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 (integer), realni (real), logički (boolean) 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 Pascal 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 Pascal koristi se rezervisana reč **type** iza koje se navodi naziv novog tipa podataka, a zatim iza znaka = naziv postojećeg tipa podataka. Na taj način definiše se novi tip podataka jednak nekom od postojećih tipova.

Sintaksa

```
type <novi_tip>=<postojeci_tip>;
```

Primer

```
type
   ceo_broj=integer;
   realan_broj=real;
   logicki=boolean;
   znak=char;
```

U prethodnom primeru definisani su novi prosti tipovi podataka *ceo_broj*, *realan_broj*, *logicki* i *znak*, koji u stvari predstavljaju *integer*, *real*, *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 *integer*, možemo koristiti i tip *ceo_broj*, a za deklarisanje znakovnih promenljivih tip *znak*, kao što je to prikazano u narednom primeru:

```
var  i:integer;
     a,b:ceo_broj;
     c:znak;
```

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:

```
var  dan:integer;
     ...
if (dan=6) or (dan=7) then writeln('Zivela sloboda!!!');
```

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}
var  marka:integer;
     premija:real;
     ...
{Ukoliko je marka  Audi, Mercedes ili Volkswagen, uracunavamo bonus}
if (marka=1) or (marka=5) or (marka=7) then 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 **type** iza koje sledi naziv novog tipa, a zatim iza znaka = u zagradama spisak mogućih vrednosti.

Sintaksa

```
type <novi_tip>=(vrednost_1,vrednost_2,...,vrednost_n);
```

Primer

```
type dani=(ponedeljak,utorak,sreda,cetvrtak,petak,subota,nedelja);
     automobili=(audi,bmw,citroen,fiat,mercedes,peugeot,volkswagen,ostali);

var  dan:dani;
     marka:automobili;

     ...
dan:=petak;
marka:=mercedes;
     ...
if (dan>=subota) then writeln('Ne radimo vikendom.');
     ...
if (marka=audi) or (marka=mercedes) or (marka=volkswagen) then
   premija:=premija*1.1;
```

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
```

Zahvaljujući jasno definisanom poretku moguće je primeniti i standardne funkcije **succ** (sledbenik), **pred** (prethodnik) i **ord** (redni broj). Na primer:

```
succ(sreda) = cetvrtak
pred(utorak) = ponedeljak
ord(petak) = 4
```

Lako je zaključiti da funkcija **succ** nije definisana za poslednji, a funkcija **pred** za prvi član nabrojivog tipa.

Promenljive nabrojivih tipova se mogu koristiti kao parametri funkcija i procedura, ravnopravno sa ostalim tipovima podataka. Međutim, promenljive nabrojivih tipova se ne mogu koristiti kao parametri ulaznih naredbi **read** i **readln**, kao ni izlaznih naredbi **write** i **writeln**. Ono što je posebno zanimljivo je činjenica da se promenljive nabrojivih tipova mogu koristiti kao parametri **for** ciklusa. Na primer, sledeći kod obezbeđuje izvršenje naredbi navedenih unutar bloka 5 puta:

```
for d:=ponedeljak to petak do
   begin
      ...
   end;
```

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

```
type
     domaci_automobili=(fica,jugo,fiat);
     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*.

Promenljive nabrojivog tipa se mogu i direktno deklarisati navođenjem definicije nabrojivog tipa unutar odeljka za deklarisanje promenljivih:

```
var  dan:(ponedeljak,utorak,sreda,cetvrtak,petak,subota,nedelja);
```

Međutim, ovakav način pisanja nije preporučljiv iz razloga što se promenljive ovog tipa mogu javljati na više mesta u programu, što bi zahtevalo stalno ponavljanje definicije tipa pri njihovom deklarisanju. Dobra programerska praksa nalaže da se nabrojivi tip definiše korišćenjem rezervisane reči **type**, a da se zatim prilikom deklaracije promenljvih navodi samo naziv prethodno definisanog tipa. Na ovaj način olakšava se pisanje programa, a sam kod postaje znatno pregledniji.

Primer

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

```
program Pozivni;
type gradovi=(beograd,novisad,nis,kragujevac,kraljevo,cacak);
```

```
var  grad:gradovi;
begin
   for grad:=beograd to cacak do
      case grad of
         beograd:     writeln('011');
         novisad:     writeln('021');
         nis:         writeln('018');
         kragujevac:  writeln('034');
         kraljevo:    writeln('036');
         cacak:       writeln('032');
      end;
end.
```

Intervalni tip podataka

Za svaki skalarni tip osim realnog se može definisati novi tip podataka koji će sadržati samo određeni podinterval postojećeg tipa. Takav novi tip podataka nazivamo **intervalni tip**. Intervalni tip podataka se definiše korišćenjem rezervisane reči **type** iza koje sledi naziv novog tipa, a zatim iza znaka = interval postojećeg tipa. Interval postojećeg tipa se definiše navođenjem početne i krajnje vrednosti, između kojih stoji znak "..".

Sintaksa

```
type <novi_tip>=<pocetna_vrednost>..<krajnja_vrednost>;
```

Primer

```
type
     jednocifreni=0..9;
     slova='A'..'Z';
     dani=(ponedeljak,utorak,sreda,cetvrtak,petak,subota,nedelja);
     radni_dani=ponedeljak..petak;
```

Intervalni tip ima sve karakteristike tipa iz koga je izveden. To znači da se nad promenljivama ovog tipa mogu vršiti sve operacije kao i kod promenljivih osnovnog tipa. Jedino ograničenje o kome treba voditi računa jeste da vrednost promenljive intervalnog tipa ni u kom slučaju ne sme da izađe van opsega navedenog u definiciji tipa. Na primer, ako je promenljiva *a* deklarisana kao

```
type jednocifreni=0..9;
var  a:jednocifreni;
```

onda su operacije

```
a:=3;
a:=a+4;
```

ispravne, a operacije

```
a:=7;
a:=a+5;
```

pogrešne iz razloga što će promenljiva *a* dobiti vrednost 12 što izlazi iz opsega tipa *jednocifreni*.

Slično kao i kod nabrojivog tipa, promenljive intervalnog tipa se mogu direktno deklarisati navođenjem definicije integralnog tipa unutar odeljka za deklarisanje promenljivih:

```
var  cifra:0..9;
```

Međutim, ovakav način pisanja nije preporučljiv iz razloga što se promenljive ovog tipa mogu javljati na više mesta u programu, što bi zahtevalo stalno ponavljanje definicije tipa pri njihovom deklarisanju. Dobra programerska praksa nalaže da se intervalni tip definiše

korišćenjem rezervisane reči **type**, a da se zatim prilikom deklaracije promenljvih navodi samo naziv prethodno definisanog tipa. Na ovaj način olakšava se pisanje programa, a sam kod postaje znatno pregledniji.