Vícerozměrná pole, struktury, uniony a výčtové typy

Pavel Čeleda

celeda@liberouter.org

Kurz jazyka C - přednáška č. 7

Vícerozměrná pole

- definice pole

```
int x[2][3];
int vektor[2][2][2];
```

Vícerozměrná pole

- definice pole
  ```
  int x[2][3];
  int vektor[2][2][2];
  ```
- přístup pomocí indexů
  ```
  x[1][0]=1;
  vek[1][0][1]=15;
  ```

Vícerozměrná pole

- definice pole

```
int x[2][3];
int vektor[2][2][2];
```

- přístup pomocí indexů

```
x[1][0]=1;
vek[1][0][1]=15;
```

- uložení pole v paměti

`int x[2][3];` → alokuje paměť 2 * 3 * 2 bajty

100	102	104	106	108	110	112
x[0][0]	x[0][1]	x[0][2]	x[1][0]	x[1][1]	x[1][2]	volno
x[0]			x[1]			
x						

Vícerozměrná pole - pole jako parametr funkce

- uvádí se pouze název pole `int pole[][3]`,

Vícerozměrná pole - pole jako parametr funkce

- uvádí se pouze název pole `int pole[][3]`,
- první parametr dimenze se vynechává (prázdné [])

Vícerozměrná pole - pole jako parametr funkce

- uvádí se pouze název pole `int pole[][3]`,
- první parametr dimenze se vynechává (prázdné [])
- druhá dimenze musí být uvedena jako konstanta ([3])

Vícerozměrná pole - pole jako parametr funkce

- uvádí se pouze název pole `int pole[][3]`,
- první parametr dimenze se vynechává (prázdné [])
- druhá dimenze musí být uvedena jako konstanta ([3])
-
```
double maxim (double pole[][3], int radky)
{
   double pom = pole[0][0];
   int i,j;
   for (i=0; i<radky; i++) {
     for (j=0; j<4; j++) {
       if (pole[i][j] > pom)
          pom = pole[i][j];
       }
     }
   return pom;
}
```

Vícerozměrná pole - inicializace polí

- inicializační hodnoty jsou uzavřeny do závorek { },

Vícerozměrná pole - inicializace polí

- inicializační hodnoty jsou uzavřeny do závorek { },
- `double f[3] = { 1.5, 3.0, 7.6 };`

Vícerozměrná pole - inicializace polí

- inicializační hodnoty jsou uzavřeny do závorek { },
- `double f[3] = { 1.5, 3.0, 7.6 };`
- `double f[] = { 1.5, 3.0, 7.6 };`

Vícerozměrná pole - inicializace polí

- inicializační hodnoty jsou uzavřeny do závorek { },
- `double f[3] = { 1.5, 3.0, 7.6 };`
- `double f[] = { 1.5, 3.0, 7.6 };`
- `double f[3] = { 1.5, 3.0 };`

Vícerozměrná pole - inicializace polí

- inicializační hodnoty jsou uzavřeny do závorek { },
- `double f[3] = { 1.5, 3.0, 7.6 };`
- `double f[] = { 1.5, 3.0, 7.6 };`
- `double f[3] = { 1.5, 3.0 };`
- `double f[3] = { 1.5, 3.0, 7.6, 10 };`

Vícerozměrná pole - inicializace polí

- inicializační hodnoty jsou uzavřeny do závorek { },
- `double f[3] = { 1.5, 3.0, 7.6 };`
- `double f[] = { 1.5, 3.0, 7.6 };`
- `double f[3] = { 1.5, 3.0 };`
- `double f[3] = { 1.5, 3.0, 7.6, 10 };`
- inicializace dvojrozměrného pole

```
double f[][2] =
{
  { 1.5, 3.0 },
  { 1.0, 2.0 },
  { 1.9, 4.0 }
};
```

Vícerozměrná pole - inicializace polí

- inicializační hodnoty jsou uzavřeny do závorek { },
- `double f[3] = { 1.5, 3.0, 7.6 };`
- `double f[] = { 1.5, 3.0, 7.6 };`
- `double f[3] = { 1.5, 3.0 };`
- `double f[3] = { 1.5, 3.0, 7.6, 10 };`
- inicializace dvojrozměrného pole

```
double f[][2] =
{
  { 1.5, 3.0 },
  { 1.0, 2.0 },
  { 1.9, 4.0 }
};
```

- `char *p_pole[] = {"aaa", "bbb", "ccc"};`

Parametry funkce main

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;

  printf("Prikazova radka ma %d retezcu\n", argc);

  for(i=0; i<argc, i++)
   printf("%s\n", argv[i]);

  return 0;
}
```

Struktury a uniony

- *pole* - homogenní datový typ (všechny prvky stejného datového typu),

Struktury a uniony

- *pole* - homogenní datový typ (všechny prvky stejného datového typu),
- *struktura* - heterogenní datový typ (složeno z různých datových typů), navenek však struktura vystupuje jako jednolitý objekt,

Struktury a uniony

- *pole* - homogenní datový typ (všechny prvky stejného datového typu),
- *struktura* - heterogenní datový typ (složeno z různých datových typů), navenek však struktura vystupuje jako jednolitý objekt,
- zápis struktury v jazyce C

```
struct {
   položky;
}a;
```

Definice struktury

1) nepojmenovaná struktura

```
struct {
    int vyska;
    float vaha;
} pavel, honza, karel;
```

Definice struktury

1) nepojmenovaná struktura

```
struct {
    int vyska;
    float vaha;
} pavel, honza, karel;
```

2) pojmenovaná struktura - **miry**

```
struct miry {
    int vyska;
    float vaha;
} pavel, honza, karel;
```

Definice struktury

1) nepojmenovaná struktura

```
struct {
    int vyska;
    float vaha;
} pavel, honza, karel;
```

2) pojmenovaná struktura - **miry**

```
struct miry {
    int vyska;
    float vaha;
} pavel, honza, karel;
```

3) pojmenovaná struktura s oddělenou definicí struktury **miry** a proměnných **pavel, honza, karel**

```
struct miry {
    int vyska;
    float vaha;
};
struct miry pavel, honza, karel;
```

Definice struktury

4) definice nového datového typu **MIRY** příkazem **typedef**

```
typedef struct{
    int vyska;
    float vaha;
} MIRY;
MIRY pavel, honza, karel;
```

Definice struktury

4) definice nového datového typu **MIRY** příkazem **typedef**

```
typedef struct{
    int vyska;
    float vaha;
} MIRY;
MIRY pavel, honza, karel;
```

5) pojmenování nového datového typu a struktury

```
typedef struct miry {
    int vyska;
    float vaha;
} MIRY;
MIRY pavel, honza, karel;
```

Přístup k prvkům struktury

- přístup pomocí *tečková notace*

```
pavel.vyska = 186;
karel.vaha = 89.5;
honza.vyska = pavel.vyska
```

Přístup k prvkům struktury

- přístup pomocí *tečková notace*

```
pavel.vyska = 186;
karel.vaha = 89.5;
honza.vyska = pavel.vyska
```

- pole struktur

```
MIRY lide[100]; ↔ struct miry lide[100];
lide[50].vyska = 176;
```

Přístup k prvkům struktury

- přístup pomocí *tečková notace*

```
pavel.vyska = 186;
karel.vaha = 89.5;
honza.vyska = pavel.vyska
```

- pole struktur

```
MIRY lide[100]; ↔ struct miry lide[100];
lide[50].vyska = 176;
```

- lze pracovat s celou strukturou najednou

```
pavel = honza;
```

Struktury a pointery

- použití pointerů na struktury:

Struktury a pointery

- použití pointerů na struktury:
  - práce se strukturami v dynamické paměti,

Struktury a pointery

- použití pointerů na struktury:
  - práce se strukturami v dynamické paměti,
  - práce se strukturami ve funkcích.

Struktury a pointery

- použití pointerů na struktury:
  - práce se strukturami v dynamické paměti,
  - práce se strukturami ve funkcích.
- definice pointeru na strukturu

```
typedef struct {
    char jmeno[30];
    int rocnik;
} STUDENT;

STUDENT s, *p_s;

p_s = (STUDENT *) malloc (sizeof(STUDENT));

p_s = &s;
```

Přístup k prvkům struktury

- `STUDENT s, *p_s = &s;`

Přístup k prvkům struktury

- `STUDENT s, *p_s = &s;`
- pomocí jména struktury `s` ... s.rocnik = 3;

Přístup k prvkům struktury

- `STUDENT s, *p_s = &s;`
- pomocí jména struktury `s` . . . s.rocnik = 3;
- pomocí pointeru `p_s` . . . (*p_s).rocnik = 4;

Přístup k prvkům struktury

- `STUDENT s, *p_s = &s;`
- pomocí jména struktury `s` . . . s.rocnik = 3;
- pomocí pointeru `p_s` . . . (*p_s).rocnik = 4;
- pomocí pointeru `p_s` . . . p_s->rocnik = 4;

Přístup k prvkům struktury

- `STUDENT s, *p_s = &s;`
- pomocí jména struktury `s` . . . s.rocnik = 3;
- pomocí pointeru `p_s` . . . (*p_s).rocnik = 4;
- pomocí pointeru `p_s` . . . p_s->rocnik = 4;

Přístup k prvkům struktury

- `STUDENT s, *p_s = &s;`
- pomocí jména struktury `s` ... s.rocnik = 3;
- pomocí pointeru `p_s` ... (*p_s).rocnik = 4;
- pomocí pointeru `p_s` ... p_s->rocnik = 4;

- velikost struktury vždy určujeme pomocí operátoru **sizeof**

Přístup k prvkům struktury

- `STUDENT s, *p_s = &s;`
- pomocí jména struktury `s` ... s.rocnik = 3;
- pomocí pointeru `p_s` ... (*p_s).rocnik = 4;
- pomocí pointeru `p_s` ... p_s->rocnik = 4;

- velikost struktury vždy určujeme pomocí operátoru **sizeof**
- k prvkům struktury vždy přistupujeme přes `.` nebo `->`

Struktury a funkce

- funkce pracuje se strukturou stejně jako s jinačím datovým typem např. int

```
typedef struct {
  double re, im;
} KOMP;

KOMP secti (KOMP a, KOMP b)
{
  KOMP c;
  c.re = a.re + b.re;
  c.im = a.im + b.im;
  return c;
}

int main(void)
{
  KOMP x, y, z;
  x.re = 1.11; x.im = 3.14;
  y = x;
  z = secti(x,y);
  return 0;
}
```

Struktury a funkce

```
typedef struct {
  double re, im;
} KOMP;

void secti (KOMP *a, KOMP *b, KOMP *c)
{
  c->re = a->re + b->re;
  c->im = a->im + b->im;
}

int main(void)
{
  KOMP x, y, z;
  x.re = 1.11; x.im = 3.14;
  y = x;
  secti(&x,&y,&z);
  return 0;
}
```

Výčtový typ - enumerate typ

- zpřehledňuje program a zvyšuje modularitu,

Výčtový typ - enumerate typ

- zpřehledňuje program a zvyšuje modularitu,
- umožňuje definovat seznam symbolických konstant,

Výčtový typ - enumerate typ

- zpřehledňuje program a zvyšuje modularitu,
- umožňuje definovat seznam symbolických konstant,
- ```
  typedef enum {
     MODRA, CERVENA, ZELENA, ZLUTA
  } BARVY;

  BARVY c,d;
  c = MODRA;
  d = CERVENA;
  ```
  ```

Výčtový typ - enumerate typ

- zpřehledňuje program a zvyšuje modularitu,
- umožňuje definovat seznam symbolických konstant,
- ```
  typedef enum {
     MODRA, CERVENA, ZELENA, ZLUTA
  } BARVY;

  BARVY c,d;
  c = MODRA;
  d = CERVENA;
  ```
- ```
  typedef enum {
     FALSE, TRUE
  } BOOLEAN;
  ```

Uniony

- vyhradí se paměť pro největší položku ze všech položek v unionu definovaných,

Uniony

- vyhradí se paměť pro největší položku ze všech položek v unionu definovaných,
- všechny položky se v unionu překrývají,

Uniony

- vyhradí se paměť pro největší položku ze všech položek v unionu definovaných,
- všechny položky se v unionu překrývají,
- v unionu může být v jednom okamžiku pouze jedna položka,

Uniony

- vyhradí se paměť pro největší položku ze všech položek v unionu definovaných,
- všechny položky se v unionu překrývají,
- v unionu může být v jednom okamžiku pouze jedna položka,
- v praxi se uniony používají velice zřídka,

Uniony

- vyhradí se paměť pro největší položku ze všech položek v unionu definovaných,
- všechny položky se v unionu překrývají,
- v unionu může být v jednom okamžiku pouze jedna položka,
- v praxi se uniony používají velice zřídka,
- ```
  union typ {
  varianta_1;
  ...
  varianta_n;
  ```
  ```

Uniony

```
typedef union {
   char c;
   int i;
   float f;
} ZN_INT_FLT;

ZN_INT_FLT a, *p_a = &a;

a.c = '#';
p_a->i = 1;  /* premaze znak '#' */
a.f = 2.3;   /* premaze cislo 1  */
```

Bitové operace a bitové pole

&	bitový součin - AND
|	bitový součet - OR
^	bitový exklusivní součet - XOR
«	posun doleva
»	posun doprava
∼	jedničkový doplněk - negace bit po bitu - unární operátor

Základní bitové operace

- bitový součin - c = c & 0x7F;

Základní bitové operace

- bitový součin - c = c & 0x7F;
- bitový součet - c = c | 0x20;

Základní bitové operace

- bitový součin - c = c & 0x7F;
- bitový součet - c = c | 0x20;
- bitový exkluzivní součet - if (x ^ y) /* true pokud jsou cisla rozdilna */

Základní bitové operace

- bitový součin - c = c & 0x7F;
- bitový součet - c = c | 0x20;
- bitový exkluzivní součet - if (x ^ y) /* true pokud jsou cisla rozdilna */
- posun doleva - x «=3; /* nasobeni osmi */

Základní bitové operace

- bitový součin - c = c & 0x7F;
- bitový součet - c = c | 0x20;
- bitový exkluzivní součet - if (x ^ y) /* true pokud jsou cisla rozdilna */
- posun doleva - x «=3; /* nasobeni osmi */
- posun doprav - x »=3; /* deleni osmi */

Základní bitové operace

- bitový součin - c = c & 0x7F;
- bitový součet - c = c | 0x20;
- bitový exkluzivní součet - if (x ^ y) /* true pokud jsou cisla rozdilna */
- posun doleva - x «=3; /* nasobeni osmi */
- posun doprav - x »=3; /* deleni osmi */
- negace bit po bitu - x &= ∼0xF; /* nastavi na nulu nejnizsi 4 bity */

Bitové pole

- struktura s velikostí typu int

Bitové pole

- struktura s velikostí typu int
- nejmenší položka v bitovém poli je 1 bit

Bitové pole

- struktura s velikostí typu int
- nejmenší položka v bitovém poli je 1 bit
- ```
  typedef struct {
     unsigned den : 5; /* bity 0 - 4 */
     unsigned mes : 4; /* bity 5 - 8 */
     unsigned rok : 7; /* bity 9 - 15 */
  } DATUM;
  ```
  ```

Bitové pole

- struktura s velikostí typu int
- nejmenší položka v bitovém poli je 1 bit
- ```
  typedef struct {
     unsigned den : 5; /* bity 0 - 4 */
     unsigned mes : 4; /* bity 5 - 8 */
     unsigned rok : 7; /* bity 9 - 15 */
  } DATUM;
  ```
- ```
  DATUM dnes, zitra;
  dnes.den = 25;
  dnes.mes = 6;
  dnes.rok = 1992 - 1980;
  zitra.den = dnes.den + 1;
  ```