8 Výčtový typ, strukturované datové typy

Kreslíková, 9. 10. 2023

8 Výčtový typ, strukturované datové typy 1
8.1 Úvod 2
8.2 Specifikátor typedef 2
8.3 Výčtový typ 3
8.4 Datový typ struktura 6
8.4.1 Definice struktury, položky struktury 6
8.4.2 Kompatibilita struktur 7
8.4.3 Inicializace struktury 7
8.4.4 Ukazatele na struktury 8
8.4.5 Alokace struktury 8
8.4.6 Struktura odkazující sama na sebe 8
8.4.7 Pole jako položka struktury 9
8.4.8 Pole struktur 9
8.4.9 Vnořená struktura 10
8.4.10 Struktury a funkce 10
8.5 Shrnutí 11
8.6 Úlohy k procvičení 12
8.7 Kontrolní otázky 12

8.1 Úvod

Tato kapitola se zabývá nejkomplexnějšími datovými typy, které může programátor při programování použít, tedy strukturami. Tyto datové typy se tvoří skládáním ze všech datových typů, které jsme již probrali, a umožňují pracovat s nejvyšší mírou abstrakce. Zde probereme, jak se tyto typy tvoří a jak se zachází s proměnnými. Klíčovou úlohu budou hrát zejména v kapitolách o dynamických datových strukturách.

Druhá část této kapitoly se zabývá standardním vstupem a výstupem programu a soubory. V jazyce C nejsou soubory součástí jazyka, ale standardní knihovny, proto se podíváme na funkce, pomocí nichž se soubory pracujeme.

8.2 Specifikátor typedef

Klíčové slovo **typedef** je určeno pro vytvoření nového označení určitého datového typu. Používá se pro zvýšení přehlednosti programu. Při definicích jednoduchých datových typů, ukazatelů a polí se příliš nepoužívá, protože tyto typy jsou samopopisné. Jeho použití je daleko častější u složitějších datových typů jako jsou struktury nebo ukazatele na funkce.

Pozor, **typedef** není operátor, jak se často mylně předpokládá. Jde o ***specifikátor***. Operátory lze používat ve výrazech a jejich aplikací vždy vzniká nějaká použitelná hodnota. Specifikátor **typedef** této definici v žádném případě nevyhovuje, protože neprodukuje žádnou hodnotu.

Obecný formát:

typedef *specifikace_typu nový_identifikátor;*

Příklad: Použití specifikátoru **typedef**.

```
// málo používané (a málo vhodné)
typedef int ROCNIK;    typedef int *P_INT;      typedef int
POLE[10];
ROCNIK a,b,c;          P_INT p1,p2,p3;          POLE x,y,z;

// častější použití
typedef double (*FUN)(double, int);
FUN f1,f2;

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

V prvním příkladu bylo vytvořeno nové jméno *ROCNIK* pro označení datového typu **int** a pomocí tohoto jména byly definovány proměnné *a,b,c*. Ve druhém případě byly proměnné *p1,p2,p3* definovány jako ukazatele na typ **int** pomocí nově vytvořeného označení *P_INT* tohoto typu. Ve třetím případě byly definovány proměnné *x,y,z* jako jednorozměrná pole typu **int** obsahující 10 prvků. V posledním případě byl definován typ ***ukazatel na funkci***

vracející hodnotu double s parametry typu double a **int**. Poslední příklad ukazuje nejčastější použití **typedef** – pojmenování datového typu struktura (**struct**). O tomto datovém typu budeme hovořit později.

Nové jméno vytvořené jedním **typedef** může být použito v dalším **typedef** pro vytvoření jiného jména.

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

Nejde o nový datový typ v pravém slova smyslu. Přesnější bude, když řekneme, že nový identifikátor tvoří „zkratku" pro původní datový typ. V tomto případě jsou typy *rozmer, delka, hloubka* a *int* zcela kompatibilní. Tato vlastnost specifikátoru **typedef**, kdy nově vytvořené „zkratky" jsou kompatibilní s původním datovým typem platí pro všechny datové typy s výjimkou struktur (viz kap. 8.4)

Pravidla pro rozsah platnosti nového identifikátoru vytvořeného pomocí **typedef** jsou stejná jako pro jakýkoli jiný identifikátor. Nový identifikátor typu lze tímto způsobem vytvořit i uvnitř bloku v libovolné úrovni zanoření.

8.3 Výčtový typ

V jazyce C můžeme definovat sadu pojmenovaných celočíselných konstant, nazývanou ***výčet (enumerace).*** Tyto konstanty lze pak použít všude, kde lze zadat celé číslo. Tyto konstanty jsou typu **int**. Výčtový typ v jazyce C slouží pouze pro definici celočíselných konstant. Nelze jej použít pro definici proměnných, které mohou nabývat pouze vyjmenovaných hodnot (jako je tomu např. v Pascalu), protože překladač považuje jakoukoli proměnnou definovanou nad tímto typem za proměnnou kompatibilní s datovým typem **int**. Neprovádí se zde žádná typová kontrola, takže do proměnné nad výčtem lze uložit jakoukoli hodnotu typu **int**. Z těchto důvodů se tento typ téměř nepoužívá pro vytváření proměnných (nemělo by to smysl).

Syntaxe výčtového typu:

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*

Obecný formát

enum *jméno_výčtu {seznam_ položek} seznam_ proměnných;*

Jednu z položek *jméno_výčtu* nebo *seznam_proměnných* lze vynechat.

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

Standardně přiřazuje překladač konstantám výčtu celočíselné hodnoty od nuly, počínaje konstantou stojící v seznamu nejvíce vlevo. Hodnoty výčtových typů nelze posílat na výstup ve tvaru, v jakém jsme je definovali – jde o identifikátory konstant. Můžeme je zobrazit pouze jako odpovídající celočíselné ekvivalenty. Obdobně je můžeme číst ze vstupu. Výčtové konstanty se tedy ve své textové podobě nacházejí pouze ve zdrojovém tvaru programu. Přeložený program pracuje již jen číselnými hodnotami výčtových konstant.

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 malými písmeny pomocí pole řetězců.
```
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]);
}
```

Jména výčtového typu jsou známá jen v programu, ale ne v žádné knihovní funkci.

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
```

Po nadefinování výčtového typu je možné použít jeho jméno pro deklaraci výčtových proměnných v jiných částech programu, ale nemá to smysl, protože do takové proměnné lze přiřadit jakoukoli hodnotu typu **int** (tedy i konstantu jiného výčtového typu). Součástí jména typu v deklaraci proměnné musí být i klíčové slovo **enum.**

Příklad:
```
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};
```

Konstanty AB a BA zde budou označovat stejné hodnoty.

8.4 Datový typ struktura

Struktura je odvozený datový typ charakteristický tím, že na rozdíl od pole může obsahovat datové položky různých typů. Říkáme, že jde o ***heterogenní*** datový typ.

8.4.1 Definice struktury, položky struktury

Obecný formát struktury:

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

Jedna z položek *jméno_typu* nebo *seznam_proměnných* může chybět.

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**.

```
struct osoba c4,c5,c6;

osoba c7; // nelze! syntaktická chyba
```

Jinou možností je zavedení nového označení datového typu pomocí **typedef.**

Příklad: Označení struktury 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;
```

Příklad: Přístup k jednotlivým položkám struktury je umožněn pomocí tečkové notace.

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

Příklad: Pro načtení hodnot do struktury je možné použít *scanf()*.

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

Příklad: Pro tisk hodnot můžeme použít: *printf()*.
```
printf("vek: %d vyska: %d vaha: %f\n",
            c3.vek,c3.vyska,c3.vaha);
```

8.4.2 Kompatibilita struktur

Pokud vytvoříme dvě struktury se stejnými položkami stejných datových typů, budou to dva různé nekompatibilní datové typy. Nad proměnnými těchto dvou datových typů nelze použít přiřazovací operátor. Tyto dva datové typy nebudou zaměnitelné při definici parametrů funkcí (jak jsme byli zvyklí například u polí). Tyto dva datové typy nelze ani vzájemně přetypovat.

Příklad: Nekompatibilní struktury.
```
typedef struct structa { int a; } Ta;
typedef struct structb { int a; } Tb;

Ta a = {1};
Tb b = a;     // chyba!
Tb b = (Tb)a; // chyba!
```

8.4.3 Inicializace struktury

Struktury se inicializují podobně jako pole uvedením seznamu inicializačních hodnot uzavřených ve složených závorkách.

Příklad: Inicializace struktury.
```
TOsoba c4 = {25, 182, 82.5};
```

Podobně lze inicializovat i pole struktur.

Příklad: Inicializace pole struktur.
```
TOsoba studenti[] = {
  {24,185,87.5},
  {25,178,82.8},
  {26,176,78.8}
};
```

Nová norma (ISO C99) zavádí možnost zapisovat inicializace přehlednějším způsobem pomocí jmen jednotlivých složek.

Příklad: Inicializace struktury pomocí jmen jednotlivých složek.
```
TOsoba c5 ={ .vek = 25, .vyska = 182, .vaha = 82.5 };
```

Se strukturou lze pracovat jako s celkem. Nad datovým typem struktura je definován přiřazovací operátor (ISO C90). Obsah jedné instance struktury můžeme přiřadit jiné, pokud jsou obě stejného typu.

Příklad: Vzájemné přiřazení proměnných typu struktura.
```
c2=c4;
```

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

Je nutné si ovšem uvědomit, že v tomto případě se provádí takzvaná ***mělká kopie***. Pokud struktura obsahuje ukazatele, pak se zkopírují pouze hodnoty ukazatelů (adresy) a nikoli takto odkazovaná data. Pokud je potřeba provádět hlubokou kopii, je nutné to udělat „ručně“.

8.4.4 Ukazatele na struktury

Podobně jako u jiných typů, lze definovat i ukazatel na proměnné typu struktura.

Příklad: Ukazatel na proměnnou typu struktura.

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

Přístup k jednotlivým položkám struktury přes ukazatele se provádí pomocí přístupového operátoru -> (znaky mínus a větší než).

Příklad: Přístup k jednotlivým položkám struktury přes ukazatele.

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

p_c2->vyska=p_c1->vyska;

// tento zápis je stručnější ...
p_c1->vek=69;
// ...než zápis
(*p_c1).vek=69;
// obě formy jsou však ekvivalentní
```

8.4.5 Alokace struktury

Pokud alokujeme paměť pro strukturu, nesmíme zapomenout použít operátor **sizeof**.

Příklad: Alokace paměti pro strukturu.

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

Pozor! Nelze se spoléhat na to, že součet velikostí jednotlivých položek bude odpovídat výsledné velikosti tohoto datového typu. Jednotlivé položky ve struktuře totiž z důvodu zarovnávání nemusí v paměti bezprostředně sousedit. Obecně pro typ *TOsoba* platí následující vztah (i když někdy může tento součet být roven velikosti výsledného typu):

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

8.4.6 Struktura odkazující sama na sebe

Pokud musí struktura odkazovat na sebe samu, nelze uvnitř struktury použít identifikátor vytvořený pomocí **typedef**. K tomuto účelu se využije skutečný název datového typu (včetně klíčového slova **struct**). Důvodem, proč nelze použít nový identifikátor vytvářený pomocí **typedef** je, že uvnitř struktury ještě není definován.

Příklad: Struktura, jejíž položka ukazuje na sebe samu.

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

Tento typ struktur odkazujících samy na sebe se často používá ve spojitosti s dynamickými datovými strukturami (seznam, fronta, strom, ...).

8.4.7 Pole jako položka struktury

Součástí struktury může být jakýkoli datový typ, tedy i pole. Díky dobře nastaveným prioritám operátorů indexování a přístupu k položkám struktury není potřeba při indexování prvků pole závorkovat.

Příklad: Definujme proměnné *p1, p2* jako strukturu *TPacient* obsahující datové položky *jmeno, prijmeni, vyska, vaha, teplota,* charakterizující zdravotní stav pacienta v nemocnici. Budeme předpokládat, že pobyt pacienta v nemocnici nebude delší než 100 dní.

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

TPacient p1,p2;
```

Po inicializaci proměnné *p1* lze zobrazit příjmení pacienta a jeho teplotu *14.* den pobytu v nemocnici následujícími příkazy:

Příklad: Zobrazení příjmení pacienta a jeho teplotu 14. den pobytu v nemocnici.

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

8.4.8 Pole struktur

S polem struktur se opět pracuje stejně jako s jakýmkoli jiným polem. Při přístupu k položkám opět není potřeba výraz nijak složitě závorkovat.

Příklad: Deklarace 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]);
```

8.4.9 Vnořená struktura

Struktura může jako položku obsahovat jinou strukturu:

Příklad: Deklarace vnořené struktury.

```
typedef enum mesic
{
  LEDEN=1,UNOR,BREZEN,DUBEN,KVETEN,CERVEN,
  CERVENEC,SRPEN,ZARI,RIJEN,LISTOPAD,PROSINEC
} TMesic;
typedef struct datum_narozeni
{
  int den;
  TMesic mes;
  int rok;
} TDatumNarozeni;
typedef struct tstudent
{
  char jmeno[30];
  char prijmeni[30];
  TDatumNarozeni datum;
  int rocnik;
} TStudent;
TStudent student;
```

8.4.10 Struktury a funkce

Podle *ANSI* normy jazyka *C* může být návratovou hodnotou funkce i struktura a struktura také může být předána funkci jako skutečný parametr.

Příklad: Struktura jako návratová hodnota funkce.

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

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

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");
  }
}
```

Předávání struktury hodnotou může být velmi neefektivní, protože by se muselo na zásobník kopírovat velké množství dat. Proto se struktury častěji do funkcí předávají přes ukazatel i v případech, kdy by je stačilo předat hodnotou. V takovém případě můžeme vytvořit konstantní parametr. Ten pak nepůjde modifikovat. Naneštěstí tím nelze ohlídat modifikaci dat přes případné ukazatele, které struktura může obsahovat.

Příklad: Součet matic předávání struktury odkazem.

```
void addMatrix(TMatrix *dest, const TMatrix *add)
{
  if(dest->rows != add->rows ||
     dest->cols != add->cols)
  {
    error();
    return ;
  }
  ... // výpočet
}
...
TMatrix m1 = allocMatrix(17, 4);
TMatrix m2 = allocMatrix(17, 4);
... inicializace
addMatrix(&m1, &m2); // musí zde být &
printMatrix(m1);     // zde & být nesmí
```

8.5 Shrnutí

Po přečtení této kapitoly je čtenář schopen specifikovat nové datové typy pomocí specifikátoru typedef. Umí jej používat spolu s jednoduchými datovými typy, ale i se strukturami a vytvářet tak komplexnější datové abstrakce. Čtenář nyní již zvládá použití výčtového typu pro tvorbu celočíselných konstant.

Čtenář dokáže vytvářet strukturovaný typ skládáním z ostatních jednodušších typů, polí, ukazatelů, ale i struktur samotných. Dokáže vytvářet hodnoty tohoto typu, inicializovat je a pracovat s jejími složkami. Dokáže také kombinovat typ struktura s typem ukazatel. Dokáže vytvářet proměnné typu ukazatel na strukturu a vytvářet položky struktury typu ukazatel, včetně ukazatele na svůj vlastní typ. Zvládá dynamickou alokaci proměnné typu struktura a přístup k jejím položkám přes ukazatel na ni.

8.6 Úlohy 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.

8.7 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;
   ```