Nestandardni tipovi podataka

Prosti tipovi podataka su takvi da njihove vrednosti ne možemo rastavljati na jednostavnije komponente. Kao što smo mogli videti u ranijim poglavljima standardni prosti tipovi podataka su celobrojni (int), realni (float, double, long double), logički (bool) i znakovni (char). Iako ovi tipovi podataka u velikom broju slučajeva mogu biti dovoljni, pri rešavanju složenijih problema od velike pomoći mogu biti i neki nestandardni tipovi podataka o kojima će u nastavku biti reči.

Tipovi nastali preimenovanjem standardnih tipova podataka

Programski jezik C omogućava programeru da pored postojećih definiše i nove tipove podataka. Jedan od načina za definisanje novih tipova podataka jeste preimenovanje postojećih standardnih tipova.

Za definisanje novog tipa podataka u programskom jeziku C koristi se rezervisana reč **typedef** iza koje se navodi naziv postojećeg tipa podataka, a zatim naziv novog tipa podataka. Na taj način definiše se novi tip podataka jednak nekom od postojećih tipova.

Sintaksa

```
typedef <postojeci_tip> <novi_tip>;
```

Primer

```
typedef int ceo_broj;
typedef float realan_broj;
typedef bool logicki;
typedef char znak;
```

U prethodnom primeru definisani su novi prosti tipovi podataka *ceo_broj*, *realan_broj*, *logicki* i *znak*, koji u stvari predstavljaju *int*, *float*, *boolean* i *char*, redom. Kada su jednom definisani, ovi nestandardni tipovi podataka se mogu koristiti dalje u programu potpuno ravnopravno sa standardnim tipovima podataka. Tako, na primer, za deklarisanje celobrojnih promenljivih u programu, osim tipa *int*, možemo koristiti i tip *ceo_broj*, a za deklarisanje znakovnih promenljivih tip *znak*, kao što je to prikazano u narednom primeru:

```
int i;
ceo_broj a,b;
znak c;
```

Nabrojivi tipovi podataka

Često je slučaj da podaci koje želimo da koristimo u programu nisu ceobrojnog, realnog, logičkog ili znakovnog tipa. Ukoliko se radi o nabrojivom tipu podataka, jedno od rešenja je da svakoj mogućoj vrednosti novog tipa pridružimo određenu vrednost nekog od standardnih prostih tipova podataka. Tako, na primer, dane u nedelji možemo obeležiti brojevima od 1 do 7, a zatim u programu koristiti brojeve kao sinonime za dane:

```
int dan;
     ...
if (dan==6 || dan==7) printf("Vikend\n");
```

Zamislimo sada da je potrebno napraviti program koji će računati premiju za osiguranje automobila. Visina premije koju osiguranik plaća osiguravajućem društvu zavisi od marke automobila, a za pojedine marke je potrebno doplatiti i određeni bonus zbog rizika od krađe. Da bismo u programu mogli da određujemo visinu premije na osnovu marke automobila, svaku marku označićemo određenim celim brojem, kao što je to prikazano u narednoj tabeli:

Marka automobila	Broj
Audi	1
BMW	2
Citroen	3
Fiat	4
Mercedes	5
Peugeot	6
Volkswagen	7
Ostali	0

Sada bi segmenti programa koji računa premiju osiguranja izgledali ovako:

```
{Marku automobila oznacavamo celim brojem}
int marka;
float premija;
     ...
{Ukoliko je marka  Audi, Mercedes ili Volkswagen, uracunavamo bonus}
if (marka==1 || marka==5 || marka==7) premija=premija*1.1;
```

Međutim, problem sa ovakvim načinom predstavljanja marke automobila je u tome što bi svaki programer koji radi na ovakvom programu pored sebe morao da ima tabelu sa markama automobila i odgovarajućim brojevima, kako bi mogao da dešifruje prethodnu liniju koda. Bez ovakve tabele, nemoguće je razumevanje koda iz razloga što ne bismo znali koje su to marke automobila 1, 5 i 7. Pored toga, čak i da programer poseduje ovakvu tabelu, tumačenje koda bi zahtevalo stalno gledanje u tabelu, što dodatno otežava rad.

Drugi ozbiljan problem koji se javlja je azuriranje spiska marki automobila. Recimo da na spisak želimo da ubacimo Mazdu, tako da zadržimo abecedni red. U tom slučaju Mazda bi trebalo da dobije šifru 5, što bi izazvalo pomeranje ostalih marki, tako da bi Mercedes sada imao šifru 6, Peugeot 7, a Volkswagen 8. Jasno je da bi ovakva promena dovela do toga da prethodna linija koda postane pogrešna, jer navedeni brojevi u uslovu ne odgovaraju željenim markama automobila.

Ovi problemi se mogu prevazići na taj način što bi se definisao novi, **nabrojivi**, tip podataka, koji bi mogao imati neku od vrednosti iz navedenog skupa. Definisanje novog nabrojivog tipa podataka obavlja se korišćenjem rezervisane reči **enum** iza koje sledi naziv novog tipa, a zatim između vitičastih zagrada spisak mogućih vrednosti.

Sintaksa

```
enum <novi_tip> {vrednost_1,vrednost_2,...,vrednost_n};
```

Primer

```
#include <stdio.h>

enum dani {ponedeljak, utorak, sreda, cetvrtak, petak, subota, nedelja};
enum automobili {audi, bmw, citroen, fiat, mercedes, peugeot, volkswagen, ostali};

main()
{
   float premija = 500;

   dani dan = petak;
   automobili marka = mercedes;

   if(dan >= subota)
     printf("Ne radimo vikendom.\n");
```

```
   if(marka==audi || marka==mercedes || marka==volkswagen)
     premija = premija * 1.1;

   printf("%.2f\n", premija);
}
```

U prethodnom primeru su definisani nabrojivi tipovi *dani* i *automobili*, pri čemu promenljive tipa *dani* mogu imati vrednosti *ponedeljak*, *utorak*, *sreda*, *cetvrtak*, *petak*, *subota* i *nedelja*, a promenljive tipa *automobili* vrednosti *audi, bmw, citroen, fiat, mercedes, peugeot, volkswagen* i *ostali*. Zatim je deklarisana promenljiva *dan* tipa *dani* i promenljiva *marka* tipa *automobili*. Deklarisane promenljive su dalje korišćene u ostatku programa, iz čega se vidi da se promenljivama nabrojivih tipova mogu dodeljivati vrednosti, kao i to da se na njih mogu primeniti operatori poređenja =, <, >, <=, >= i <>. Prilikom korišćenja operatora poređenja važno je napomenuti da je odnos među nabrojanim vrednostima uspostavljen redosledom njihovog nabrajanja u definiciji tipa, iz razloga što se vrednostima automatski numerišu celim brojevima počev od 0. Tako, u slučaju tipa *dani* važe relacije:

```
ponedeljak < utorak < sreda < cetvrtak < petak < subota < nedelja
```

Promenljive nabrojivih tipova se mogu koristiti kao parametri funkcija i procedura, ravnopravno sa ostalim tipovima podataka. Promenljive nabrojivih tipova se mogu koristiti kao parametri ulazne naredbe **scanf**, kao i izlazne naredbe **printf**. Promenljive nabrojivih tipova se u stvari čuvaju kao celi brojevi gde prva nabrojiva promenljiva ima vrednost 0, sledeća ima vrednost 1, itd. Pošto se čuvaju kao celi brojevi promenljive nabrojivih tipova se mogu koristiti kao parametri **for** ciklusa. Na primer, sledeći kod obezbeđuje izvršenje naredbi navedenih unutar bloka 5 puta:

```
for(dan=ponedeljak; dan<=nedelja; dan=(dani)(dan+1))
{
   ...
}
```

Važno je napomenuti i to da jedna ista vrednost ne može biti navedena unutar dva različita nabrojiva tipa podataka. Na primer, definicija

```
enum domaci_automobili {fica, skala, jugo, fiat};
enum strani_automobili {audi, bmw, fiat, mercedes, volkswagen};
```

nije ispravna iz razloga što se vrednost *fiat* javlja i u tipu *domaci_automobili* i u tipu *strani_automobili*.

Primer

Napisati program koji štampa pozivne telefonske brojeve većih gradova u Srbiji.

```
#include <stdio.h>

enum gradovi {beograd, novisad, nis, kragujevac, kraljevo, cacak};

main()
{
   gradovi grad;

   for(grad = beograd; grad <= cacak; grad = (gradovi)(grad + 1))
      switch(grad)
      {
         case beograd:
            printf("011\n");
            break;
         case novisad:
            printf("021\n");
```

```
            break;
         case nis:
            printf("018\n");
            break;
         case kragujevac:
            printf("034\n");
            break;
         case kraljevo:
            printf("036\n");
            break;
         case cacak:
            printf("032\n");
            break;
      }
}
```