Sedemnajste vaje iz računalništva I

Majda Jurinec

23.3.1999

Povzetek

Na današnjih vajah smo po kolokviju nadaljevali z razlago tipov. Povedali smo kaj tipi pravzaprav so, našteli tiste, ki jih poznamo in povedali kako jih definiramo, nazadnje smo se še nekaj naučili o pomnilniku.

1. Splošno o tipih

Tip spremenljivk določa zalogo vrednosti, ki jih lahko ima spremenljivka tega tipa. Vsak operator ali funkcija pričakuje operande določenega tipa in da rezultat določenega tipa. Vsaka spremenljivka lahko pripada le enemu tipu.

Osnovni tipi podatkov so:

tip	deklarira
int	cela števila
short int	majhna cela števila
long int	velika cela števila
unsigned int	nepredznačena cela števila
float	realna števila
double	velika realna števila
char	znake

Primeri deklaracij:

```
int stevilo;
short int stevec;
unsigned int pozStevilo;
float realnaStevilo;
double velikoRealnoStev;
char znak;
```

2. Tipi v jeziku c

Programski jezik **C** pozna tri osnovne tipe, ki jih uporabljamo za spremenljivke. Le-te hranijo **cela števila, realna števila** in **znake**. **Kazalci** lahko hranijo naslove spremenljivk.

C ne pozna logičnega tipa. Spremenljivke logičnega tipa lahko hranijo le dve vrednosti: pravilno (true) ali nepravilno (false). V programskem jeziku C se uporabljajo celoštevilske spremenljivke za hranjenje logičnih vrednosti. Logična vrednost pravilno je ponazorjena s celoštevilsko vrednostjo 1 in logična vrednost nepravilno z 0.

Poleg osnovnih poznamo tudi sestavljene tipe podatkov. Sestavljeni tipi podatkov se uporabljajo za hranjenje več vrednosti: več realnih in celih števil, znakov in kazalcev. Če moramo hraniti več vrednosti istega tipa, na primer 30 temperatur, potem uporabimo spremenljivko sestavljenega tipa **polja**. Za hranjenje tridesetih temperatur bo polje veliko 30 elementov in bo lahko hranilo 30 realnih števil.

Velikokrat srečamo podatke, ki so sestavljeni iz več elementarnejših podatkov. Tak podatek je datum, ki je sestavljen iz dneva, meseca in leta. Datum bomo hranili v spremenljivki **strukturnega tipa**, ki ga bomo morali še natančneje opisati.

Povedati bomo morali, da bo strukturni tip za hranjenje datuma sestavljen iz treh **komponent**. Prva bo hranila celo število dneva v mesecu, druga bo hranila polje znakov, ki bodo opisovali mesec in tretja komponenta bo znova celo število, ki bo opisovala leto.

Naslednja tabela prikazuje opis tipov v C-ju. Spremenljivke posameznih tipov potrebujejo različen pomnilniški prostor.

TIP	C
STANDARDNI celoštevilski realni znakovni	int spremenljivka; short int pozStevilo; int a=5; unsigned short int pozStevilo; float realnaSprem; double velikoRealnoStev; char znak ='a';

KAZALCI	int *naslovB; /* naslov B je kazalec na int */
	int b, c;
	naslov B=&b;
	c = *naslovB;

POLJE	int a[50];
	char beseda[25]; char nic[]="Noc ima moc.";
	float b[4];
	a[0]=5;...a[49]=12;
	beseda="abcde"; NAPAKA.!!.
	strcpy(beseda, "abcde");
	b[0]=1.2; b[3]=12.3;

STRUKTURA	struct dnevni { unsigned short int dan; unsigned short int mesec; unsigned int leto; } struct dnevni d; d.mesec=3; d.leto=1999; ali struct dnevni d={3, 5, 1999}

3. Deklaracija novih podatkovnih tipov s `typedef`

Nove tipe, se pravi zaloge vrednosti, definiramo s pomočjo ukaza `typedef`:

```
typedef <opis> <ime>;
```

Opis je lahko bolj zapleten oz. lahko je združen iz več opisov.

Tip z imenom celaStevila zgleda takole:

```
typedef int celaStevila;
```

Zgoraj smo že omenili, da je opis lahko hkrati združen iz več opisov. Kot primer navedimo merilec krvnega pritiska. Krvni pritisk ponazorimo z dvema številkama. Na primer, zdrav, mlad človek ima krvni pritisk 110/75. Tako zalogo vseh vrednosti meritev krvnega pritiska (tip meritev krvnega pritiska) opišemo z 2 številoma oz. definiramo kot par:

Pritisk: (spodnji, zgornji)

Formalno definiramo pritisk:

```
typdef struct {
   int spodnji;
   int zgornji;
} tPritisk;
```

Sedaj lahko definiramo posamezne spremeljivke, ki ponazarjajo pritiske posameznih ljudi. Na primer:

```
tPritisk TomazevPritisk, JaninPritisk;

TomazevPritisk.spodnji=80;
JaninPritisk.zgornji=120;
```

Točke v večrazsežnostnem prostoru definiramo kot n-terice:

- točka v 2D prostoru: (x, y)
- točka v 3D prostoru: (x, y, z)-
- točka v nD prostoru: $(x_1, x_2, ..., x_n)$

Kot primer si poglejmo definicijo funkcije, ki izračuna razdaljo med 2 točkama v 2D prostoru in vseh pripadajoče ostale definicije:

```
typdef struct {
  int x;         /* x - koordinata */
  int y;         /* y - koordinata */
} t_2DTocka;

double Razdalja (t_2DTocka a, t_2DTocka b) {
  return sqrt ((ax - bx) * (ax - bx) + (ay - by) * (ay - by));
} / * Razdalja */
```

Iz definicije točke lahko izpeljemo definicijo trikotnika v ravnini:

```
typdef struct {
  t_2DTocka A;      /* prvo oglisče */
  t_2DTocka B;      /* drugo oglisče */
  t_2DTocka C;      /* tretje oglisče */
} t_Trikotnik;
```

In potem funkcija, ki izračuna obseg trikotnika:

```
double obseg (t.Trikotnik trik) {

  double obseg;

  obseg = 0;
  obseg += Razdalja (trik.A, trik.B);
  obseg += Razdalja (trik.B, trik.C);
  obseg += Razdalja (trik.C, trik.A);

  return obseg;
} /* Obseg */
```

3.1 Množica točk

Mnoţico točk smo doslej predstavili kot:

```
int tocke[MAX_TOCK][MAX_COORD];
```

Kjer:

- število v 1. oklepaju popisuje število točk in
- število v 2. oklepaju število koordinat

Recimo, da imamo naslednje točke v ravnini:

```
                    . 5[x, y]



           . 3[x, y]


                        . 4[x, y]


           . 0 (0, 0)
```

in naj bo vektor 5 vsota vektorjev 3 in 4. Kako ga dobimo? Tako, da seštejemo koordinate (točke) 3 in 4 po komponentah:

```
                  /* Ničla predstavlja x => sestevamo x-e */
tocka[5][0] = tocka[3][0] + tocka[4][0];
                  /* Enka predstavlja y => sestevamo y-e */
tocka[5][1] = tocka[3][1] + tocka[4][1];
```

Pri tem seštevanju moramo paziti, da ne zamenjamo indeksov in se zavedati, kaj posamezni indeksi sploh pomenijo.

Isti postopek sedaj naredimo z drugačno definicijo točk:

```
t_2DTocka tocke[MAX_TOCK];

                  /* x koordinata točke */
tocke[5].x = tocke[4].x + tocke[3].x;
                  /* y koordinata točke */
tocke[5].y = tocke[3].y + tocke[4].y;
```

4. Pomnilnik

Pomnilnik, (*memory*) tudi spomin, je, poeostavljeno rečeno, prostor v računalniku, ki služi za spravljanje podatkov in programov. ves spomin, ki je v računalniku je v lasti operacijskega sistema – operacijski sistem z njim upravlja. Če želimo od operacijskega sistema dobiti nov pomnilnik, moramo to izrecno zahtevati in, podobno, ko pomnilnika ne potrebujemo več, ga vrnemo operacijskemu sistemu.

4.1 Funkciji za delo s pomnilnikom

Za delo s pomnilnikom so na voljo tri osnovne funkcije, ki se nahajajo v datoteki `stdlib.h`:

- Funkcija dodeli pomnilnik velik `steviloBytov`, in vrne referenco (kazalec), kje se dodeljeni pomnilnik nahaja ali pa posebno vrednost `NULL`, če nam prostora ni bil dodeljen.

  ```
  void* malloc(unsigned int steviloBytov);
  ```

- Sprosti, se pravi vrne operacijskemu sistemu, kos (blok) pomnilnika, ki se prične na naslovu `naslovPomnilnika`.

  ```
  int free(void* naslovPomnilnika);
  ```

- Funkcija vrne število zlogov, ki jih zaseda posamezna spremenljivka tipa `tip`.

  ```
  int sizeof(tip);
  ```