



Výčtový typ, strukturované datové typy

Jitka Kreslíková, Aleš Smrčka
2023

Fakulta informačních technologií
Vysoké učení technické v Brně

IZP – Základy programování



Výčtový typ, strukturované datové typy

- ☐ Specifikátor **typedef**
- ☐ Výčtový typ
- ☐ Datový typ struktura



Specifikátor **typedef**

☐ Klíčové slovo **typedef**

- Vytvoří nové označení datového typu
- Zejména pro složité datové typy (struktury, ukazatele na funkce)
- Zpřehledňuje program

☐ Specifikátor

- Specifikuje nové jméno typu

☐ Operátor ?

- Ve výrazech, výsledkem je hodnota
- **typedef NENÍ** operátor 



Specifikátor **typedef**

❑ Obecný formát:

typedef specifikace_typu nový_identifikátor;

// málo používané (a málo vhodné)

typedef int ROCNIK;

ROCNIK a,b,c;

typedef int *P_INT;

P_INT p1,p2,p3;

typedef int POLE[10];

POLE x,y,z;

// častější použití

typedef double (*FUN)(double, int);

// FUN je ukazatel na funkci se dvěma parametry

// (specifikovanými pro předávání hodnotou), která

// vrací hodnotu typu double

FUN f1,f2;



Specifikátor **typedef**

- ❑ Často používán se strukturami

```
typedef struct tosoba  
{  
    char *jmeno;  
    char *prijmeni;  
    float vyska;  
    float sirka;  
} TOsoba;  
TOsoba clovek;  
clovek.vyska=180.0;
```

- ❑ Nový identifikátor lze použít i v jiné specifikaci typedef

```
typedef int rozmer;  
typedef rozmer delka;  
typedef delka hloubka;  
hloubka d;
```

Velmi nepřehledné

❗ raději nepoužívat ❗



Specifikátor **typedef**

- ❑ Nevytváří nový datový typ
- ❑ Vytváří „zkratku“ pro původní datový typ.
- ❑ Nový identifikátor
 - Stejná pravidla pro rozsah platnosti jako pro ostatní identifikátory.
 - Lze vytvořit i uvnitř libovolně zanořeného bloku.



Výčtový typ

□ Obecně

- Sada hodnot vyjmenovaných programátorem (enumerace)

□ V jazyce C

- Typ **enum**
- Sada pojmenovaných ***celočíselných*** konstant
- Kompatibilní s celočíselnými typy
- Typová kontrola vzhledem k číselným typům
→ nemá smysl vytvářet proměnné



Výčtový typ

[C99]

□ Syntaxe

enum-specifier:

enum *identifier*_{opt} { *enumerator-list* }

enum *identifier*_{opt} { *enumerator-list* , }

enum *identifier*

enumerator-list:

enumerator

enumerator-list , *enumerator*

enumerator:

enumeration-constant

enumeration-constant = *constant-expression*



Výčtový typ

□ Obecný formát

enum jméno_výčtu {seznam_položek} seznam_proměnných

```
enum typ_barvy {CERVENA, ZELENA, ZLUTA};
```

```
enum {CERVENA, ZELENA, ZLUTA} mojeBarva;  
mojeBarva = ZLUTA; // přiřadí hodnotu 2
```

□ Konstanty výčtového typu

- Hodnoty standardně od nuly
- Lze přiřadit vlastní hodnoty
- Jako jiné konstanty



Výčtový typ

Příklad: zobrazení konstanty výčtového typu.

```
enum computer{KLAVESNICE,CPU,OBRAZOVKA,TISKARNA};  
enum computer comp = CPU;  
printf("%d\n", comp);    // zobrazí 1
```

Příklad: výpis řetězcového ekvivalentu výčtové konstanty.

```
enum typ_doprava {AUTO, VLAK, LETADLO, AUTOBUS};  
void vypisVozidlo(int vozidlo)  
{  
    if (vozidlo<AUTO || vozidlo>AUTOBUS) return;  
    const char *trans[] =  
        {"auto", "vlak", "letadlo", "autobus"};  
    printf("%s", trans[vozidlo]);  
}
```



Výčtový typ

- ❑ Identifikátory konstant nelze načítat nebo vypisovat pomocí knihovních funkcí

Příklad: vstup hodnot výčtového typu.

```
enum numbers {NULA, JEDNA, DVE, TRI} num;  
printf("Zadejte cislo: ");  
! scanf("%d", &num); // nelze zadat např. JEDNA
```

Příklad: hodnotu konstanty lze změnit zadáním explicitní hodnoty při deklaraci.

```
enum typ_barvy {CERVENA, ZELENA=9, ZLUTA} barva;  
// ZLUTA bude mít nyní hodnotu 10
```



Výčtový typ

- ❑ Součástí jména typu je i klíčové slovo **enum** (tag)
- ❑ Vytváření proměnných nemá příliš smysl

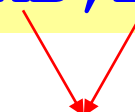
```
enum typ_barvy mojeBarva;
```



```
mojeBarva = VLAK; //! toto je v C legální
```

Příklad: Konstanty výčtového typu mohou označovat stejné hodnoty.

```
enum krev_skupiny{NULA,A,B,AB,BA=3};
```



hodnota 3



Datový typ struktura

- ☐ Definice struktury
- ☐ Kompatibilita struktur
- ☐ Inicializace struktur
- ☐ Ukazatele na struktury
- ☐ Alokace struktury
- ☐ Struktura odkazující na sebe
- ☐ Struktury a pole
- ☐ Vnořené struktury
- ☐ Struktury a funkce



Datový typ struktura

- ❑ Heterogenní datový typ
 - Může obsahovat položky různých typů
- ❑ Obecný formát:

```
struct jméno_typu  
{  
    typ prvek1;  
    typ prvek2;  
    .  
    .  
    typ prvekN;  
} seznam_proměnných;
```

některá může chybět

Úkol: vyzkoušejte možné varianty
[HePa13], str. 233



Definice struktury

Příklad: definice proměnných typu struktura.

```
struct osoba {  
    int vek;  
    int vyska;  
    double vaha;  
} c1, c2, c3;
```

Příklad: součástí jména datového typu v deklaracích proměnných je i klíčové slovo **struct** (tag).

```
struct osoba c4, c5, c6;  
osoba c7; // syntaktická chyba!
```



Definice struktury

Příklad: Jinou možností je zavedení nového označení datového typu pomocí **typedef**:

```
typedef struct osoba
{
    int vek;
    int vyska;
    double vaha;
} TOsoba;
```

Příklad: deklaraci proměnných *c1,c2,c3* lze pomocí nově zavedeného typu *TOsoba* zapsat takto:

```
TOsoba c1,c2,c3;
```




Struktury – přístup ke složkám

Příklad: přístup k jednotlivým položkám struktury pomocí tečkového operátoru (tečková notace):

```
c1.věk=69;           c1.vyska=182;  
c1.vaha=81.4;        c2.vyska=c1.vyska;
```

Příklad: pro načtení hodnot do struktury po složkách je možné použít **scanf()**:

```
scanf ("%d%d%lf", &c3.věk, &c3.vyska, &c3.vaha);
```

Příklad: pro tisk hodnot po složkách můžeme použít: **printf()**:

```
printf ("věk: %d vyska: %d vaha: %f\n",  
        c3.věk, c3.vyska, c3.vaha);
```



Kompatibilita struktur

- ❑ Struktury (typy) se stejnými položkami
 - Nekompatibilní (kvůli zarovnávání)
 - Proměnné těchto různých struktur nelze **přiřadit** ani **přetypovat**
 - Nelze zaměnit v parametrech funkcí (jako pole)

Příklad: nekompatibilita struktur.

```
typedef struct structa { int a; } Ta;  
typedef struct structb { int a; } Tb;  
Ta a = {1};      // inicializace  
Tb b = a;       // chyba!  
Tb b = (Tb) a;  // chyba!
```



Inicializace struktury

Příklad: inicializátor je podobný jako u polí, nová norma zavádí vylepšení.

```
TOsoba c4 = {25, 182, 82.5};  
// nově podle ISO C99  
TOsoba c5 = {.vek=25, .vyska=182, .vaha=82.5};
```

Příklad: podobně lze inicializovat i pole struktur:

```
TOsoba studenti[] = {  
    {24, 185, 87.5}, {25, 178, 82.8}, {26, 176, 78.8}  
};  
  
printf("%d\n", studenti[0].vek);
```



Kopie proměnných

☐ Proměnné **stejného** typu

- Lze přiřadit jako celek

Příklad: vzájemné přiřazení proměnných typu struktura.

```
c2=c4;  
c2=studenti[2];
```

☐ Provádí se **mělká kopie**

- Položky – ukazatele – kopíruje se jen adresa

☐ Hluboká kopie

- Vše co je za ukazatelem
- Nutno „ručně“ alokovat nový prostor a ručně kopírovat.



Ukazatele na struktury

Příklad: ukazatele na proměnné typu struktura.

```
TOsoba *p_c1=&c1, *p_c2=&c2;
```

□ Přístup k položkám přes ukazatel

■ Operátorem **->** (lepší)

Přístupový operátor

```
p_c1->vek=69;  
p_c1->vyska=182;  
p_c1->vaha=81.4;  
p_c2->vyska=p_c1->vyska;
```

■ Kombinací * a . (nutno závorkovat)

```
(*p_c1).vek=69;
```



Alokace struktury

- ❑ Při alokaci nezapomínat na **sizeof**!

```
TOsoba *novyClovek=malloc ( sizeof (TOsoba) ) ;
```

- ❑ POZOR na zarovnávání!
 - Součet velikostí položek nemusí být shodný s velikostí celé struktury
 - Zarovnání se používá kvůli efektivnějšímu přístupu k položkám

`sizeof(int)+sizeof(int)+sizeof(double)!=sizeof(TOsoba)`



Struktura odkazující sama na sebe

- ❑ Identifikátor vytvořený pomocí **typedef**
 - Nelze použít, zatím neexistuje
- ❑ Skutečný název typu
 - struct jmeno

```
typedef struct tpolozka
{
    int data;
    struct tpolozka *dalsi;
//    TPolozka *dalsi; -- častá chyba, nelze
} TPolozka;
```

- ❑ Využití – seznam, fronta, strom, ...



Pole jako položka struktury

Příklad: Definujme proměnné *p1*, *p2* jako strukturu *TPacient* obsahující datové položky *jmeno*, *prijmeni*, *vyska*, *vaha*, *teplota*, charakterizující zdravotní stav pacienta v nemocnici po *100* dní.

```
typedef struct pacient
{
    char *jmeno;
    char *prijmeni;
    double vyska;
    double vaha[100];
    double teplota[100];
} TPacient;
TPacient p1, p2;
```




Pole jako položka struktury

Příklad: zobrazení teploty pacienta 14. den pobytu v nemocnici.

```
// inicializace
...
int den=14;
printf("%s\t %d.den teplota: %f",
      p1.prijmeni,
      den,
      p1.teplota[den-1]);
```



Pole struktur

```
TPacient pacienti[200];
```

Příklad: (po inicializaci) zobrazení příjmení 25. pacienta a jeho teplota 4. den pobytu v nemocnici:

```
int pacient=25-1, den=4;  
printf("%s %d.den teplota: %f",  
       pacienti[pacient].prijmeni,  
       den,  
       pacienti[pacient].teplota[den-1]);
```



Vnořená struktura

```
typedef enum mesic {LEDEN=1, UNOR, BREZEN, // atd.  
} TMesic;
```

```
typedef struct datum {  
    int den, rok;  
    TMesic mes;  
} TDatum;
```

student.datum.mes=SRPEN;

```
typedef struct tstudent {  
    char jmeno[30], prijmeni[30];  
    TDatum narozeni;  
    int rocnik;  
} TStudent;  
TStudent student;
```



Struktury a funkce

- ❑ Struktura může být parametrem funkce i návratovou hodnotou (ISO C99)

```
typedef struct tmatrix {  
    int rows, cols, **matrix;  
} TMatrix;
```

```
TMatrix allocMatrix(int rows, int cols)  
{  
    TMatrix m = { .rows = rows, .cols = cols,  
        .matrix = NULL };  
    ... //alokace  
    return m;  
}
```



Struktury a funkce

Příklad: struktura jako parametr funkce.

```
void printMatrix(TMatrix m)
{
    for(int r = 0; r < m.rows; r++)
    {
        for(int s = 0; s < m.cols; s++)
        { printf("%d ", m.matrix[r][s]); }
        printf("\n");
    }
}
```



Struktury a funkce

❑ Předávání struktury

- Hodnotou – neefektivní, raději přes ukazatel

```
int addMatrix(TMatrix *dest,  
             const TMatrix *add)  
{  
    if (dest->rows != add->rows ||  
        dest->cols != add->cols)  
    {  
        return ERR_INCOMPATIBLE;  
    }  
    ... // výpočet  
}
```



Struktury a funkce

Příklad: předávání struktury odkazem.

```
TMatrix m1 = allocMatrix(17, 4);  
TMatrix m2 = allocMatrix(17, 4);  
... inicializace  
addMatrix(&m1, &m2); // musí zde být &  
printMatrix(m1);      // zde & být nesmí
```

DP8 Výčtový typ, strukturované datové typy





Kontrolní otázky

1. K čemu slouží klíčové slovo typedef?
2. K čemu v jazyce C slouží datový typ enum?
3. Co je datový typ struktura?
4. Je následující zápis v pořádku? Proč?

typedef balance float;



Úkoly k procvičení

1. Vytvořte výčtový typ obsahující dny v týdnu.
2. Deklarujte strukturu, jež bude obsahovat ukazatel na sebe samu.
3. Deklarujte pole ukazatelů na struktury obsahující údaje o osobách (jméno, rok narození, výšku, váhu). Vhodně využijte specifikátor `typedef`. Vytvořte proměnnou typu pole a naplňte první prvek hodnotou s údaji o své osobě.
4. Vytvořte pole údajů o osobách. U každé osoby je potřeba zaznamenat její jméno, rok narození, výšku a váhu. Vytvořte funkce pro vložení údajů o jedné osobě na zadané místo v poli a inverzní funkci pro získání údajů ze zadané pozice v poli. Vytvořte pole o deseti prvcích a pomocí těchto operací je inicializujte a vypište.