PRESEK
List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **17** (1989/1990)
Številka 5
Strani 272–277

Matija Lokar:

NOVI TIPI V TURBO PASCALU

Ključne besede: matematika, računalništvo, turbo pascal, podatkovni tipi.

Elektronska verzija: http://www.presek.si/17/1001-Lokar.pdf

NOVI TIPI V TURBO PASCALU

V prejšnji številki smo spoznali, kako uporabljamo okolje Turbo pascala. V tem članku pa si oglejmo Turbo pascal kot jezik. Ima več lastnosti, ki razširjajo standardni pascal. Spoznali bomo dodatne tipe in delo z njimi. Nove številske tipe lahko zasledimo v verzijah 4 in 5, medtem ko so bili nizi vpeljani že v verziji 3. Omenimo še, da je najnovejša verzija Turbo pascala, verzija 5.5, prinesla še nekaj novosti prav na to področje. Tako je bil vpeljan nov podatkovni tip **object**, s pomočjo katerega lahko Turbo pascal uporabljamo kot objektno usmerjeni jezik. Vendar o objektnem programiranju kdaj drugič.

1. Dodatni vgrajeni številski tipi

Poleg standardnih tipov `real` in `integer` za delo s števili, pozna Turbo pascal še več drugih tipov. Oglejmo si najprej celoštevilske tipe.

Ime	Območje	Format
shortint	−128..127	predznačeni 8-bitni
integer	−32768..32767	predznačeni 16-bitni
longint	−2147483648..2147483647	predznačeni 32-bitni
byte	0..255	nepredznačeni 8-bitni
word	0..65535	nepredznačeni 16-bitni

Vseh pet tipov lahko mirno mešamo med sabo. Rezultat je vedno tistega najmanjšega tipa, ki vključuje vse uporabljene tipe. Seveda pa moramo paziti pri prirejanju vrednosti spremenljivkam, da ne prekoračimo obsega. Prednost teh novih celoštevilskih tipov je v tem, da lahko natančneje določimo, s kakšnimi števili želimo delati. Še posebej je koristen tip `longint`, saj se hitro zgodi, da moramo računati s celimi števili, večjimi od 32767 oz. 65535. Npr. radi bi izračunali zmnožek vseh naravnih števil do 9.

```
program produkt;
var i,p : integer;
begin
  p := 1;
  for i := 2 to 9 do p := p * i;
  writeln('9! = ',p);
end.
```

Program nam izračuna, da je produkt −30336, kar je očitno narobe. Če pa spremenljivko p deklariramo kot `longint`, bomo dobili pravilni rezultat 362880.

Omeniti velja še, da je bil tip `byte` vpeljan že v verziji 3.

Turbo pascalu lahko cela števila posredujemo tudi kot šestnajstiške konstante. To storimo tako, da kot prvi znak števila uporabimo $. Tako npr. `$A1` pomeni število 161, `$F4` je 244 ipd. Šestnajstiške konstante so lahko med `$00000000` in `$FFFFFFFF`.

Poleg standardnega tipa za realna števila pozna Turbo pascal še štiri.

Ime	Območje	Točnih cifer
real	$2.9 \times 10^{-39}..1.7 \times 10^{38}$	11-12
single	$1.5 \times 10^{-45}..3.4 \times 10^{38}$	7-8
double	$5.0 \times 10^{-324}..1.7 \times 10^{308}$	15-16
extended	$3.4 \times 10^{-4932}..1.1 \times 10^{4932}$	19-20
comp	$-2^{63}+1..2^{63}-1$	19-20

Zadnji tip `comp` lahko vsebuje le cela števila v navedenem obsegu, ki je približno $-9.2 \times 10^{18}..9.2 \times 10^{18}$ in ga uporabljamo, če potrebujemo res ogromna cela števila.

Medtem ko so nam celoštevilski tipi vedno na voljo, dodatnih štirih realnih tipov ne moremo uporabiti, če ne vključimo izračunavanja preko matematičnega koprocesorja. To najenostavneje storimo v samem programu, tako da v prvo vrstico datoteke napišemo
`{$N+,E+}`. Na ta način bomo v verziji 5 poskrbeli, da se bo naš program lahko izvajal na vseh računalnikih, tudi tistih, ki niso opremljeni z matematičnim koprocesorjem.

Z uporabo dodatnih tipov lahko delamo z bistveno večjimi števili, lahko pa tudi povečamo natančnost računanja.

2. Nizi

Delo z nizi je prava Ahilova peta standardnega pascala. Ker pascal ne pozna standardnega tipa za niz, si moramo pomagati s poljem znakov. To samo po sebi ne bi bilo nič slabega, če bi imeli na voljo ustrezne funkcije za delo s tako definiranimi tipi. Vendar jih v standardnem pascalu ni, čeprav jih v praksi mnogokrat potrebujemo. Zato so avtorji različnih pascalskih prevajalnikov poskrbeli za ustrezne razširitve. Žal jih je večina prikrojila nekoliko po svoje, tako tudi avtorji Turbo pascala.

Turbo pascal pozna poseben tip **string.** Pri deklaraciji spremenljivk navedemo tudi maksimalno dolžino niza, ki mora biti med 1 in 255. Če le-ta ni navedena, privzamemo maksimalno dolžino 255.

```
type vrstica = string[80];
var  dolga_vrstica : string;
                vrs1 : vrstica;
```

Nizi so spremenljivih dolžin. Deklarirana je *maksimalna* dolžina niza, dolžina, ki je niz tega tipa ne more preseči. Turbo pascal si za vsak niz zapomni njegovo dejansko dolžino. Torej nam ni potrebno uvajati posebnih spremenljivk - računalnik bo vedno vedel, kateri del maksimalno deklariranega prostora je zavzet. Pri branju spremenljivk tipa `string` uporabljamo stavek `readln`. Narekovajev pri tem ne pišemo. Če pa jih boste napisali, bodo del niza.

```
write('Ali naj nadaljujem (da/ne) ? ');
readln(odg);
if odg = 'ne' then ...
```

Vsi tipi nizov so med seboj enakovredni in jih lahko mešamo. Če tako navedemo, da je na nekem mestu potrebna spremenljivka tipa `string[23]`, lahko uporabljamo tudi spremenljivke, deklarirane kot krajši in daljši nizi. Seveda pa bodo predolgi nizi avtomatično odrezani (krajši pa ohranijo svojo vsebino, saj je 23 maksimalna in ne dejanska dolžina).

2.1. Operacije nad nizi

Oglejmo si sedaj, kaj lahko počnemo z nizi.

- Posamezne znake iz niza dobimo kot posamezne komponente polja. Tako je npr. `vrs1[1]` prvi, `vrs1[10]` pa deseti znak niza `vrs1`. Na ta način lahko tudi spremenimo posamezni znak v nizu npr. `vrs1[5] := 'e'`. Pri tem pa lahko uporabljamo le indekse med 1 in trenutno dolžino niza. Na ta način torej lahko le spreminjamo obstoječe znake v nizu, ne moremo pa niza podaljševati. Nize lahko torej jemljemo kot polja znakov ali pa kot celoto.

- Nize lahko primerjamo po velikosti. Le-ta je določena glede na kodo ASCII istoležnih znakov. Tako je npr. niz `'Pascal'` manjši od niza `'pascal'`, ker je v tabeli znakov črka P pred črko p. Če istoležnega

znaka ni (en niz je daljši od drugega), ima obstoječi znak večjo vrednost. Tako je niz 'Da' večji od niza 'D'. Znaki se primerjajo od leve proti desni. Primerjanje se zaključi pri prvem paru neenakih znakov. Tako je 'dinar' manjši od 'dolar'. Pri primerjanju lahko uporabljamo naslednje relacijske operatorje z znanim pomenom: =, <>, <, >, <=, >= .

- nize lahko stikamo s pomočjo operatorja '+'.

```
a := 'Turbo';
b := 'pascal';
c := 'Programski jezik ' + a + ' '+ b;
```

Če je dobljeni niz predolg, se presežek odreže.

2.2. Procedure in funkcije

- `function concat(s1, ..., sn) : string;`
  Spajanje nizov `s1, ..., sn`. Funkcija je nekaj posebnega, saj ima spremenljivo število argumentov. Primer:

```
a := concat('Programski ','jezik ','pascal');
```

- `function copy(s: string; i,j: integer): string;`
  Prepis podniza. Iz niza `s` vzamemo `j` znakov in sicer od `i`-tega dalje. Če je `i` večji od dolžine niza `s`, dobimo prazen niz. Če je `j` prevelik, dobimo segment niza `s` od `i`-tega znaka do konca. Primer:

```
a := 'pascal';
b := copy(a,5,3);
writeln(b);              ----->      al
```

- `procedure delete(var s:string; i,j:integer);`
  Brisanje podniza iz niza. Iz niza `s` odstrani `j` znakov in sicer od `i`-tega dalje. Če je `i` večji od dolžine niza `s`, ni zbrisan noben znak. Če je `i + j - 1` večje od dolžine niza, je rezultat prvih `i - 1` znakov niza `s` (zbriše se toliko znakov, kot se pač lahko). Primer:

```
a := 'pascal';
delete(a,2,4);
writeln(a);                    ---->     Pl
```

- `procedure insert(s1:string; var s2:string; i:integer);`
  Vrivanje niza v niz. V niz s2 na i-to mesto vstavimo niz s1. Primer:

```
a := 'Programski jezik pascal';
insert ('Turbo ', a, 18);
```

- `function length(s: string): integer;`
  Dolžina niza. Vrne dolžino niza s. Če je niz s prazen, je dolžina 0. Primer:

```
var a  :  string[100];
begin
  a := 'Programski jezik pascal ';
  writeln('dolzina = ';length(a));
```

- `function pos(s1,s2: string): integer;`
  Začetek podniza v nizu. Poišče prvo pojavitev podniza s1 v nizu s2. Če ga ni, vrne 0. Primer:

```
var  s : string;
begin
   s := '     123.5';
   { spremeni vodilne presledke v nicle }
   while pos(' ',s) > 0 do
     s[pos(' ',s)] := '0';
```

- `procedure str(r: real; var s: string);` ali
  `procedure str(r: integer; var s: string);`
  Spremeni število v niz. Spremeni realno ali celo število v niz. Pri tem spremembo lahko nadzorujemo kot pri uporabi `write`, z določanjem širine izpisnega polja in (pri realnih številih) z določanjem števila decimalk. Učinek je enak kot pri ukazu `write`, le-da se tukaj rezultat pač izpiše v spremenljivko tipa `string`. Primer:

```
str(25.3232:10:2,s);
writeln('Niz =',s);   ---->   Niz =      25.32
```

- `procedure val(s:string; var v:real; var i:integer);` ali
  `procedure val(s:string; var v:integer; var i:integer);`
  Pretvori niz v število. Če je v nizu zapisana realna ali celoštevilčna konstanta, dobi parameter `i` vrednost 0 in `v` vrednost te konstante. Če pa je pri pretvorbi prišlo do napake, torej če v `s` ni smiselni številski niz, nam `i` vrne mesto prvega znaka v nizu, pri katerem so nastopile težave. Tako kot se prejšnja procedura obnaša podobno kot `write`, je ta podobna podprogramu `read`.

Za konec pa še nekaj nalog z nizi

1. Izračunaj kvocient dveh števil na 200 decimalk natančno.
2. Ugotovi, če je beseda palindrom.
3. V danem aritmetičnem izrazu ugotovi, če so oklepaji pravilno postavljeni.
4. Preberi niz in ugotovi, če ustreza imenu datoteke v operacijskem sistemu MS-DOS.

Matija Lokar