Datové struktury v jazyce C

Petrásek Jan

Pole

Pole

- Nejjednodušší statická struktura
- Je možné definovat i dynamicky
- Všechny prvky stejného datového typu
- Přístup k prvkům pomocí indexů
- Statická pole: viz Pole
- Dynamická pole: viz ukazatele

Struktury

Enum

- Výčtový typ slouží k definování seznamu symbolických konstant
- Hodnoty jednotlivých symbolických konstant začínají nulou a postupně se zvětšují o jedna
- Zle nastavit také specifické číselné hodnoty:
  typedef enum
  { zelena = 2, oranzova =4
  }barva;

```
5  //definice enum
6  typedef enum //zahájení definice
7  {
8      zelena, oranzova, cervena
9  }barva; //pojmenování enum
10
11 void main()
12 {
13     barva semafor;
14     semafor = zelena;
15     switch (semafor)
16     {
17     case 0:
18         printf("Můžeš jít");
19         break;
20     case 1:
21         printf("Připrav se");
22         break;
23     case 2:
24         printf("Stůj");
25         break;
26     default:
```

Struktury

Struktura

- Jedná se o datový typ (záznamový typ)
- Do jedné proměnné dokáže uložit několik prvků
- Prvky mohou být různých datových typů
- Prvky nejsou indexovány čísly, ale pojmenovány slovně
- Definice struktury se uvádí v globálním prostoru
- Definujeme pomocí dvojice klíčových slov: **typedef struct**
- Typedef nastavuje, že definujeme datový typ
- Strukct že definovaný datový typ je strukturou
- Název se uvádí velkými písmeny
- K položce struktury přistupujeme pomocí tečky

Struktura – příklad

```
Typedef struct
{
        char jmeno[51];
        int vek;
}UZIVATEL;
```

Struktura – příklad

```
struct UZIVATEL{
    char jmeno[51];
    int vek;
}; //nezapomínat na tento závěrečný středník!
```

Reprezentace v paměti

- Pokud vytvoříme proměnnou datového typu struktury, překladač naalokuje paměť pro všechny členy této struktury
- Členové struktury budou v paměti uloženi ve stejném pořadí, v jakém byli popsáni při deklaraci struktury
- Jednotlivý členové struktury nemusí ležet v paměti přesně za sebou jako prvky pole
- Zjištění velikosti pomocí sizeof

Umístění a platnost struktur

- Lze deklarovat i uvnitř funkcí
- Lze využít pouze v tom scope, kde je deklarována
- Nejčastěji jsou deklarovány v global scope

Inicializace struktury

- Strukturu můžete nainicializovat pomocí složených závorek se seznamem hodnot pro jednotlivé členy struktury:
  struct UZIVATEL Karel {"Karel", 20};
- Stejně jako u polí platí, že hodnoty, které nezadáte, se nainicializují na nulu:
  struct UZIVATEL Karel {}; //jméno i věk budou 0
  struct UZIVATEL Karel {"Karel"}; //věk bude 0
- Abyste si nemuseli pamatovat pořadí členů struktury můžete jednotlivé členy inicializovat explicitně pomocí tečky a názvu daného členu:
  struct UZIVATEL Karel {.vek = 20, .jmeno = "Karel"};
- Nekombinujte inicializaci pomocí pořadí a pomocí názvů členů

Přístup ke členům struktur

- Abychom mohli číst a zapisovat jednotlivé členy struktur, můžeme použít operátor přístupu ke členu (member access operator), který má syntaxi <výraz typu struktura>.<název členu>
- Příklad: UZIVATEL.vek = 22;
- Definice ukazatele na strukturu: struct UZIVATEL *ukazatel;
- Přístupu k členu přes ukazatel (member access through pointer), který má syntaxi <ukazatel na strukturu> -> <název členu>: ukazatel ->vek = 22;

Přístup ke členům struktur

```c
#include <string.h>

//definice struktury
typedef struct //zahájení definice
{
    char jmeno[31]; //textový řetezec
    int vek; //celé čílo
}UZIVATEL; //pojmenování struktury

void main()
{
    UZIVATEL *ukazatel; //ukazatel na stukturu Uzivatel
    //dynamická alokace struktury v paměti
    ukazatel = (UZIVATEL*) malloc(sizeof(UZIVATEL));
    //přítup ke členům struktury
    ukazatel->vek = 2; //přiřazení celého čísla pomocí ukazatele
    strcpy(ukazatel->jmeno, "Michal"); //přiřazení textu pomocí ukazatele
    //vypsání obsahu dynamicky alokované struktury
    printf("Malý %s chodí do školky už ve %d letech", ukazatel->jmeno, ukazatel-
    free(ukazatel);
}
```

Porovnání stukur

- Pokud chceme proměnné struktur porovnávat, musíme tak učinit člen po členu

```
5  //definice struktury
6  typedef struct //zahájení definice
7  {
8      char jmeno[31]; //textový řetezec
9      int vek; //celé čílo
10 }UZIVATEL; //pojmenování struktury
11
12 void main()
13 {
14     UZIVATEL Karel; //inicializace struktury
15     Karel.vek = 20; //přiřazení hodnoty
16     strcpy(Karel.jmeno, "Karel"); //přiřazení stringu jako hodnoty
17     UZIVATEL Martin;
18     Martin.vek = 20;
19     strcpy(Martin.jmeno, "Martin");
20     //porovníní stuktur Karel a Martin
21     if (Karel.jmeno != Martin.jmeno || Karel.vek != Martin.vek){
22         printf("Struktury jsou rozdilne");
23     }else{
24         printf("Struktuy jsou shodne");
25     }
26 }
```

Reprezentace v paměti

- Z důvodu dodržení tzv. zarovnání (alignment) jednotlivých datových typů členů struktury se překladač může rozhodnout mezi členy vložit nějaké byty navíc
- Každý datový typ má krom délky také zarovnání
- Zarovnání *n* říká, že daný datový typ může ležet na adresách, které jsou dělitelné číslem *n*
- Např.: datový typ se zarovnáním 4 může ležet na adresách 4, 8, 12, 200 nebo 512, nikoli na adresách 1, 3 nebo 134
- Důvodem existence zarovnání je efektivita načítání dat procesorem (některé procesory nemusí umět načíst nezarovnaná data)

Reprezentace v paměti

- Primitivní datové typy mají zarovnání stejné, jako je jejich velikost
- Struktury mají zarovnání nastavené na nejvyšší zarovnání ze všech datových členů typů dané struktury
- Zarovnání jednotlivých datových typů ovlivňuje to, jak překladač rozmístí jednotlivé členy struktur v paměti.
- Překladač se snaží o to, aby každý člen struktury ležel na adrese, která bude zarovnaná vzhledem k datovému typu daného členu.

Reprezentace v paměti

> Poznámka: ve všech případech níže budeme předpokládat, že `short` zabírá `2` byty, `int` zabírá `4` byty, a ukazatel zabírá `8` bytů.

```
typedef struct {
    char a;
    int b;
} Str1;
```

Jelikož `char` zabírá `1` byte a `int` zabírá `4` byty, mohli bychom si myslet, že `sizeof(Str1)` bude `5`. Nicméně překladač musí zajistit, že člen `b` bude ležet na adrese, která bude zarovnaná na `4` byty, protože zarovnání datového typu `int` je `4`. Dejme tomu, že by proměnná typu `Str1` ležela třeba na adrese `16`, tj. i člen `a` by ležel na adrese `16`. Pokud by překladač umístil člen `b` na adresu `17`, tak by tento člen ležel na nezarovnané adrese[1]:

Z tohoto důvodu překladač vloží za `a` tři byty tzv. **výplně** (*padding*). Tyto byty nebudou k ničemu využívány, budou sloužit pouze k tomu, aby byl člen `b` správně zarovnaný. Struktura tedy bude v paměti uložena takto, její velikost bude `8` bytů a její zarovnání budou `4` byty:

Funkce pro manipulaci se strukturami

- Snaha o zvýšení čitelnosti kódu
- Jako první parametr přebírají ukazatel na strukturu

```
//funkce pro posunutí vozu
void VUZposun(VUZ* vuz, float sirka, float delka)
{
    vuz->poloha.delka += delka; //přidání zeměpisné délky do polohy vozu
    vuz->poloha.sirka += sirka; //přidání zeměpisné šířky do polohy vozu
}

void main()
{
    VUZ osobak; //vytvoření struktury jménem osobak
    osobak.id = 1; //uložení hodnoty do prvku struktury
    osobak.poloha.delka = 41.40338; //uložení hodnoty do prvku vnořené struktury
    osobak.poloha.sirka = 2.17403;
    printf("Osobni auto má id %d a začíná na poloze %fN, %fW", osobak.id, osobak
    VUZposun(&osobak, 10, 10); //volání funkce s předáním reference na strukturu
    printf("\nOsobni auto má id %d a polohu %fN, %fW", osobak.id, osobak.poloha.
}
```

Struktura jako návratová hodnota

- Řeší problém s navracením většího množství hodnot z funkce

```
6  typedef struct //zahájení definice
7  {
8      int x;
9      int y;
10 }POLOHA; //pojmenování struktury
11
12 //funkce pro generování pseudonáhodné počáteční polohy
13 POLOHA PocatecniPoloha()
14 {
15     POLOHA p; //deklarace struktury typ PLOHA
16     p.x = rand()%100; //náhodná hodnota sořadnice x
17     p.y = rand()%100; //náhodná hodnota souřadnice y
18     return p; //využití struktury jako návratové hodnoty funkce
19 }
20
21 void main()
22 {
23     POLOHA zacatek; //deklarace pstruktury typ PLOHA
24     zacatek = PocatecniPoloha(); //naplnění struktury návratovými hodnotami
25     printf("Hráč začíná na souřadnicích %d, %d", zacatek.x, zacatek.y);
26 }
```

Kopírovaní struktur

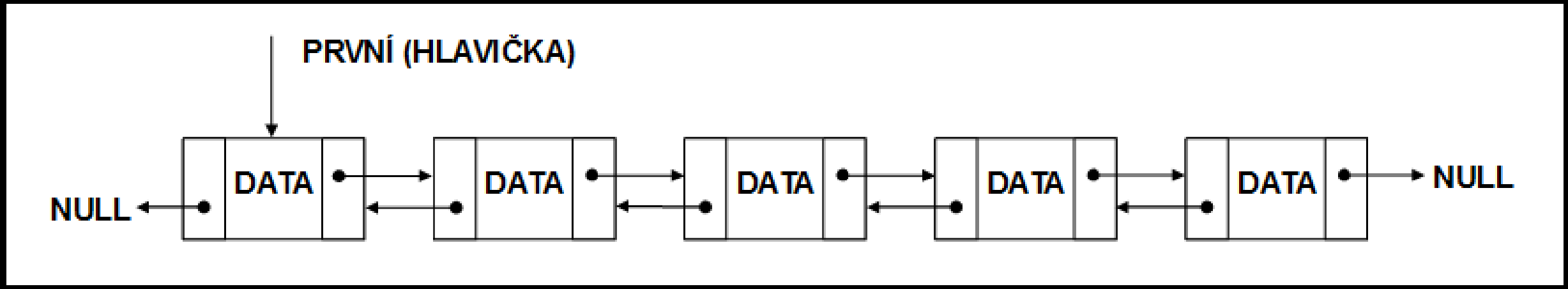
- Struktury stejného datového typu lze kopírovat prostým přiřazením
- Aby je bylo možné jednoduše kopírovat, nesmí obsahovat dynamicky alokovaná data

```
5  //definice struktury
6  typedef struct //zahájení definice
7  {
8      char jmeno[31]; //textový řetezec
9      int vek; //celé čílo
10 }UZIVATEL; //pojmenování struktury
11
12 void main()
13 {
14     UZIVATEL Karel; //inicializace struktury
15     Karel.vek = 20; //přiřazení hodnoty
16     strcpy(Karel.jmeno, "Karel"); //přiřazení stringu jako hodnoty
17     UZIVATEL KarelKopie;
18     KarelKopie = Karel; //Struktura Karel se zkopíruje do struktury KarelKopie
19     //porovníní stuktur Karel a KarelKopie
20     if (Karel.jmeno == KarelKopie.jmeno || Karel.vek == KarelKopie.vek){
21         printf("Struktury jsou shodne");
22     }else{
23         printf("Struktuy jsou rozdilne");
24     }
25 }
```

Spojový seznam

Spojový seznam

- Je dynamickou datovou strukturou
- Jedná se o lineárně uspořádané prvky, které jsou propojeny ukazateli
- Seznam může být jednosměrný (každý prvek má ukazatel na následující prvek) nebo obousměrný (každý prvek má ukazatel na předchozí i následující prvek)



Spojový seznam

- Vstupním bodem do spojového seznamu je ukazatel na jeho první prvek, který se také nazývá hlavička
- Každý prvek v sobě nese datovou hodnotu (například číslo, znak, datovou strukturu…) a ukazatel.
- Definice položky seznamu:
  typedef struct data {

```
  int cislo;
  //další datové položky různých typů
  struct data *nasl, *pred; //ukazatel na následující a předchozí prvek seznamu
  }SEZNAM;
```

Zápis dat do seznamu

1. Inicializace seznamu: SEZNAM *prvni = NULL;
2. Alokování místa pro nový prvek
3. Naplnění datových položek - načtení potřebných informací z klávesnice nebo ze souboru
4. Zařazení nového prvku do obousměrného spojového seznamu

- Nové prvky můžeme přidávat na začátek nebo na konec seznamu, případně je můžeme vkládat na určené místo v seznamu

Zápis dat do seznamu

```
16  //alokace paměti pro nový prvek
17  SEZNAM *novy = (SEZNAM*) malloc(sizeof(SEZNAM));
18  if(novy == NULL) //kontroloa alokování paměti  ···
24  //inicializace ukazatelů
25  novy->pred = NULL;
26  novy->nasl = NULL;
27  //naplneni dat
28  printf("Zadej cele cislo: ");
29  scanf("%d", &novy->cislo);
30  //zařazeni na začátek seznamu
31  if (prvni == NULL) //pokud byl seznam prázdný, zařadí se na první místo
32  {
33    prvni = novy;
34  }
35  else
36  {
37    prvni->pred = novy; //zařazení před první prvek
38    novy->nasl = prvni; //nastavení následujícího záznamu v aktuálně vkládaném p
39    prvni = novy; //kopírování dat
40  }
```

Procházení spojového seznamu

```
SEZNAM *p;
 for( p = prvni; p != NULL; p = p->nasl)
 {
   //zpracování hodnot prvku, na který ukazuje ukazatel p
 }
```

Union

Union

- Používají se ve specifických případech v nízkoúrovňovém kódu
- V daný okamžik existuje v paměti pouze jeden člen unie
- Unie má tak velikost odpovídající prostorově největšímu typu v unii, mění se pouze interpretace dat
- Všechny položky začínají na adrese shodné s počáteční adresou unie

Union

```
5  //definice unie
6  typedef union
7  {
8    int cislo;
9    char znak;
10 } unie; //pojmenování struktury
11 // unie bude mít velikost rovnou sizeof(int)
12
13 void main()
14 {
15   unie ZnakCislo; //inicializace unie
16   ZnakCislo.cislo = 101; // v unii bude číslo 101
17   if (ZnakCislo.znak == 'e')
18   {
19     printf("Do unie bylo ascii kódem vloženo písmeno e");
20   }
21
22 }
```