Datoteke

Skladišta podataka

Sva skladišta podataka u računaru se mogu grubo podeliti na primarna i sekundarna, a njihova uloga je privremeno ili dugotrajno čuvanje podataka. Primarno skladište često nazivamo i radna memorija, ili skraćeno samo memorija. Radna memorija je jedino skladište podataka koje je direktno dostupno procesoru. Procesor čita i izvršava instrukcije smeštene u memoriji u kojoj su, takođe, smešteni i svi podaci sa kojima se operiše. Kao primarno skladište podataka današnji računari najčešće koriste takozvanu RAM memoriju (*Random Access Memory*).

Da bi bilo moguće izvršavanje bilo kog programa, bez obzira u kom programskom jeziku je napisan, neophodno je da se on u celini ili delimično nalazi u primarnoj memoriji računara. Tako se i sve instrukcije programa napisanog u Pascal-u pre izvršenja moraju naći u primarnoj memoriji. Takođe, prilikom deklarisanja svake promenljive, ona dobija odgovarajući prostor u radnoj memoriji u kojoj će biti smeštena njena vrednost. Međutim, po završetku izvršavanja programa, sve njegove instrukcije i podaci nad kojima je operisao se brišu iz radne memorije, kako bi se oslobodio prostor za druge programe i njihove podatke. Drugi veliki nedostatak RAM memorije je taj što podaci u njoj postoje samo dok memorija ima odgovarajuće napajanje električnom energijom. To znači da se prilikom željenog ili neželjenog (nestanak struje i sl.) isključenja računara gube svi podaci koji su u tom trenutku bili smešteni u memoriji. Takođe, primarnu memoriju sa podacima u njoj nije moguće preneti na drugu lokaciju, jer je memorija fizički vezana za računar kako bi imala konstantno napajanje strujom.

Svi navedeni nedostaci mogu se prevazići nekim od sekundarnih skladišta podataka. Sekundarna skladišta podataka kao što su magnetne trake, magnetni diskovi, optički diskovi, fleš memorije i sl. ne omogućavaju direktan pristup procesoru, već se njima pristupa posredstvom primarne memorije. S obzirom na njihovu prirodu, sekundarna skladišta ne zahtevaju konstantno napajanje električnom energijom kako bi sačuvali podatke. Zahvaljujući ovoj osobini, podaci na njima ostaju očuvani i nakon završetka izvršavanja programa ili isključivanja računara. Zabeleženi podaci se mogu kasnije ponovo očitati i koristiti neograničeni broj puta. Takođe, ovi uređaji su najčešće i prenosivi, tako da se podaci na njima mogu fizički prenositi sa jedne lokacije na drugu (na pr. muzički CD, fleš memorija itd.). Nedostatak sekundarnih skladišta u odnosu na primarna je višestruko manja brzina pisanja i čitanja podataka, što treba imati u vidu prilikom pisanja programa koji koriste podatke sa ovih uređaja, kako se ne bi ugrozile njihove performanse.

Organizacija podataka na sekundarnim skladištima

Programi koje smo do sada pisali su koristili isključivo radnu memoriju za smeštanje podataka. Svaka promenljiva koju smo deklarisali je u zavisnosti od tipa dobijala prostor odgovarajuće veličine u memoriji. Vrednost svake promenljive, bez obzira da li se radilo o prostom ili složenom tipu podatka, smo upisivali i čitali iz memorije korišćenjem naziva promenljive.

Međutim, u slučaju sekundarnih skladišta podaci su organizovani po takozvanim **datotekama** ili **fajlovima** (eng. *file*). Datoteka podrazumeva određeni memorijski prostor na sekundarnom skladištu podataka u kome može biti smeštena jedna ili više istorodnih ili raznorodnih informacija. Svaka datoteka na disku mora imati naziv, koji se najčešće sastoji od imena i nastavka (ekstenzije), razdvojenih tačkom. Ekstenzija fajla najčešće označava vrstu i format podataka koji su smešteni u tom fajlu. Tako, na primer, fajl *Proba.txt* ima ime *Proba* i ekstenziju *txt*, koja u ovom slučaju označava da se radi o tekstualnoj datoteci. Kako bi se olakšalo korišćenje podataka na disku, fajlovi su najčešće organizovani po takozvanim direktorijumima ili folderima. Direktorijumi omogućavaju lakše snalaženje pri radu sa velikim brojem fajlova, jer su na taj način fajlovi grupisani u logičke celine, baš kao što se dokumenti u nekoj kancelariji smeštaju u fascikle (eng. *folder*). Svaki direktorijum može sadržati neograničen broj fajlova, ali i drugih direktorijuma, što sve zajedno čini jednu strukturu u vidu stabla. Naziv fajla unutar jednog direktorijuma mora biti jedinstven, što omogućava da dva fajla u različitim direktorijumima mogu imati isti naziv. Da bismo znali sa kojim od ova dva fajla radimo, moramo da navedemo kompletnu putanju kojom se dolazi do željenog fajla. Putanja podrazumeva spisak svih direktorijuma, počev od osnovnog (*koren*, eng. *root*), pa sve do onog u kome se

taj fajl nalazi, razdvojenih znakom ****. Na primer, ukoliko na disku *c:* imamo direktorijum *Informatika*, pa unutar njega direktorijum *Predavanja* i konačno u njemu fajl *Proba.txt*, onda bi pun naziv fajla bio *c:\Informatika\Predavanja\Proba.txt*. Na ovaj način nedvosmisleno se zna o kom fajlu se radi. Ukoliko se ne navede putanja do fajla, onda se smatra de se misli na fajl u trenutno aktivnom direktorijumu operativnog sistema.

Pristup podacima

Prema načinu pristupanja podacima smeštenim u datotekama, sve datoteke možemo grupo podeliti na datoteke sa **sekvencijalnim** i **direktnim** pristupom. Kod sekvencijalnih datoteka čitanje podataka je moguće samo onim redom kojim su upisivani. To praktično znači da ako bismo želeli da očitamo podatak na desetom mestu, morali bismo prethodno da pročitamo svih devet podataka upisanih pre njega. U slučaju datoteka sa direktnim pristupom moguće je pročitati određeni podatak tako što mu se pristupa direktno, navođenjem njegove pozicije u datoteci.

U zavisnosti od vrste datoteke koja se koristi, programski jezik Pascal omogućava korišćenje datoteka i sa sekvencijalnim i sa direktnim pristupom.

Datoteke u Pascal-u

Da bi se iz programa napisanog u programskom jeziku Pascal moglo pristupiti nekoj datoteci na disku, potrebno je deklarisati promenljivu datotečnog tipa, koja će praktično predstavljati vezu između Pascal-a i konkretnog fajla na disku. Datotečni tip podatka u Pascal-u deklarišemo korišćenjem rezervisane reči **file** iza koje navodimo tip podataka koji su smešteni u datoteci. Korišćenjem definisanog datotečnog tipa moguće je zatim deklarisati datotečnu promenljivu, posredstvom koje će biti obavljane sve operacije nad određenim fajlom.

Sintaksa

Datotečni tip:

```
type <datotecni_tip>=file of <tip_podataka_u_datoteci>;
```

Datotečna promenljiva:

```
var  <datotecna_promenljiva>:<datotecni_tip>;
```

ili

```
var  <datotecna_promenljiva>:file of <tip_podataka_u_datoteci>;
```

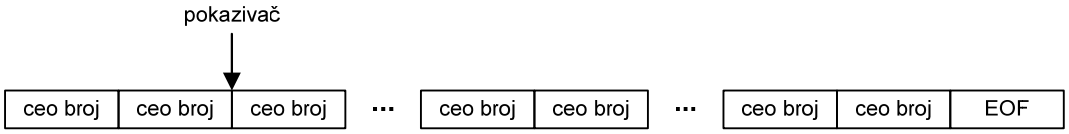
Primer

```
type
   Brojevi=file of integer;
   Reci=file of string[15];
   Tekst=file of char;
   Student=record
              ime:string[30];
              indeks:integer;
              prosek:real;
           end;
   Kartoteka=file of Student;

var  b:Brojevi;
     r:file of real;
     f:Kartoteka;
```

U prethodnom primeru je definisano više datotečnih tipova i datotečnih promenljivih. Na primer, datotečni tip *Brojevi* je definisan kao datoteka koja u sebi sadrži cele brojeve. Ovako definisanu datoteku nazivamo **tipiziranom datotekom** što znači da su svi podaci koji su zapisani u njoj (elementi datoteke) tačno

određenog tipa, slično kao i kod nizova. Korišćenjem ovog tipa deklarisana je datotečna promenljiva *b*, koja predstavlja datoteku celih brojeva. Na sledećoj slici dat je šematski prikaz jedna ovakve datoteke koja u sebi sadrži cele brojeve:



Slika ### Šematski prikaz datoteke celih brojeva

Svi podaci se u datoteku upisuju u binarnom obliku, baš onako kako su predstavljeni u memoriji računara, tako da ovakve datoteke nazivamo i **binarnim datotekama**. Ovakve datoteke nisu čitljive u tekstualnim editorima, već samo u programima koji zapisane podatke mogu da pročitaju u formatu u kome su oni i upisani u datoteku.

Prilikom korišćenja svake datoteke pridružuje joj se nevidljivi pokazivač, koji određuje do koje pozicije se stiglo sa čitanjem podataka, odnosno koji će podatak biti očitan sledeći. Kod sekvencijalnih datoteka pokazivač je na početku pozicioniran na prvom podatku u datoteci. Nakon svakog očitavanja podatka, pokazivač se pomera za jedno mesto, odnosno na sledeći podatak. Na taj način podaci se čitaju sekvencijalno jedan za drugim sve dok se ne dođe do kraja datoteke koji je označen konstantom EOF. U slučaju datoteka sa direktnim pristupom moguće je pozicionirati pokazivač na proizvoljan podatak u datoteci i očitati ga bez prethodnog isčitavanja svih podataka koji mu prethode.

Datotečna promenljiva može biti parametar potprograma, ali samo kao promenljiv parametar. Operator dodele se ne može koristiti za datotečne promenljive.

Povezivanje sa fajlom

Datotečna promenljiva se povezuje sa konkretnim fajlom na disku korišćenjem naredbe **assign** čiji su parametri naziv datotečne promenljive i naziv fajla sa kojim će ona biti povezana. Od tog trenutka sve operacije nad navedenim fajlom možemo obavljati korišćenjem datotečne promenljive sa kojom je fajl povezan. S obzirom da datotečna promenljiva u programu predstavlja ekvivalent datoteke na disku, datotečne promenljive ćemo skraćeno nazivati datoteke.

Sintaksa

```
assign(<datotecna_promenljiva>,<naziv_fajla>);
```

Primer

```
assign(f,'Proba.dat');
```

ili

```
assign(f,'c:\Informatika\Predavanja\Proba.dat');
```

Na ovaj način datotečna promenljiva *f* je povezana sa fajlom *Proba.dat*, čime je omogućeno pisanje i čitanje podataka iz navedenog fajla posredstvom ove promenljive.

Upisivanje u datoteku

Nakon što smo datotečnu promenljivu povezali sa fajlom na disku korišćenjem naredbe **assign**, datoteku možemo otvoriti za upisivanje podataka uz pomoć naredbe **rewrite** iza koje kao parametar sledi datotečna promenljiva:

```
rewrite(f);
```

Ukoliko fajl sa kojim je povezana datotečna promenljiva nije postojao, kreira se prazan fajl koji je spreman za upis. Ako je fajl postojao, svi podaci u njemu se brišu, a pokazivač za upis novi podataka se postavlja na početak fajla.

Upisivanje podataka u datoteku se vrši korišćenjem naredbe **write**, koja kao prvi parametar ima datoteku (datotečnu promenljivu) u koju se podaci upisuju, a zatim listu podataka koji se upisuju.

```
write(f,i);
```

ili

```
write(f,a,b,c);
```

Podaci koji se upisuju u datoteku moraju biti istog tipa kao i elementi datoteke. Nakon upisivanja podatka u datoteku, pokazivač se pomera na sledeću poziciju i datoteka je spremna da primi nove podatke.

Prilikom rada sa datotekama treba imati u vidu da se podaci ne upisuju istog trenutka u datoteku nakon pozivanja komande **write**, već se čuvaju u posebnom delu memorije koji se naziva **bafer**. Sekundarna skladišta podataka su znatno sporija od radne memorije, pa se korišćenjem bafera izbegava često obraćanje ovim uređajima i kvarenje performansi programa. Podaci se prebacuju na sekundarno skladište tek kada se bafer napuni, čime se veća količina podataka upisuje odjednom na disk ili drugi uređaj. Nakon prepisivanja podataka iz bafera na disk, bafer se prazni i spreman je da primi nove podatke. Ukoliko se izvršavanje programa iz nekog razloga prekine pre nego što se podaci iz bafera prebace na disk, podaci koji su se u tom trenutku zatekli u baferu neće biti upisani u datoteku.

Nakon završetka rada sa datotekom neophodno je istu zatvoriti korišćenjem naredbe **close** čiji je jedini parametar datoteka koju želimo da zatvorimo:

```
close(f);
```

Pre zatvaranja datoteke svi podaci koji su se zatekli u baferu se automatski upisuju na disk, čime je osigurano da se svi podaci zapisani naredbom **write** nađu u željenoj datoteci.

Čitanje iz datoteke

Da bismo otvorili datoteku za čitanje koristimo naredbu **reset** čiji je parametar datotečna promenljiva:

```
reset(f);
```

Naredba **reset** otvara datoteku za čitanje i pri tome pozicionira pokazivač na početak datoteke.

Čitanje podataka iz datoteke se vrši korišćenjem naredbe **read**, koja kao prvi parametar ima datoteku (datotečnu promenljivu) iz koje se podaci čitaju, a zatim listu promenljivih u koje će biti učitani podaci.

```
read(f,i);
```

ili

```
read(f,a,b,c);
```

Promenljive u koje se učitavaju podaci moraju biti istog tipa kao i elementi datoteke. Nakon čitanja podatka iz datoteke, pokazivač se pomera na sledeću poziciju i datoteka je spremna za učitavanje sledećeg podatka.

Prilikom čitanja podataka iz datoteke neophodno je stalno proveravati da li se stiglo do kraja datoteke korišćenjem funkcije **eof** čiji je argument datotečna promenljiva:

```
eof(f);
```

U slučaju da je pokazivač stigao do kraja datoteke funkcija vraća vrednost **true**, što je znak da više nije moguće čitanje iz datoteke. U suprotnom, funkcija vraća vrednost **false**.

Primer 1

Napisati program koji učitava podatke o studentima sa tastature i zapisuje ih u binarnu datoteku *Studenti.dat*.

```
program Upis;
type Student=record
               ime:string[20];
               adresa:string[30];
```

```
              indeks:integer;
            end;
var  s:Student
     f:file of Student;
     n,i:integer;

begin
   assign(f,'Studenti.dat');

   writeln('Unesite broj studenata:');
   readln(n);

   rewrite(f);

   for i:=1 to n do
      begin
         write('Ime: '); readln(s.ime);
         write('Adresa: '); readln(s.adresa);
         write('Indeks: '); readln(s.indeks);

         write(f,s);
      end;

   close(f);
end.
```

Primer 2

Napisati program koji iz postojeće datoteke *Studenti.dat*, formirane u prethodnom primeru, učitava podatke o studentima i ispisuje ih na ekranu u tabelarnom obliku.

```
program Citanje;
type Student=record
              ime:string[20];
              adresa:string[30];
              indeks:integer;
            end;
var  s:Student
     f:file of Student;

begin
   assign(f,'Studenti.dat');
   reset(f);

   writeln('Indeks    Ime                 Adresa');

   while not eof(f) do
      begin
         read(f,s);

         writeln(s.indeks:10, s.ime:20, s.adresa:30);
      end;

   close(f);
end.
```

Tekstualne datoteke

Pored binarnih datoteka, Pascal omogućava rad i sa posebnom vrstom datoteka koje se nazivaju **tekstualne datoteke**.

Tekstualna datoteka je niz znakova zapisan na nekom sekundarnom skladištu podataka. Svaki znak je predstavljen odgovarajućim ASCII kodom, odnosno celim brojem koji zauzima jedan bajt. Tako se, na primer, veliko slovo "A" predstavlja ASCII kodom 65. Pored vidljivih znakova, postoje i znaci koji se ne ispisuju grafički, ali su neophodni za uređenje teksta.

Prilikom ispisivanja teksta na ekranu ili nekom drugom izlaznom uređaju pogodno je tekst podeliti na linije, kao što je to uobičajeno. U tu svrhu koriste se nevidljivi znaci *carridge return* (ASCII kod 13) koji obezbeđuje povratak na početak linije i *line feed* (ASCII kod 10) koji obezbeđuje prelazak u novi red. Ova dva znaka označavaju novi red u tekstualnoj datoteci i nazivaju se **eoln** (eng. *end of line*). Kraj svake tekstualne datoteke označava se nevidljivim znakom **eof** (eng. *end of file*), čiji je ASCII kod 26. Tako bi datoteka koja sadrži sledeći niz znakova:

T	e	k	s	t	u	a	l	n	a		d	a	t	o	t	e	k	a	EOLN	m	o	z	e		i	m	a	t	i		v	i	s	e		r	e	d	o	v	a	EOLN	-	4	7	3	.	5	1	EOF

prikazana na ekranu ili štampaču izgledala ovako:

```
Tekstualna datoteka
moze imati vise redova
-473.51
```

Za razliku od binarnih datoteka tekstualne datoteke se deklarišu korišćenjem rezervisane reči **text**.

Sintaksa

```
var  <datotecna_promenljiva>:text;
```

Primer

```
var  f:text;
```

Pre korišćenja datotečne promenljive, neophodno ju je povezati sa odgovarajućim fajlom na disku korišćenjem naredbe **assign**, kao i u slučaju binarnih datoteka. Ovakve datoteke se otvaraju za pisanje i čitanje na isti način kao i binarne datoteke, naredbama **rewrite** i **reset**. Pored toga, u tekstulanu datoteku je moguće dodati novi tekst bez brisanje prethodnog tako što bi se datoteka otvorila za dodavanje korišćenjem naredbe **append**:

```
append(f);
```

Pored funkcije **eof** koja oređuje da li je pokazivač stigao do kraja datoteke, kod tekstualnih datoteka postoji i funkcija **eoln** koja ispituje da li je pokazivač stigao do kraja reda:

```
eoln(f);
```

Ukoliko je dosegnut kraj reda, funkcija **eoln** vraća vrednost **true**. U suprotnom ova funkcija vraća vrednost **false**.

Važno je naglasiti da se pisanje i čitanje iz tekstualne datoteke vrši na potpuno isti način kao i ispisivanje na ekran ili učitavanje sa tastature, sa jedinom razlikom što se kao prvi argument naredbi **write**, **writeln**, **read** i **readln** mora navesti datoteka (datotečna promenljiva) u koju se upisuje, odnosno iz koje se čitaju podaci:

```
write(f,a,b:5);
writeln(f,'Zbir je ',a+b);

read(f,x);
readln(f,x,a,b);
```

Primetimo da u slučaju tekstualnih datoteka parametri naredbi **write**, **writeln**, **read** i **readln** mogu biti proizvoljnog tipa.

Tekstualne datoteke se ponašaju potpuno isto kao i standardni ulaz i izlaz, pa iz tog razloga nećemo detaljno objašnjavati pravila pisanja i čitanja, jer su ona ekvivalentna učitavanju podataka sa tastature i ispisivanju na ekran. Programer se pri radu sa tekstualnim datotekama može potpuno osloniti na dosadašnja iskustva u korišćenju ulazno-izlaznih naredbi. Učitavanje podataka iz tekstualne datoteke je ekvivalentno učitavanju podataka sa tastature, a upisivanje podataka u datoteku ekvivalentno ispisivanju podataka na ekran.

Primer 1

Napisati program koji sa tastature učitava *n* prirodnih brojeva i upisuje ih u tekstualnu datoteku *Matrica.txt*, tako da u svakom redu bude *k* brojeva. Poslednji red u datoteci može imati i manje od *k* brojeva, ukoliko *n* nije celobrojan umnožak broja *k*.

```
program Upis;
var  f:text;
     n,k,i,x:integer;

begin
   assign(f,'Matrica.txt');

   writeln('Unesite n i k:');
   read(n,k);

   rewrite(f);

   for i:=1 to n do
      begin
         read(x);

         write(f,x);

         if i mod k=0 then writeln(f);
      end;

   close(f);
end.
```

Primer 2

Iz tekstualne datoteke formirane u prethodnom primeru učitati brojeve i smestiti ih u odgovarajuću matricu. Elemente dobijene matrice odštampati na ekranu u matričnom obliku.

```
program Ucitavanje;
const   MAX_REDOVA=100;
        MAX_KOLONA=100;
var  f:text;
     a:array[1..MAX_REDOVA,1..MAX_KOLONA] of integer;
     i,j:integer;

begin
   assign(f,'Matrica.txt);
   reset(f);
```

```
   i:=1;
   while not eof(f) do
      begin
         j:=1;
         while not eoln(f) do
            begin
               read(f,a[i,j]);
               write(a[i,j]);
               j:=j+1;
            end;

         readln(f);
         writeln;
         i:=i+1;
      end;

   close(f);
end.
```