

Jazyk C

Struktury, Uniony a výčet

3) **pojmenovaná struktura**, oddělena definice od proměnných, definice lze provádět vícekrát (v 1) nelze).

```
struct miry {           /*definice struktury*/
    int vyska;
    float vaha;
};
```

```
struct miry honza:      /*definice promennych */
struct miry pavel, karel;
```

POZOR! Toto nelze!!!
miry pavel, karel;

Struktury

- **Heterogenní datový typ** – může být složen z libovolných datových prvků různých typů – vnitřní záležitost, navenek vystupuje jako .jednotlivý objekt.

4) **definice nového typu struktury, nepojmenovaná**, typ pojmenovaný je – použití definice prom., přetypování

```
typedef struct { /*definice typu
                                struktury*/
    int vyska;
    float vaha;
} MIRY;
```

```
MIRY honza, pavel, karel; /*definice
                                promennych */
```

POZOR !!! Zde není potřeba použít slova **struct**.

Struktury - definice

Pět způsobů definice struktury:

1) struktura **není pojmenovaná**, dají se použít jenom definované proměnné

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

2) **pojmenovaná** struktura

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

5) **definice nového typu struktury, pojmenovaná**, typ pojmenovaný je – použití když odkazuje struktura sama na sebe

```
typedef struct miry { /*definice typu
                                struktury*/
    int vyska;
    float vaha;
} MIRY;
```

```
MIRY honza, pavel, karel; /*definice
                                promennych */
```

Přístup k prvkům struktury: tečková konvence

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

Struktura v jiné struktuře

(Vhnížděná struktura)

```
typedef struct {  
    char ulice[30]; /*retezec  
    int cislo;      ukoncen'\0'*/  
}ADRESA;  
  
typedef struct {  
    char jmeno[20];/*retezec  
    ADRESA adresa; ukoncen'\0'*/  
    float plat;  
} OSOBA;  
  
OSOBA lide[1000];
```

Struktury a pointery

Pointery na struktury mají dvě oblasti použití:

- Při práci se strukturami v dynamické paměti.
- Při práci se strukturou ve funkci.

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

```
STUDENT s, *ps;  
PSTUDENT ps2;
```

s – struktura typu student

ps – pointer na strukturu typu student, nemá zatím přidělenou paměť

Přístup k prvkům struktur:

```
strcpy(lide[0].adresa.ulice, "Kratka");  
lide[0].adresa.cislo = 666.
```

Nebo

```
OSOBA *pkdo = lide;  
strcpy(pkdo->adresa.ulice, "Kratka");  
pkdo->adresa.cislo = 666.
```

Přidělení paměti pointeru:

```
ps = (STUDENT *) malloc(sizeof(STUDENT));
```

Nebo

```
ps = &s;
```

Pak lze do struktury s přistupovat:

Přes jméno struktury s.rocnik = 3;

Komplikovaně point. (*ps).rocnik = 4;

Jednoduše point. ps->rocnik = 4;

Struktury a funkce

K&R

- Funkce může vracet pointer na strukturu.
- Struktura jako parametr funkce – voláním odkazem (pointer).

ANSI

- Funkce může vracet strukturu.
- Struktura jako skutečný parametr funkce - hodnotou.

Funkce pro sčítání komplexních čísel:

```
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);
}
```

Jak se dá přepsat tato funkce pomocí pointerů?

Inicializace struktur

```
typedef struct {
    int i, j;
    float f;
} PRIKLAD;

PRIKLAD a = {1, 2, 3.14};

Pole struktur:
PRIKLAD b[] = {
    {4, 5, 4.44}
    {2, 8, 9.6}
    {1, 1, 1.0}
};
```

Volání:

```
main()
{
    KOMP x,y,z;
    x.re = 1.1; x.im = 3.14;
    y = x;

    z = secti(x,y);
}
```

Počet prvků – jednotlivých struktur:

```
pocet = sizeof(b) / sizeof(PRIKLAD);
```

Pomocí pointerů:

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

main()
{
    KOMP x,y,z;
    x.re = 1.1; x.im = 3.14;
    y = x;

    secti(&x, &y, &z);
}
```

Výčtový typ

- Časté použití, zpřehledňuje program.
- Snadná definice seznamu symbolických konstant, které jsou na sobě závislé -> Proto je píšeme **velkými** písmeny.
- Položky výčtového typu **nejsou** L-hodnoty!
- Můžeme přiřadit **explicitně** nějakou hodnotu, pak následující neinicializované mají hodnotu o 1 větší než předchozí. Jinak mají implicitně 0, 1, 2 ...
- Nelze tisknout jméno položky výčtu jako

Definice výčtu

```
typedef enum {
    MODRA, CERVENA, ZELENA, ZLUTA
} BARVY;

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

Union

- NENÍ Variantní záznam z Pascalu!!!
- Vyhradí se **paměť pro největší položku** unionu.
- V unionu může být pouze **jedna položka v jednom okamžiku**.
- Používá se málo.
- Pro šetření paměti, například u velkých polí.
- Přístup k položkám je stejný jako u struktury.

Příklad použití

```
#define FALSE 0
#define TRUE 1
```

Ale lépe:

```
typedef enum {
    FALSE, TRUE;
} BOOLEAN;
```

Použití:

```
if(isdigit(c) == FALSE)
```

```
typedef union {
    Společné položky nejsou!!!
    Varianta_1;
    Varianta_2;
    ...
    Varianta_n;
} CISLA;
```

Explicitní inicializace:

```
typedef enum {
    MODRA = 0,
    CERVENA = 4,
    ZELENA = 2,
    ZLUTA
} BARVY;
```

Jakou hodnotu bude mít ZLUTA?

Takto neinicializovat!!!

Když už tak všechny a popořadě vzestupně!

Lepší union

```
typedef enum{
    CISLO = 'I',
    ZNAK = 'C'
} TYP;

typedef union {
    char c;
    int i;
} CHARINT;

typedef struct{
    TYP typ;
    CHARINT hodnota;
}PRVEK;

PRVEK pole[10];
```

„Variantní záznam“ v C

Pascal:

```
TYPE ABC = record
  a, b : integer;

  Case boolean of
    TRUE : (i,j: integer);
    FALSE : (f,g :real);
end;
```

C: v hodnou kombinací struktur, unionu a výčtu se dá vytvořit obdoba Variantního záznamu z Pascalu

```
typedef enum {
  FALSE, TRUE
} BOOLEAN;

typedef union {
  struct{
    int i, j;
  }v1;
  struct{
    float f, g;
  }v2;
}VARIANTY;
```

```
typedef struct{
  int a, b; /*spolecne polozky*/
  BOOLEAN cele; /*rozhodovaci prvek*/
  VARIANTY var;
}ABC;
```

Definice proměnné a přístup k prvkům:

```
ABC x;
x.a = 1;
x.cele = TRUE;
x.var.v1.i = 3;
x.cele = FALSE;
x.var.v2.f = 5.6;
```