což se určuje jako:

$$FAR = \frac{N_{FR}}{N_{IIA}} = \frac{N_{FR}}{N_{IVA}}$$

Kde N_{FR} je počet chybně odmítnutých pokusů o identifikaci, N_{IIA} je počet pokusů neoprávněných osob o identifikaci a N_{IVA} je počet pokusů o verifikaci neoprávněných osob. Naproti tomu FRR - False Reject Rate – je pravděpodobnost chybného odmítnutí (oprávněný je odmítnut), což se určuje:

$$FRR = \frac{N_{FR}}{N_{EIA}} = \frac{N_{FR}}{N_{EVA}}$$

Kde N_{FR} je počet chybně odmítnutých pokusů o identifikaci, N_{EIA} je počet pokusů oprávněných osob o identifikaci a N_{EVA} je počet pokusů o verifikaci oprávněných osob. FAR a FRR jsou vzájemně provázány, a právě toto provázání, reps. průsečík průběhů těchto funkcí je klíčem k optimálnímu nastavení biometrického ozpoznání uživatele.



Proces pozorování vzorku a šablony je závislý na použoté metodě a její realizaci konkrétním algoritmem, avšak můžeme zcela obecně říct, že je možné jej posat následující funkcí:

$$s = Sim(P, P')$$

Kde P je šablona vytvořená při zavádění užvatele pro biometrické ověření, P' je vzorek sejmutý při pokusuo identifikaci užovatele, funkce Sim je míra ztotožnění a je předepsána použotým algoritmem, Th je práh, který udává minimální povolenou hodnutu míry ztotožnění. Pokud je

$$s \geq Th$$

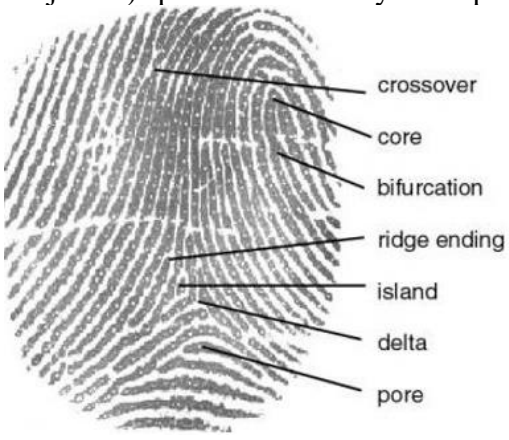
Došlo ke ztotožnění užovatele, jinka je uživatel odmítnut a pokud je oprávněný, tak musí absolvovat proces identifikace znovu, nebo se ověřit vůči systému jinou metodou, například pomocí čipové karty a hesla.

FAR/FRR zobecňují a předpokládají rovnoměrné zastoupení napříč celou skupinou uživatelů. Celově se nám systém povede lépe nastavit, pokud vypustíme užovatele, kteří se nemohou registrovat (bez prstu, bez oka…) tzv. FTE – failure to enroll, a uživatele, u kterých nelze pořídit vzorek (prsty bez otisku apod.) tzv. FTA – failure to acquire.

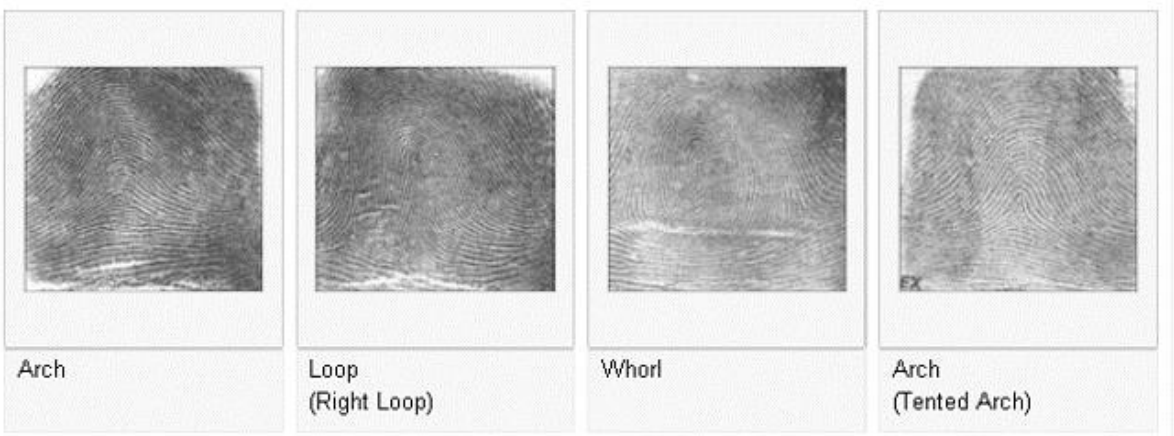
21.2.12.1. Otisky prstů

Identifikace uživatele na základě otisků prstu je relativně spolehlivá metoda prověřená dlouholetým používáním, poprvé se objevila ve starověké Číně. Cca jedno století je aktivně využívaná v moderní forenzní vědě a v posledních 30 let probíhá automatizace zpracování. Z hlediska využití v biometrii převažuje proces verifikace.

Následující obrázek ukazuje markanty otisku, jejichž přítomnost a poloha (především vzájemná) spolu s rotací se využává pro rozpoznání otisku v databázi.



Otisky prstů běžně ksifikujeme do následujících 4 typů:



S rozpoznáním a třízením otisků nám může pomoci CD s programy poskytované na vyžádání americkou vládou (NIST) zcela volně. CD může získat občan libovolného amerického a Evropského státu, po vyplnění žádosti a výměně několika emailů se zástupci NIST, nebo aktuální Opensorce project NIST si můžeme stáhnout.

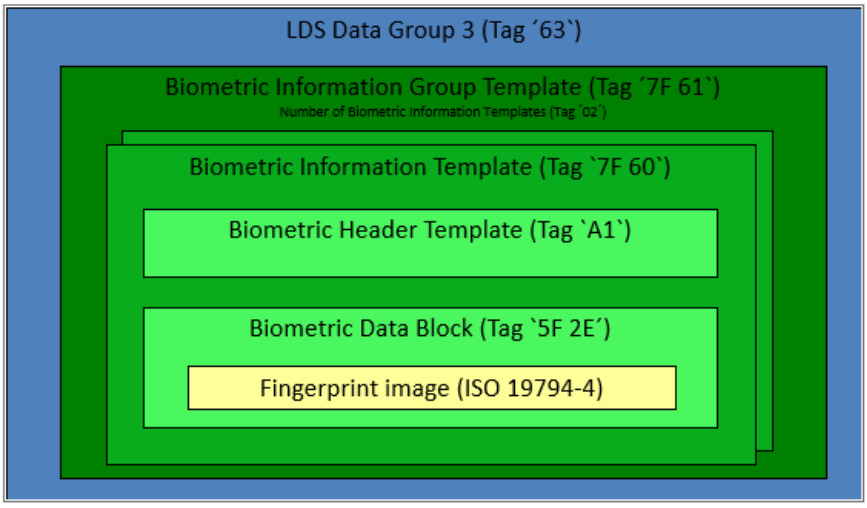
Pro snímání ootisků se běžně využívají následujcí dvě skupiny senzorů:

- Kontaktní – optické, elektronické, kapacitní, tlakové, teplotní, optoelektronické
- Bezkontaktní – optické a ultrazvukové

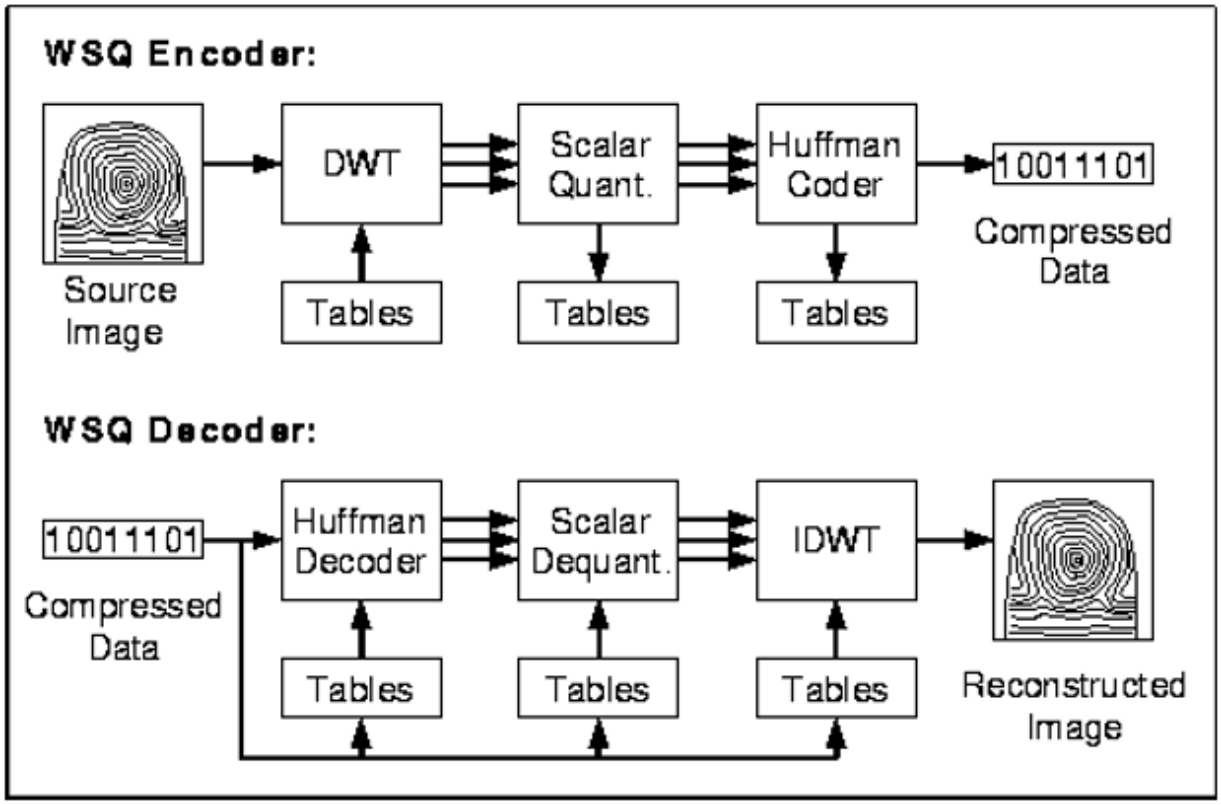
Pro výměnu a zpracování dat používáme standardyzované formáty ISO/IEC 19784-1:2005, Information Technology – BioAPI Specification v2.0 – a ISO/IEC 19794:2005, Biometric Data Interchange Formats:

- Part 1: Framework
- **Part 2: Fingerprint Minutae Data – souřadnice** a natočení markaty (textová data)
- **Part 3: Fingerprint Pattern Spectral Data**
- **Part 4: Finger Image Data** – formát "fotografie"
- Part 5: Face Image Data
- Part 6: Iris Image Data
- Part 7: Signatire/Sing Time Series Data
- Part 8: Finger pattern skeletal data
- Part 9: Vascular Image Data
- Part 10: Hand geometry silouhette data

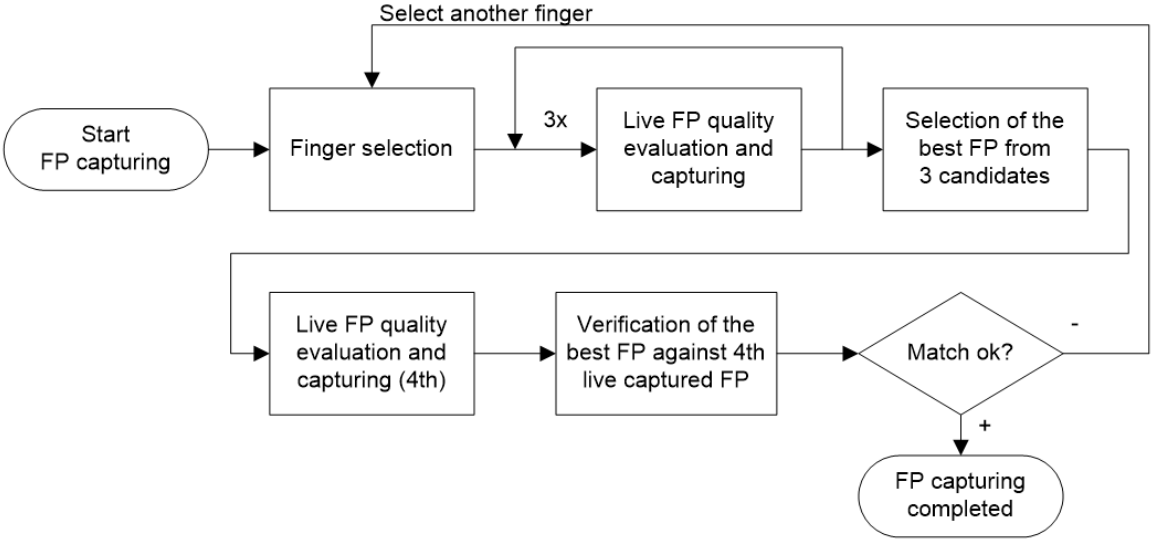
Kromě uložení a předávání dat se definuje i kvalita snímku pomocí doporučení **NIST IR 7151 – Fingerprint Image Quality** – definujícího stupnici od 0 do 5, která udává kvalitu snímku otisku prstu uloženého ve formátu LDS/CBEFF obdobně, jako další biometrické fotografie.



Speciálně pro otisky prstů byl NIST vyvynut formát WSQ (Wavelet Scalar Quantization), který byl následně standardizován ISO normou. Je založen na wavelet tranformaci, díky čemuž má velmi dobrý kompresní poměr a zároveň menší náchylnost pro vznik artefaktů, nežli JPEG.



Povinně jsou od roku 2009 v EU otisky prstů snímány a ukládány do Epasů všem žadatelům o cestovní pas. Sbírá se po jednom otisku prstu z každé ruky. Nicméně v ralizaci Epasů jsou sporné body ohledně otisků prstů, jedním je nejednoznačný standard uložení informace o chybějících otiscích, dnes již řešeno dodatkem ISO normy, další chybou návrhu je problematika nekvalitních otisků, jejichž ukládání (mazání) je řešeno novelou evropské legislativy. Biometrická data o uživateli Epasu jsou chráněna jako citlivá osobní data, z čehož plynou další problémy jako například, že ne každý stát může číst v případě potřeby otisky uložené v pase cizího státního příslušníka. Platnost certifikátu pro čtení je 3 měsíce pro cizí stát, pokud se povede získání certifikátu vyjednat, následně se celý administrativní proces opakuje. Proces vytvoření šablony otisku prstu uložené v Epasu popisuje následující obrázek.



Po vytvoření beta verze šablony, je proveden pokus o ztotožnění a pokud je úspěšné, dojde k vytvoření šablony. V pasu jsou uloženy 2 otisky prstu (jeden z každé ruky, pokud nelze, tak oba z jedné ruky) a je uloženo, o které prsty jde:

- 0,7% of applicants are not able to achieve NFIQ=3 or better on any FP
- 0,8% of applicants are able to achieve NFIQ=3 or better on 1 FP only
- 98,5% of applicants have no problem to achieve NFIQ=3 or better on 2 FPs

21.2.12.2. Obraz obličeje

Obraz obličeje je základní prostředek identifikace v mezilidských vztazích, ale není zcela vhodný pro využití v biometrii, protože se mění s věkem a jeho zpracování je složité a citlivé na pořízení referenčních dat. Převládá jeho využitá pro identifikaci uživatele. Lidský obličej ukrývá 12 bodů, které umožňují jeho rozeznání, o které body jde je vidět na následujícím obrázku.



Zcela v obecné rovině můžeme mluvit o dvou skupinách metod pro identifikaci tváře, jde o 2D a 3D metody, které se liší v tom, co zpracovávají. 2D metoody využívají klasický 2D obraz („fotografii“) obličeje, zatím co 3D metody pracují s prostorovým modelem tváře. Obě skupiny metod se ovšem potýkají s probléme nošení dioptrických brýlí, protože brýle jsou různě výrazným prvkem obličeje a zároveň přestavují nejčastěji se měnící prvek, nesmíme také opomenout zkreslení obrazu, který je vidět zkrze dioptrická skla. Dále pak obě skupiny metod trápí špatné osvětlení, zakrití části obličeje nebo různá natočení. Na druhou stranu tato metoda disponuje jednou velkou výhodou, kterou je možnost bezprostřední a snadné lidské kontroly správnosti identifikace.

Pro ukládání dat je doporučeno užít JPG200, nebo některý z formátů s bezstrátovou kompresí jako je např PNG. Běžně rozlišujeme dva druhy snímání, full frontall image a token image. Full frontall image využívá přímého pohledu do fotoaparátu a v závislosti na technlogii má definovaný rozměr snímku, nebo doporučenou minimální velikost snímku, na jejímž základě se definuje použitá velikost v jednotlivých řešeních. Tokem image považujeme za biometrický údaj a má pevně definovanou velikost, typycky je to vzdálenost očí 60/90 px. Pro oba typy snímání platí určitá pravidla, která jsou znázorněna na následujícím obrázku.



21.2.12.3. Biometrický podpis

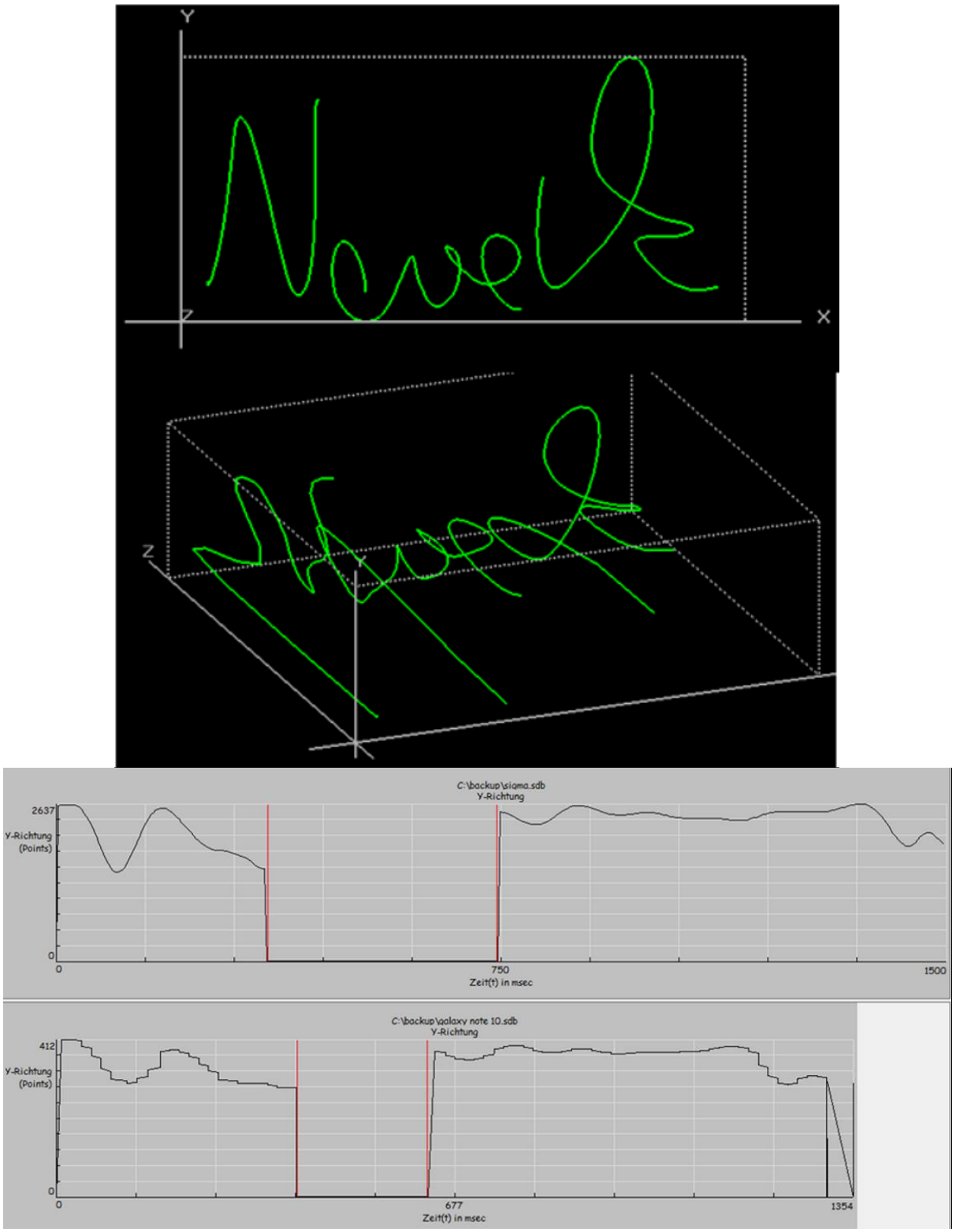
Biometrický podpis je zákonem uznávaný a vychází ze standartu eidas. Při podepisování se snímá:

- Tvar podpisu (x,y)
- Dynamika tvorby podpisu (čas)
- Síla přítlaku (z) - až 1024 úrovní tlaku

Důležitá je kvalita vzorkování - vzorkovací frekvence 500 Hz, vzorkování musí být opravdu hodně stabilní, aby byla zaručena dostatečná kvalita.

Historicky biometrický podpis vychází z grafologie, kdy se znalci snažili určit podepisujícího podle toho, jak vypadala čára podpisu. Grafologie ovšem má i dost silnou neodbornou stránku a tou jsou pokusy o sestavování profilu pisatele na základě jeho písma, ovšem jediné, co je na písmu v tomto směru pozorovatelné je aktuální rozložení pisatele (nervozita apod.).

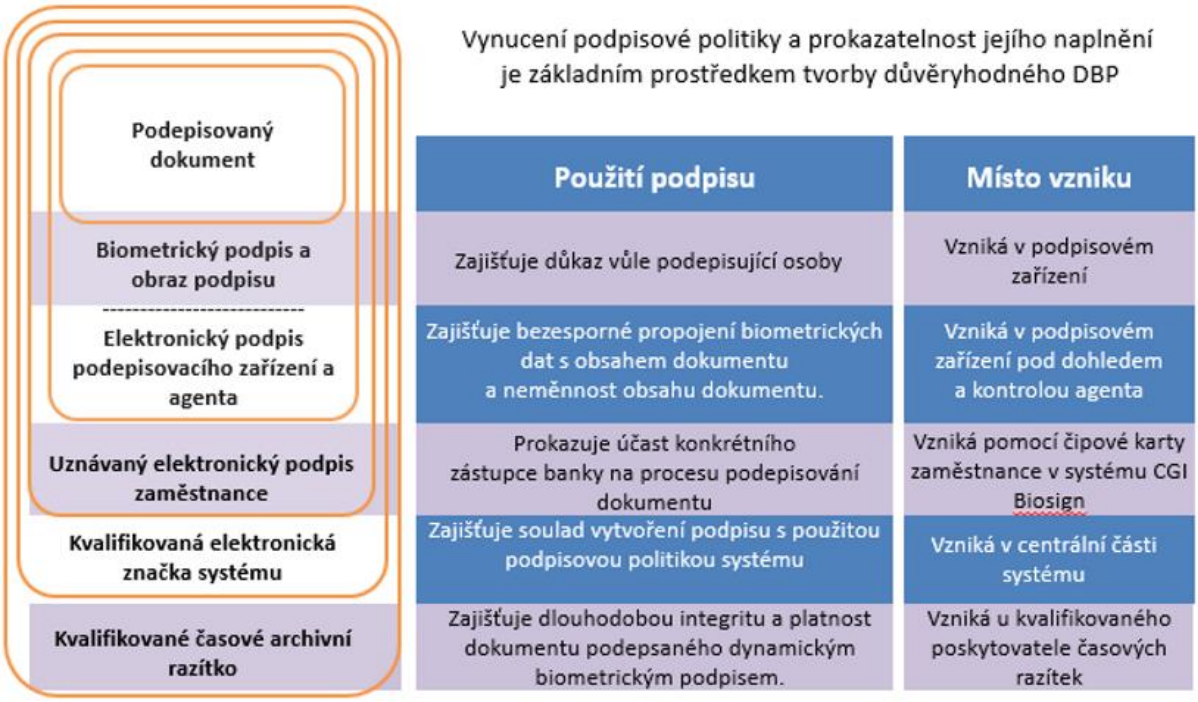
V rámci legislativního procesu to nemá biometrický podpis jednoduché, ačkoli se dnes čím dál tím častěji stává běžným v bankovnictví a pojišťovnictví. Existují snahy (s podporou soudních znalců) legitimizující biometrický podpis jako alternativu kryptografickému elektronickému podpisu. Za splnění podmínek může být akceptován biometrický podpis na stejnou úroveň, jako běžný elektronický podpis. Je nutné splnit: důvěryhodné pořízení, ochranu důvěrnosti biometrického vzorku a ochranu integrity dokumentu včetně prokazatelného připojení podpisu k dokumentu. Následující obrázek je ukázkou toho, jak mohou vypadat data biometrického podpisu v podobě čitelné člověku.



Zcela obecně máme dva druhy biometrického podpisu, s biometrickým podpisovaným vzorem (problémy se spolehlivostí verifikace) a bez biometrického podpisového vzoru (problémy s ověřováním pravosti – nutno přizvat znalce). Při praktickém využívání se design systému nechává posoudit znalcem a finální návrh systému je opět posuzovaný znalcem. V průběhu používáná se znalecký posudek systému pravidelně audituje.

Biometrická data se zašifrují pomocí veřejného klíče, ke kterému nemá organizace k dispozici privátní klíč a vlastní dokument je podepsán biometrickým podpisem i

kvalifikovaným elektronickým podpisem organizace. Jak vypadá finální struktura podpisu dokumentu ukazuje následující obrázek.



Běžně se v souladu se standardem PADES podepisují dokumenty formátu pdf, pro další formáty dokumentů máme standardy XADES a CADES.

Je vžd ypotřeba prokázat původ podpisu, proto je nutné používat důvěryhodná zařízení a důvěryhodný proces, za který svým podpisem ručí garant (dohlížející osoba). Aby byl dokument považován za platný, musí být platný podpis na dokumentu a ten je platný pouze v případě, že je platné časové razítko, které organizace oobdrží v podobě certifikátu a je platné po 4 roky. Před vypršením časového razítka je nutné dokument odeslat na časovou autoritu a nechat doplnit nové časové razítko, které je nabaleno jako další vrstva na staré. Následné časové razítko ověřuje platnost předchozího časového razítka, nikoliv dokument samotný. Je nutné zajistit akci obnovení platnosti dokumentu s dostatečným předstihem, protože všechny dokumenty podepsané konkrétní sadou razítek vyprší v jeden okamžik. Druhou možností, která vyvolává nutnost přerazítkování je vypršení platnosti šifrovacího algoritmu, ještě v době platnosti původních algoritmů dojde k přešifrování celého dokumentu v šifrované podobě novým algoritmem a znovu orazítkování.

21.2.12.4. Psychologické aspekty biometrie

Při nasazení biometrie musíme pamatovat krom bezpečnsoti z oblasti IT, také především na pohodlí a bezpečnost uživatele. Musíme si uvědomit, že sběr některých typů vzorků může být pro uživatele zátěží, proto se snažíme sběru těchti vzorků v řešeních vyhnout. Dalším překážkou může být náboženské přesvětšení uživatele, které nám může komplikovat především biometrické fotografie obličeje.

Další stránkou, na kterou musíme myslet jsou uživatelé, kteří mají osobní důvody k neposkytnutí biometrického vzorku, těmi důvody bývá nemožnost jeho poskytnutí, vážný úraz a jeho následky, či nemoc nebo postižení. Tito uživatelé jsou vystaveni vysoké psychické zátěži, pokud se budeme snažit o získání vzorku, nebo pokud je procesem získávání vzorku „vyčleníme“ z kolektivu. Pokud máme v kolektivu uživatele, kteří nemohou z nějakého důvodu poskytnout vzorek, musíme tomu uzpůsobit proces sběru vzorků, abychom je nevyčlenili z kolektivu a zároveň pro ně musíme vymyslet vhodnou a pohodlnou alternativu autentizace.

21.2.12.5. Ochrana a podvežení biometrických údajů

Biometrické vzorky jsou dle zákona o ochraně osobních údajů chápány jako citlivá data, čemuž musí odpovídat bezpoečnsotní opatření jak fyzická, tak programová. Zvýšený důraz na ochranu dat je pod kontrolou inspektorů ÚOOÚ a je potřeba jej brát opravdu vážně. Pokud není pořízení biometrického vzorku nařízeno ze zákona (otisky prstů v pase), tak je potřeba explicitní písmný souhlas uživatele. Zařízení pro soukromou potřebu (mobilní telefony s čtečkou otisku prstu, počítače a telefony s Windows Halow apod.), která užívají biometrická data přijímají explicitní souhlas uživatele pomocí souhlasu s licenčním ujednáním SW, kde je tento souhlas explicitně uveden.

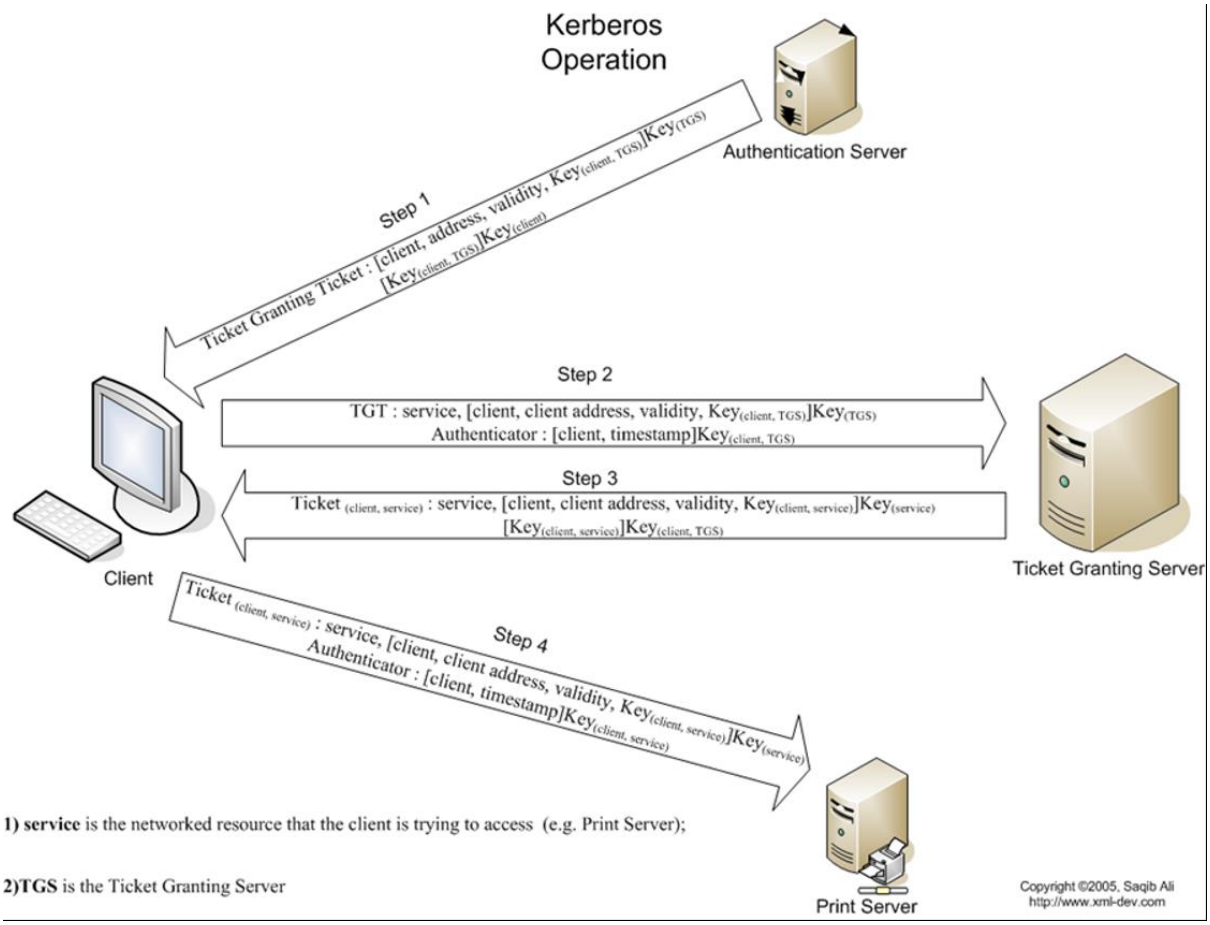
Když se na biometrické vzorky podíváme z druhé strany, tak je potřeba systém vybavit ochranými algoritmy proti podvžení biometrického vzorku. Nejčastěji se biometrický obraz duhovky či obličeje podvrhuje pomocí fotografie a otisk prstu pomocí jeho silikonové kopie (tzv. gumáku). Ochranou je ve všech přípradech kontrola živosti objektu. Pro čtečku otisku prstů je vhodné využít kontrolu teploty subjektu spolu s měřením elektrického odporu nebo vhidněji pomocí detekce pulsu. V případě biometrické fotografie obličeje je možné biometrickou černobílou fotografii doplnit o infračervenou fotografii objektu, nebo o zaostření na duhovku, kde jde použít stejný mechanismus jako pro kontrolu živosti při ověření obrazem duhovky. Duhovka není stabilní objekt, i když jsme si toho v zrcadle, nebo na svém protějšku nikdy nevšimli, tak duhovka neustále pulsuje, což je jedním z klíčů, jak poznat živou duhovku od fotografie. Další možností, pro uživatele značně nepříjemnou je rychlá změna intenzity osvitu, fotografie zůstane neměnná, zatímco žiivá duhovka změní svou viditelnou část v důsledku reakce oka na změnu osvětlení.

21.2.13. Kerberos protokol

Je síťová autentizační služba vycházející z principu sdíleného tajemnství. Pro to, aby mohla být použita usí být splněny 3 předpoklady, online spojejí, synchronizovaný čas a existence KDC (Key Distribution Center). Nejznámnějším příkladem jejího použití je přihlášení se k počítači s Windows pomocí čipové karty. Každému uživateli či službě je přiřazeno PrincipalName, které má tvar usernama@domena.xx. Dalším základním pojmem je ticket, což je Datová struktura potvrzující identitu (prostřednictvím třetí důvěryhodné strany - KDC), která je kriptograficky chráněná a odolává všem základním síťovým útokům. Doba platnosti ticketu je běžně 8 hodin, ale je možné nastavit jinak. Autentizace sůběktů využívajcích kerberos je vzájemná a jejím výsledkem je kllíč a identita.

Velká síla spočívá v tom, že aplikace nevidí dovnitř kerberosu, ale pokud chceme v aplikaci použít ověření uživatele pomocí kerberos protoku, tak použijeme GSS API. S využitím GSS je možné pro přenos využít HTTP a v Microsoftu vymysleli SPNEGO, které umožňuje pomocí prohlížeče využít kerberos pro přihlášení k serveru, typycky se toho využívá v SharePointu a Exchange. Server má svoje principal name ve tvaru: nazevPocitace@domena.xx/jenoSlužby.

Autentizace probíhá následujícím postupem. Ticket garanting server posílá lístek na službu, který obsahuje sesion key zabalený tak, že v je tam 2 krát, jednou se dešifruje na požadující stanici a druhá část obsahující stejný sesion key se dá dešifrovat pouze na cílové službě. Zbývající popis přehledněji ukáže následující obrázek.



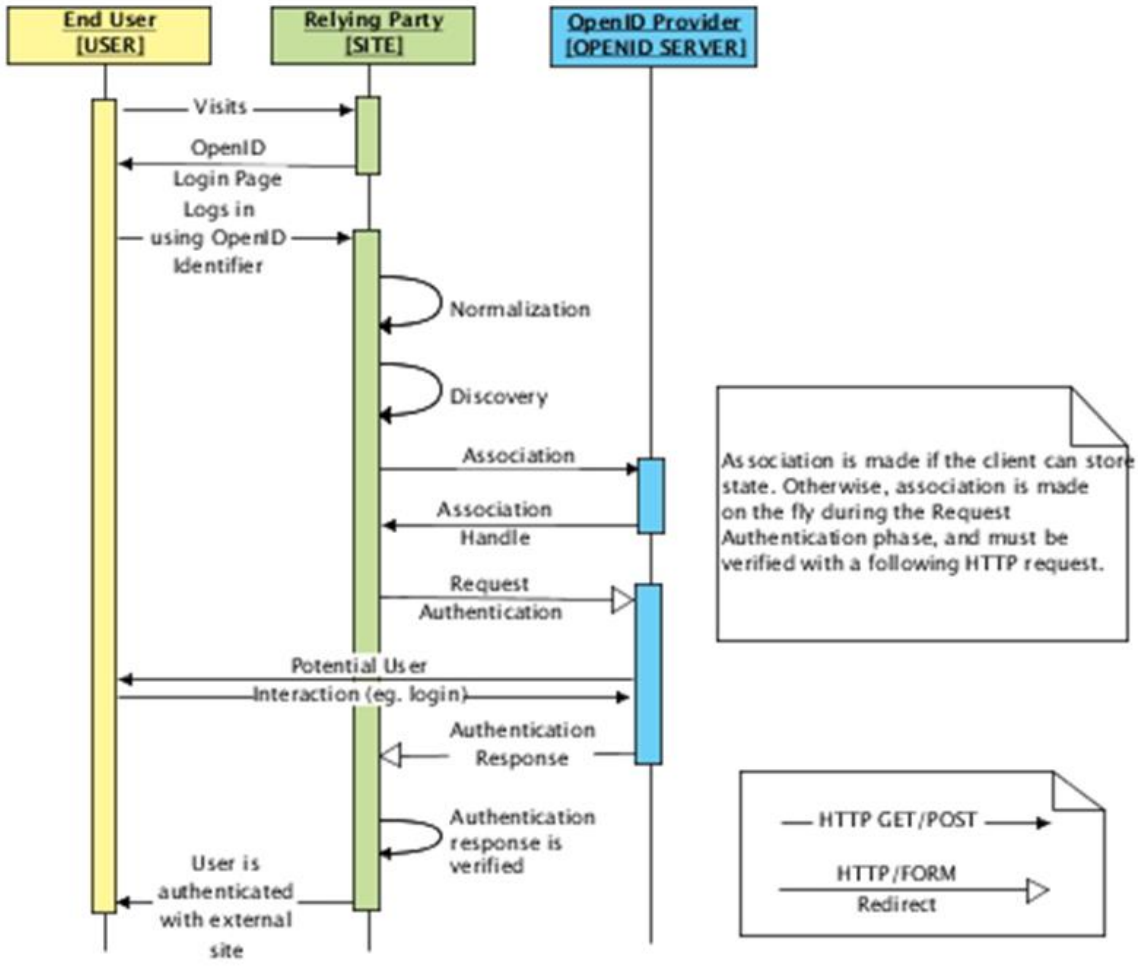
21.2.14. Federalizace identit

O federalizaci mluvíme při překročení hranic domény. Cílem federalizace je úplná implementaci Single Sign On. Při federalizaci pohlížíme na celek jako na spolupráci dvojice prvků, Identity provideta a Servicce providera. Service provider poskytuje ve výsledné síti sluýby a využívá služeb poskytovatele identity, aby nemusel implementovat vlastní ověřování uživatele. Často dostává identitu pomocí Kerberosu. Identity provider (poskytovatel identity) poskytuje ověřené identity uživatelů dle definované politiky. Ručí za údaje uživatelů a za to, že služba komunikuje opravdu s uživatelem, se kterým si myslí, že komunikuje.

Sinlge Sign On (SSO) je mnohdy nedosažitelný ideál autentizace uživatle. Uživatel se pro přihlášení ke stanici již nadále nemusí přihlašovat a při tom jeho identita je bezpečně předávána mezi službami. U služeb v rámci firemní sítě se tak nejčastěji děje pomocí Kerberosu, na internetu je pak kromě Kerberosu využíváno Open ID Connect nebo PKI.

Nyní si něco málo řekneme o částčně fungujících příkladech, které známe ze svého každodenního života. Jedním z příkladů může být ověření uživatele na eshopu pomocí pčtu Google, Facebook, nebo MojeID. Protože služba Moje ID, která je provozována společností cznic, realizuje Open ID a umožňuje různé stupně ověření uživatele, tak si o níněco málo řekneme. V první řadě jde o službu, která je podporována českými poskytovateli služeb a vybranými poskytovateli služeb na Slovensku. Služba Moje ID má různé stupně ověření uživatele, čemuž odpovídají i různé způsoby přihlášení uživatele. Nejzákladnějším způsobem ověření uživatle je email, dalším stupněm je poštovní adresa ověřená zasláním dopisu s dokumenty k podpisu a nejvyšší mírou ověření je kvalifikovaný certifikát, kdy je nutná odobní návštěva poskytovale identiy. Základním způsobem přihlášení se k Moje ID je jméno a heslo pro nižší stupně ověření, pro vyšší stupně ověření je vydán certifikát.

Standard OpenID má velmi jednoduchou specifikaci a je určen pro prostředí interntu. Jeho fungování asi nejlépe vystihuje následující obrázek.



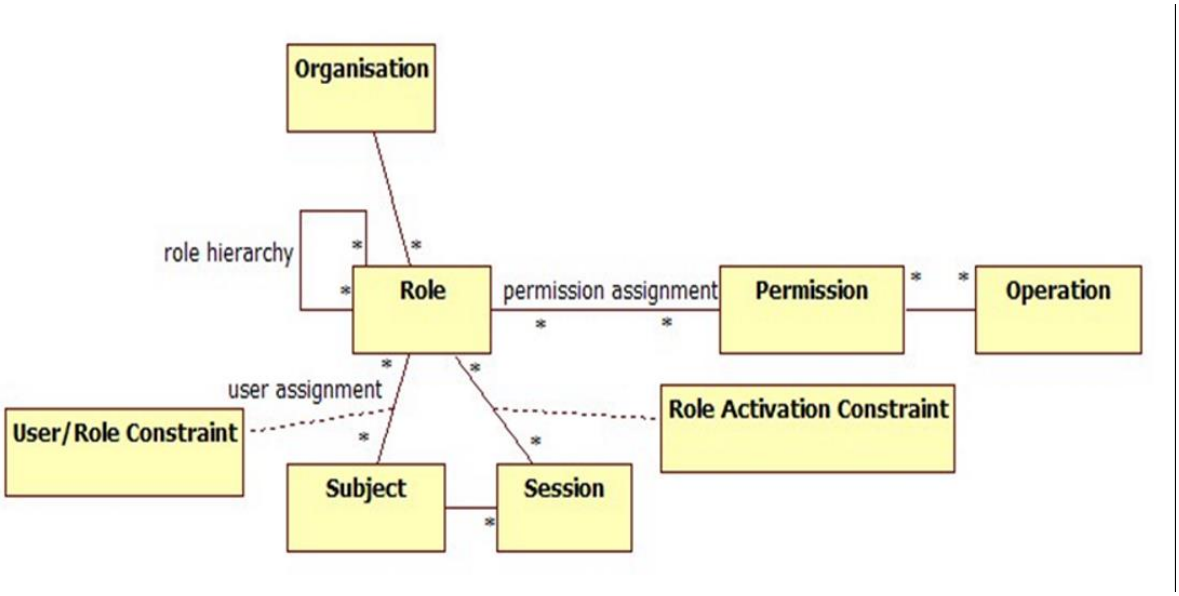
21.3. Autorizace

Autorizace je určení, zda daný subjekt (po autentizaci) je oprávněn přistupovat k aktivu, tedy funkce F(subjekt, operace, aktivum) -> {povolit, odepřít}. Funkce autorizace může být rozšířena o další parametry (např.: místo, odkud dochází k přístupu).

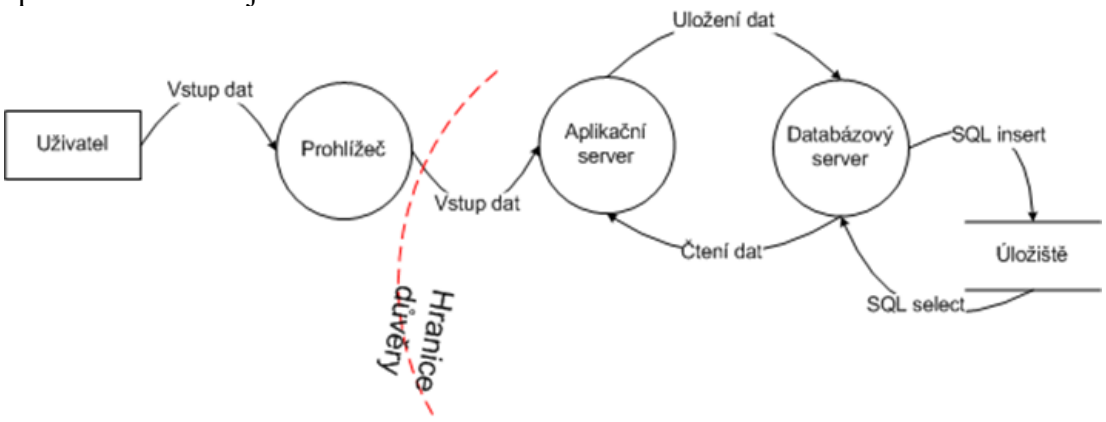
DAC - Dicretionary Access Control je jednou ze základních metod autorizace. K objektu je připojen seznam oprávnění pro jednotlivé subjekty (uživatele, skupiny) s popisem oprávnění pro tu kterou operaci. Oprávněné subjekty (vlastník) mohou poskytovat oprávnění dále, běžnou realizací jsou UNIX oprávnění a Windows ACL. V této metodě není vyšší instance, nežli vlastník objektu, na Windows může vlastnictví převzít administrátor systému, v případě zařazení počítače do domény pak doménový administrátor. Chybným nastavením práv si vlastník může i sám sobě podříznout práva.

MAC - Mandatory Access Control další základní metodou autorizace. Přístupová oprávnění jsou určována globální politikou na základě atributů objektů a subjektů, není zde zohledňováno, za je uživatel vlstníkem objektu či nikoliv. Poměrně často se s MAC můžeme setka na SeLinuxu a různých rozšíření UNIXů.

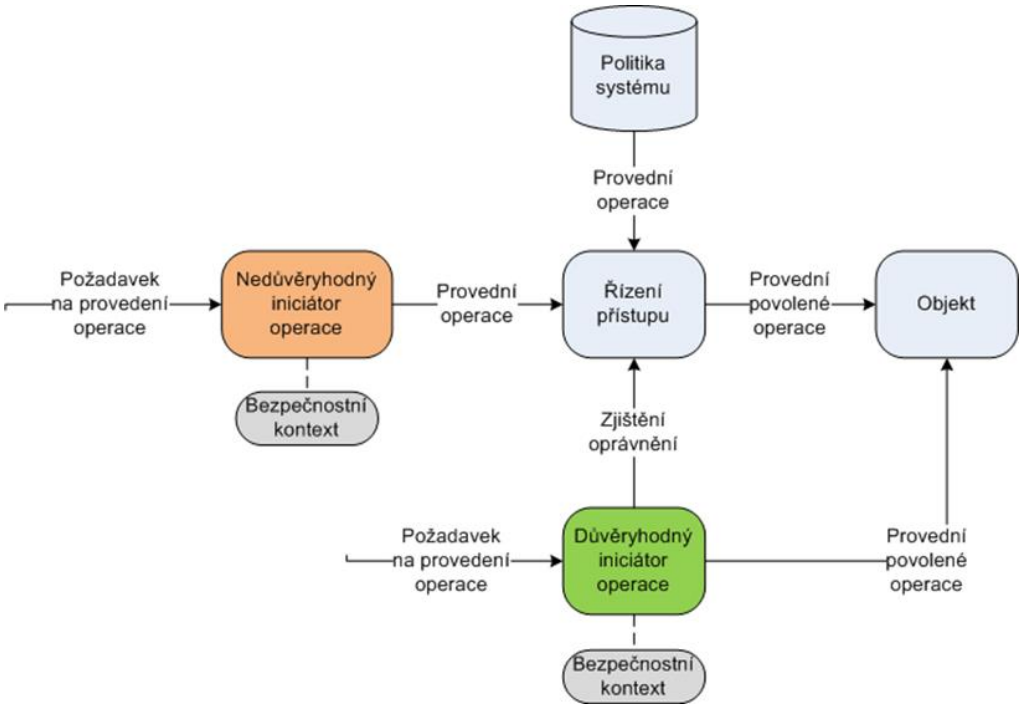
RBAC - Role Based Access Control je metoda autorizace, která je kombinací DAC a MAC. Umožňuje stavět hierarchie, dědičnosti a tranzitivitu, přiřazování role lze ověřovat na základě vlastností session.



Při provádění autorizace je nusté stanovit hranice důvěry jak je to vidět např. na webové aplikaci na následujícím obrázku.



Pokud známe hranice důvěry, tak je potřeba se ještě podívat na to, z jakých komponent se autorizace vůbec skládá.



Webový prohlížeč není důvěryhodnou komponentou, pouze zobrazuje uživatelské rozhraní a přijímá vstup uživatele včetně autentizačních údajů. Běží pod identitou uživatele v OS, která je možná propsat na aplikační serrver, ale není důvěryhodné.

Princip oddělní zodpovědností je významný prvek bezpečnosti systémů. Jeden uživatel nemůže sám provádět všechny citlivé operace, pro dokončení operace je nutná účast více uživatelů v různých rolích. Běžnou administrátorskou praxí je koncept "čtyř očí", který je uplatňován například při správě účtů, kdy by pro vytvoření uživatle měla spolupracovat minimálně dvojice správců, jeden založí uzivatele a druhý mu přiřadí oprávnění. Důležité je, že jeden druhého nemohou zastoupit, ale i tak je potřeba kombinace s důsledným auditingem.

Autorizace operací se používá pro provádění citlivých operací v prostředí internetu. Autentizovaná role je využita pro zadání citlivé operace (platební příkaz) a je vyžadována dodatečná explicitní autorizace dané operace zadavatelem - SMS kód, elektronický podpis, EMV CAP/DPA potvrzení, telefonické schválení apod. Vlastní autorizaci provádí systém na konci, člověk provádí dodatečnou autentizaci.

22. Obsah přilženého DVD

Jak již bývý zvykem, tak k učebnicím informatiky se přikádají CD nebo DVD, tak i tato kniha tento nepsaný zvyk dodržuje. A co vlastně na přiloženém disku (v případě elektronické verze knihy iso otisku disku) nalezeneme?

- ✓ Elektronickou verzi knihy ve formátu xps – první verzi
- ✓ Projekty MS Visual studio pro příklady v jzyce C++ a další v knize neuvedené kódy
- ✓ Návod na Visual studio 2013 a tvorbu instalačních souborů z našich prográmků
- ✓ Microsoft Net Framework 3.5.1 a 4.5.1.
- ✓ Microsoft Visual Studio 2013 for Desktop
- ✓ Český jazykový valíček Visual studia
- ✓ Formát Factory
- ✓ 7zip
- ✓ Zkušení verzi Winraru
- ✓ Nástroj na formátování USB jednotek a paměťových karet
- ✓ AVI Demux
- ✓ Nástroj CPU-Z pro zjištění výkonu HW, volných paměťových slotů atd.
- ✓ PS Pad
- ✓ BlueJ cz
- ✓ Java JDK 8.2
- ✓ Crystall disk info

Pokud zapomenete adresu, tak je možné obraz přiloženého disku nalést na adrese:

Heslo: muj_web/učebniceICT

- ✓ Part1: http://ulozto.cz/xUgGqBbB/disk-k-ucebnici-ict-zip-001
- ✓ Part2: http://ulozto.cz/xN9VUso4/disk-k-ucebnici-ict-zip-002
- ✓ Part3: http://ulozto.cz/xxUgLmZa/disk-k-ucebnici-ict-zip-003
- ✓ Part4: http://ulozto.cz/xspZ9VEA/disk-k-ucebnici-ict-zip-004

Pro rozbalení těchto komprimovaných souborů je potřeba užít 7zip, až po extrahování 7zipem se ukáže iso otisk přiloženého disku.

Záložní obraz:
http://www.edisk.cz/stahni/49793/Disk_k_ucebnici_ict.zip.001_1.95GB.html
http://www.edisk.cz/stahni/97442/Disk_k_ucebnici_ict.zip.003_1.95GB.html
http://www.edisk.cz/stahni/86662/Disk_k_ucebnici_ict.zip.002_1.95GB.html
http://www.edisk.cz/stahni/69609/Disk_k_ucebnici_ict.zip.004_971.85MB.html
Adresář souborů: http://www.edisk.cz/adresar/968f41fe7c7766e9d54c622244a49e08

23. Zdroje

Hlavním zdrojem jsou vlastní přednášky umístěné na webu: http://elektrotechnika.vesele.info obsahující zdrojové informace k jednotlivým tématům, dále pak přednášky ze studia na Přírodovedecké fakultě JCU.

Lasák , Pavel. 2014. Jak na Excel. [Online] 5. 2. 2014. [Citace: 23. 7. 2014.] http://office.lasakovi.com/excel.

FILKA, Jaroslav. Metodika tvorby diplomové práce. Praktická pomůcka pro studenty vysokých škol.1.vyd. Brno : Knihař ve spolupráci s CMTF UP v Olomouci, 2002.ISBN 80-86292- 05-3.

KAPOUNOVÁ, Jana. Formální úprava diplomové práce. 1. vyd., dotisk. Ostrava : Ostravská Univerzita,1998. ISBN 80-7042-141-X.

ČESKÝ NORMALIZAČNÍ INSTITUT. ČSN 01 6910.Úprava písemností psaných strojem nebo zpracovávaných textovými editory. 1. vyd. Praha : Český normalizační institut,1997.36s.

ČESKÝ NORMALIZAČNÍ INSTITUT. ČSN ISO 690 (010197) .Dokumentace – Bibliografické Citace : obsah, forma ,struktura. 2.vyd. Praha : Český normalizační institut,1996.

ČESKÝ NORMALIZAČNÍ INSTITUT. ČSN ISO 7144. Dokumentace – Formální úprava disertací a podobných dokumentů. Praha : Český normalizační institut, 1996. 24 s.

KEN, Jones. Understanding Microsoft Virtualization Solutions. 2. vyd., A C1P catalogue record for this book is available from the British Library : Microsoft Press, 2010.

HONEYCUTT Jerry. Introducing Windows 8 An Overview for IT Professionals. Microsoft Press, 2012, ISBN: 978-0-7356-7050-1.

HORTON, Ivor. Java 5. Praha : Neocortex spol s.r.o., 2005. ISBN 80-86330-12-5.

KENT, Jeff. C++ bez předchozích znalostí. Brno : computer press a.s., 2009. IBSN 978-80-251-2411-6

PROKOP, Jiří. Algoritmy v jazyku C a C++. Praha : Grada Publishing a.s., 2012. IBSN 978-80-247-3929-8

24. Rejstřík

.

.NET, 1, 121, 122, 185

A

Adobe, 123, 142, 185
Android, 140, 141, 142
Aplikace, 26, 125, 144, 149
Apple, 41, 105, 110, 111, 115, 137, 138, 140, 141, 147
Arial, 153, 187

B

Bibliografické citace, 155
bibliografie, 150
Binární, 27, 29
BIOS, 26, 60, 61, 71, 82, 115
buňka, 211

C

Calc, 207
CD, 75, 76, 78, 89, 90, 113, 115, 121, 138, 139, 153, 156
citace, 152, 155, 156, 173, 174
Citace, 150, 156, 157, 626
CPU, 61, 62, 63, 66, 71, 72, 74, 78, 91, 111, 147
ctrl, 164

Č

Číslo, 28, 119, 156, 163
číslování, 149, 150, 170, 179, 189
Číslování stránek, 153, 163
ČSN ISO, 152, 155, 157, 626
ČSN ISO 690, 152, 155, 157, 187, 626

D

dělení, 28, 32, 105, 150, 155
Dělení slov, 154
docx, 48, 147, 183
DVD, 73, 76, 78, 85, 90, 92, 121, 123, 139

E

editor, 26, 138, 148, 149, 155
enter, 40, 164
Excel, 112, 143, 144, 147, 176, 177, 187, 207, 208, 209, 211, 214, 215
export, 162
externí, 66, 68, 71, 76, 77, 81, 86, 87, 91, 104, 187

F

filtry, 43, 44, 60, 123, 223
Firmware, 26, 40, 71
font, 189
formát, 46, 88, 123, 149, 150, 153, 160, 167
formátování, 149, 150, 175, 184, 189

G

galerie, 160, 172
GB/s, 27, 65, 72, 73, 79
graf, 149, 176, 177, 178, 179, 189, 217, 219, 222
grafika, 41, 58, 67, 153
Grafy, 176

H

HDD, 23, 63, 66, 74, 76, 85, 87, 88, 94, 104, 109, 117, 118, 120, 147
Hexadecimální, 29
Hypertextový odkaz, 189
Hz, 27, 51, 55, 60, 68, 71, 75

I

IBM, 31, 33, 35, 36, 64, 69, 73, 85, 86, 109, 110, 113, 115
Ilustrace, 149
Import, 162
importovat, 160
INSTITUT, 157, 626
iOS, 141, 142
IOS, 140, 141
ISO, 135, 155, 179, 184, 185

J

jednotka, 27, 66, 67, 72, 108, 115

K

karta domů, 157, 160
karta reference, 150
Karta vložení, 163
kartu Návrh, 149
kartu vložení, 149
klávesnice, 23, 33, 39, 40, 41, 63, 81, 111, 113, 144
knihovna funkcí, 216
kód, 23, 26, 29, 33, 40, 46, 66, 67, 84, 150
Komentář, 149, 189
Konce, 169
kontextové menu, 158, 161
kopírovat, 149, 162, 181, 188
kopírovat formát, 188
Kreslení, 189

L

laser, 43, 90
Linux, 1, 26, 88, 105, 109, 110, 139, 140, 141

M

Macintosh, 137, 138, 207
MacOS, 26, 105, 109, 111, 138
mechaniky, 26, 39, 49, 60, 76, 85, 87, 90, 91, 139
Mezery, 154
Microsoft, 41, 72, 75, 111, 112, 115, 116, 119, 120, 121, 122, 125, 126, 135, 136, 142, 143, 144, 147, 148, 626
monitor, 23, 24, 25, 49, 50, 55, 72
Multimedia, 149
myš, 24, 25, 41, 42, 43

N

nadpis, 154, 160, 187
nadpisy, 153, 169, 187
náhled, 151

O

Obrázek, 142, 221
Obrázky, 172
obsah, 29, 31, 79, 112, 126, 150, 152, 157, 163, 165, 626
Obsah, 2, 60, 150, 152, 166
oddíl, 171, 188
Odkazy, 149, 152, 156
odrážky, 149, 189
odstavec, 160, 167
Odstavec, 149, 150, 189
Office, 115, 125, 127, 139, 143, 144, 147, 148, 149, 160
okraj, 153, 163, 166, 167
okraje, 90, 150, 166, 167
Operační systém, 26, 147
OS, 26, 40, 76, 82, 87, 88, 92, 94, 104, 105, 106, 107, 108, 109, 110, 111, 113, 115, 135, 136, 137, 138, 139, 140, 141, 142, 145
Ovladač, 26, 40

P

PDF, 123, 185, 186
Periferie, 39
písma, 25, 35, 46, 149, 153
písmo, 126, 137, 153, 172, 182, 187, 193, 194
Písmo, 149, 189
Počíta, 23
počítač, 1, 23, 24, 25, 32, 33, 34, 36, 38, 41, 70, 74, 75, 78, 91, 92, 104, 105, 106, 109, 116, 118, 121, 142, 148, 160
podklady, 185, 187
Polovodič, 26
položka, 92, 148, 152, 166
PowerPoint, 143, 144, 147, 149, 187
Poznámky, 150, 152, 154, 186
Poznámky pod čarou, 150
prezentace, 1, 136, 144, 184, 187, 190, 191, 194
Procesor, 66, 92, 115, 116, 117, 118, 125, 135
program, 23, 26, 40, 44, 46, 48, 78, 82, 104, 105, 118
prohlížeč, 26, 144, 147
přechod, 35, 142, 187
převod, 28, 75
Přílohy, 153

R

RAM, 23, 61, 68, 69, 72, 79, 80, 81, 90, 91, 92, 94, 117, 119, 124, 127, 135, 139, 147
Rejstřík, 150, 627
Rok, 156
Rovnice, 179, 180, 221, 222
rozložení, 39, 150, 166, 170, 175, 188
rozložení stránky, 150, 166

Ř

řádkování, 51, 149, 154, 189
Řádkování, 154

S

sada, 26, 60, 67, 143, 147
Sazba, 153
sdílet, 137, 151
sešit, 184, 209, 211
Seznam, 152, 153
Schránka, 149, 188
směr textu, 189
Snímky, 188, 189
Software, 24, 26, 137, 141
soubor, 48, 88, 94, 104, 105, 127, 137, 138, 150, 153, 162
Soubor, 88, 185, 210, 222
soustava, 27, 29, 31, 44, 62
spravovat, 121, 158, 161
Spustitelný soubor, 26
Stať, 152
Stránky, 149, 153
styl, 157, 159, 160, 162, 166
styly, 137, 157, 158, 160, 161, 162, 163
Styly, 149
SUMA, 214, 216
SW, 1, 24, 26, 27, 30, 48, 70, 109
Symbian, 140, 142
symbol, 149
Symbol, 221, 222
Symboly, 149, 189
systém, 24, 26, 29, 60, 61, 82, 88, 91, 94, 104, 105, 106, 110, 111, 113, 114, 115, 116, 119, 120, 121, 122, 123, 124, 125, 127, 135, 136, 137, 138, 139, 140, 141, 142, 148

Š

Šablona, 172

T

tabulátor, 167
Tabulka, 28, 29, 68, 175, 211, 212, 213
Tabulky, 149
Text, 125, 149, 156
Times New Roman, 153, 187
tisk, 55, 56, 57, 123, 150, 151
Tisk, 56, 222, 223
Titulky, 150
Titulní strana, 152, 171
Tranzistor, 26

U

UGI, 26
Úkol, 164
uložit, 79, 104, 107, 151
úprava, 150, 152, 157, 626
Úpravy, 149, 171, 189
USB, 23, 39, 40, 42, 43, 44, 46, 49, 63, 66, 75, 76, 86, 91, 92, 104, 118, 135
Uspořádat, 150
Úvod, 152

V

VBA, 208
video, 71, 73, 81, 91, 92, 136, 148, 149, 187, 189
Virtualizace, 94, 108
Visual Basic for Applications, 208
vložení, 149, 150, 170, 171, 174, 175, 176, 177, 179, 188, 189, 194
vodoznak, 150, 181, 182
Vyhledávání, 148
vyjmout, 149, 188
výpočet, 23, 214, 216
Vzhled stránky, 150
Vzorec, 214, 216
Vzorkovací frekvence, 26

W

Windows, 1, 26, 40, 42, 70, 88, 94, 104, 105, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 147, 185, 186, 626
Word, 89, 91, 112, 137, 143, 144, 147, 148, 149, 150, 152, 154, 155, 160, 163, 164

X

XML, 121, 123, 143, 183, 184, 185, 186
XPS, 123, 125, 185, 186

Z

Záhlaví, 149
zápatí, 149
zarovnání, 149, 175, 179, 189
zařízení, 1, 23, 24, 26, 30, 31, 32, 33, 40, 42, 43, 46, 47, 49, 50, 63, 66, 69, 70, 72, 73, 76, 80, 83, 85, 87, 88, 91, 92, 104, 111, 116, 118, 123, 125, 136, 140, 141, 142, 144
Závěr, 152
Zdroj, 69, 70, 74
Zdrojový kód, 26
Zkratky, 155
změnit, 142, 159, 160, 166
zvuk, 49, 60, 185, 187, 189

Ž

žena, 36, 37