Univerzitet u Novom Sadu
Prirodno-matematički fakultet
Departman za matematiku i informatiku

Dragan Mašulović

Uvod u programiranje I
(za gimnazijalce)

Novi Sad, 2016.

Predgovor

Računarski program, na prvi pogled, predstavlja sredstvo komunikacije čoveka i mašine. Zapravo, radi se o komunikaciji *među ljudima*, i to višeslojnoj i komplikovanoj. Na jednom nivou računarski program predstavlja oblik komunikacije između dizajnera programa i korisnika programa: dizajneri u program ugrađuju svoj pogled na svet ili jedan njegov segment, i to prenose korisniku programa. Računarski program, međutim, predstavlja i oblik komunikacije među programerima. Zato je važno da progami budu ne samo dobro osmišljeni, već i dobro napisani, kako bi mogli da se upotrebe na oba nivoa komunikacije.

Ovaj udžbenik predstavlja kurs progamiranja. *Naglasak je na programerskim tehnikama*, dok se programski jezik Paskal (Pascal) koristi samo kao sredstvo (komunikacije čoveka i mašine).

Materijal je razvijen u skladu na nastavnim planom i programom predmeta *Računarstvo i informatika* za specijalna matematička odeljenja, mada mogu da ga koriste i učenici ostalih srednjih škola, kao i zainteresovani učenici starijih razreda osnovnih škola. Od zahteva predviđenih planom i programom odstupilo se samo u dva segmenta: rekurziju ostavljamo za narednu godinu, a umesto ove nastavne jedinice obrađujemo tekstualne datoteke. Od rekurzije smo odustali u ovoj godini zbog nedostatka prirodnih primera, a rad tekstualnim datotekama je važan učenicima koji žele da učestvuju na takmičenjima iz informatike.

Na kraju svake sveske rezimiraćemo konstrukcije onog dela programskog jezika Paskal koga smo do tog trenutka naučili. Rezime sintakse programskog jezika će biti dat u prilično (mada ne potpuno) formalnom obliku koji se zove *Bakusova notacija*, a leksiku jezika ćemo rezimirati neformalno. Formalni rezime sintakse ne treba učini naizust! On treba samo da nam posluži kao način da se izdignemo iznad niza primera koji su navedeni u tekstu i tako sistematizujemo znanje koje smo nakupili do tog trenutka.

U Bakusovoj notaciji element sintakse se definiše ovako

$$\langle \textit{element} \rangle \equiv \langle \textit{pravilo} \rangle .$$

Pravilo koje se navodi na desnoj strani će obično biti niz elemenata, alternacija ili

opcija. *Niz* ima oblik

$$\langle \textit{niz elemenata} \rangle \equiv \langle \textit{element}_1 \rangle \langle \textit{element}_2 \rangle \ldots \langle \textit{element}_n \rangle$$

i označava jezičku konstrukciju kod koje je potrebno navesti $\langle \textit{element}_1 \rangle$, potom $\langle \textit{element}_2 \rangle$, i tako dalje sve do $\langle \textit{element}_n \rangle$, tačno tim redom. *Alternacija* ima oblik

$$\langle \textit{element} \rangle \equiv \langle \textit{pravilo}_1 \rangle \mid \langle \textit{pravilo}_2 \rangle$$

i označava da se $\langle \textit{element} \rangle$ može formirati prema $\langle \textit{pravilu}_1 \rangle$ ili prema $\langle \textit{pravilu}_2 \rangle$. *Opcija* ima oblik

$$[\langle \textit{element} \rangle]$$

i označava da $\langle \textit{element} \rangle$ može, ali i ne mora da se pojavi pri formiranju rečenice. Naravno, moguće su i razne kombinacije nizanja, opcija i alternacija.

Glava 1

Prvi program u Paskalu

U ovoj glavi počinjemo sa predstavljanjem osnovnih osobina programskog jezika Paskal. Počećemo analizom jednostavnog primera, a onda ćemo pokušati da po analogiji napravimo nekoliko sličnih programa.

Pored je dat Paskal program kojim se računa obim kruga. Program radi ovako:

(1) učita od korisnika poluprečnik kruga,

(2) izračuna obim, i

(3) ispiše dobijenu vrednost obima.

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

1.1 Osnovna objašnjenja

Prvi red svakog Paskal programa sadrži rezervisanu reč `program` (kasnije ćemo detaljnije objasniti šta su tačno rezervisane reči) iza koje treba navesti ime programa. Ime može biti bilo koja reč. Iza svega se *obavezno* navodi ";".

Ime programa nema nikakve veze sa imenom datoteke u kojoj

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

se nalazi taj program. Tako, program `ObimKruga` možemo, ako to baš želimo, smestiti u datoteku `kocka.pas` bez ikakvog problema. Ipak, ime datoteke se najčešće bira tako da bar malo podseća na ime programa, samo zato da bismo se kasnije lakše mogli snaći.

Kada smo programu nadenuli ime, krećemo polako sa opisom onoga što ćemo koristiti u programu. U ovom primeru, posle imena programa definišemo jednu konstantu. Konstante se definišu tako što navedemo rezervisanu reć `const` iza koje sledi niz (jedna ili više) definicija konstanti. Svaka definicija izgleda ovako:

ime = *vrednost*;

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Ako u programu ne koristimo konstante, onda se ceo ovaj odeljak izbaci. Nema ga.

☞ *Primetite da se decimalni brojevi pišu koristeći* ***decimalnu tačku,*** *prema engleskom pravopisu, a ne decimalni zarez kako nalaže naš pravopis!*

Nakon definicije konstanti (ako ih ima), slede deklaracije promenljivih. *Promenljiva* je parče memorije računara u koje može da se smesti neka vrednost.

Promenljiva ima svoje *ime* i svoj *tip*. Valjda je svima jasno šta je ime promenljive i zašto promenljiva mora da ima ime. Tip promenljive objašnjava računaru koliko memorije da zauzme za tu promenljivu, kao i to koje operacije se mogu izvoditi sa vrednostima koje su smeštene u promenljivoj. Ako ovo nije baš sasvim jasno, strpite se malo. Razjasniće se kasnije. U ovom slučaju trebaju nam dve promenljive koje smo nazvali `r` i `Obim`. Obe su tipa `real` što znači da se u svakoj može zapamtiti neki decimalan broj.

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Na početku odeljka u kome se deklarišu promenljive nalazi se rezervisana reč `var` (od engleskog *variables*). Potom sledi niz deklaracija. Svaka deklaracija izgleda ovako

ime `:` *tip*`;`

ili ovako

ime$_1$`,` `...` `,` *ime*$_k$ `:` *tip*`;`

čime deklarišemo nekoliko promenljivih istog tipa.

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Kada smo računaru opisali sva važna imena u programu (ime programa, imena konstanti [i njihove vrednosti] i imena promenljivih [i njihove tipove]), sledi deo programa koji objašnjava računaru šta u stvari treba da radi. Taj deo programa nazivamo "glavni deo programa" ili "glavni program".

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Na početku glavnog dela programa se nalazi rezervisana reč `begin`, a na kraju rezervisana reč `end`. Na kraju svega, na samom kraju, nalazi se tačka. Kao na kraju rečenice. A to i jeste ideja: program treba da shvatimo kao jednu dugačku rečenicu jednog veštačkog jezika. Pa je onda red da se na kraju rečenice stavi tačka.

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Kviz.

1. Šta će se desiti ako "rečenicu" počnemo velikim slovom, tj. ako umesto `program` napišemo `Program`?

Odgovor. Ništa. Paskal ne pravi razliku između velikih i malih slova. Rezervisane reči i imena možemo pisati kako god želimo. Može `PrograM`, `prOGRam` ili `PROGRAM` umesto `program`; `obim`, `oBIm` ili `OBIM` umesto `Obim`.

2. Šta će se desiti ako programu damo drugo ime? Da li će to uticati na rad programa?

3. Da li bi program radio drugačije da smo promenljivu `Obim` nazvali `ob`?

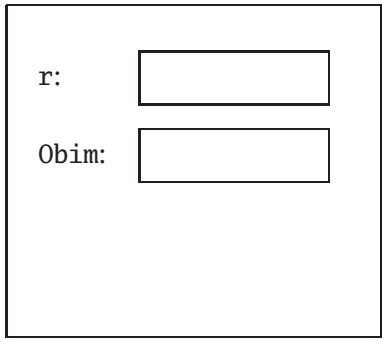
4. Da li možemo imati dve promenljive sa istim imenom? A da li promenljiva i konstanta smeju da imaju isto ime?

Odgovor. Ne. U programu ne smeju postojati dva "entiteta" (promenljive, konstante i tome slično) sa istim imenom. (Doduše, ponekad i smeju, ali za sada neka bude ovako, da ne iskomplikujemo previše odmah na početku.)

1.2 Glavni deo programa

Glavni deo programa objašnjava računaru šta treba da radi, korak po korak. Šta je tačno "jedan korak" objasnićemo kasnije. Iako ste do sada već svi sigurno pogodili kako radi program `ObimKruga`, ipak ćemo ga malo prokomentarisati.

Pre nego što počne da radi, računar u svojoj memoriji rezerviše prostor za promenljive. O prostoru za promenljive možemo razmišljati kao o ormanu sa fijokama u koje računar može da stavi neki broj, i kojima može da izmeni sadržaj. Kako naš program ima dve promenljive u memoriji će biti napravljene dve fijoke: jedna će se zvati `r`, a druga `Obim`.




```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Prva naredba ispisuje poruku na monitoru. Reč `writeln` je kovanica i čita se "write line" (*'ρajt lajn*), a znači "ispiši red (teksta)". Tekst koji je naveden između navodnika u naredbi `writeln` se ispiše u jednom redu na monitoru.

MONITOR:

```
Unesi poluprecnik
```

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

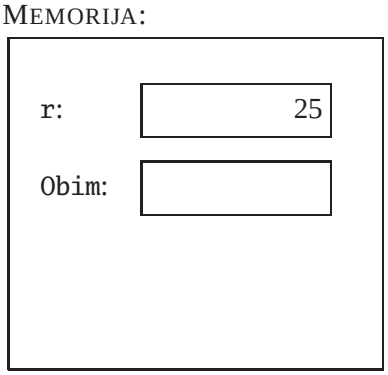
Drugom naredbom se od korisnika učitava neki decimalan broj. Reč `readln` je takođe kovanica i čita se "read line" (*ρi:d lajn*), a znači "čitaj iz reda". Kada naiđe na ovu naredbu, računar prekine sa radom i sačeka da korisnik otkuca neki decimalan broj i pritisne taster Enter. (U primeru korisnik je uneo broj 25. Obratite pažnju na to da se nakon pritiska na taster Enter na monitoru neće pojaviti uokvirena reč "Enter"; ovo je samo da se podsetimo da računar neće nastaviti sa radom dok mu pritiskom na Enter ne stavite do znanja da ste završili sa kucanjem.)

MONITOR:

```
Unesi poluprecnik
25 Enter
```

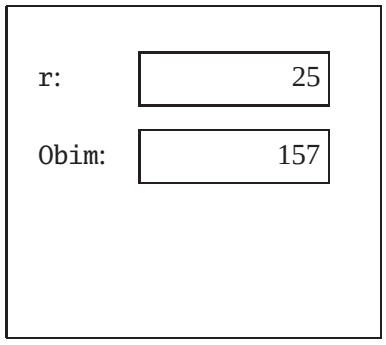
```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Računar potom smesti uneti broj u promenljivu koja je navedena (u ovom slučaju r), to jest u odgovarajući deo memorije, i nastavi sa radom.




```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Sledeća naredba je naredba dodele. Računar izračuna vrednost izraza sa desne strane i rezultat smesti u promenljivu čije ime je navedeno sa leve strane. U programskom jeziku Paskal naredba dodele se označava operatorom := (dvotačka-jednako) kako bi se razlikovala od znaka = koji se koristi u matematičkom smislu jednakosti. Nakon naredbe dodele, u fijoku sa imenom Obim upiše se broj 157.




```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Ponovo naredba writeln, sada u malo komplikovanijem obliku. Ovom naredbom se korisnik obaveštava o rezultatu računanja. Niz znakova pod navodnicima je tekst koji se doslovno ispisuje na monitor. Sledi ime promenljive, što znači da će na monitoru biti ispisana njena vrednost. Dakle, naredbom writeln se mogu ispisivati poruke, ali i vrednosti promenljivih.

MONITOR:

```
Unesi poluprecnik
25
Obim = 1.5700000000E+002
```

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

☞ *Prilikom ispisa, na monitoru se* ne *pojavljuju apostrofi!*

Primetimo i to da je broj 157 ispisan na neuobičajen način. Računar je, zapravo, ispisao $1.57 \cdot 10^2$ (niz simbola iza slova E predstavlja eksponent). Ovaj specijalitet kuće ćemo detaljno razjasniti u narednoj glavi.

Paskal program može imati više konstanti i više promenljivih, kako to pokazuje primer ObimKruga2.

Program ne mora imati ni konstante ni promenljive. Kao primer navodimo čuveni program HelloWorld koji na monitoru ispiše samo jednu poruku (ali sa dubokim smislom).

Ukoliko program ima i konstante i promenljive, redosled navođenja elemenata je *strogo određen*: prvo se definišu konstante, a potom se deklarišu promenljive. Obrnuti redosled *nije dozvoljen* mada noviji Paskal prevodioci dopuštaju i takve egzibicije.

```
program ObimKruga2;
const
  Pi = 3.14;
  TPi = 3.141592653589793;
var
  r, Obim : real;
  TObim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * Pi;
  TObim := 2 * r * TPi;
  writeln('Obim = ', Obim);
  writeln('Tacniji obim = ', TObim)
end.
```

```
program HelloWorld;
begin
  writeln('Hello World!')
end.
```

I za kraj, pogledajmo još jednom u čemu je razlika između naredbi `writeln(Obim)` i `writeln('Obim')`:

MONITOR:

```
Obim
1.5700000000E+002
Obim 1.5700000000E+002
```

```
program WritelnObim;
var
  Obim : real;
begin
  Obim := 157;
  writeln('Obim');
  writeln(Obim);
  writeln('Obim ', Obim);
end.
```

Prva `writeln` naredba ispisuje tekst `Obim`, zato što se se u programskom jeziku Paskal tekst pod apostrofima direktno prepisuje na monitor. Druga `writeln` naredba ispisuje vrednost promenljive `Obim`, a treća `writeln` naredba ispisuje prvo tekst, a onda vrednost promenljive `Obim`, sve u istom redu.

Kviz.

1. Označiti deklaracije promenljivih koje su ispravne:

☐
```
var
    a, b : real;
```

☐
```
var
    a; b : real;
```

☐
```
var
    a, b : real, real;
```

☐
```
var
    3.14 : real;
```

☐
```
var
    a = 10;
```

☐
```
var
    a and b : real;
```

☐
```
var
    a : real;
    b : real;
```

☐
```
var
    a : real;
var
    b : real;
```

2. Označiti deklaracije konstanti koje su ispravne:

☐ `const porez = 0.15;`	☐ `const porez = 15%;`
☐ `const pi = 4;`	☐ `const 2pi = 6.28;`
☐ `const 2*pi = 6.28;`	☐ `const deset = 9;`
☐ `const DesetPolovina = 5;`	☐ `const Deset2 = 5;`
☐ `const Deset/2 = 5;`	☐ `const pi := 3.14;`
☐ `const 5 = 5;`	☐ `const a = b = 5;`
☐ `const a = 5; b = 5;`	☐ `const a = 5; const b = 5;`

Zadaci.

1.1. Napisati po analogiji Paskal program koji računa površinu kruga.

1.2. Napisati po analogiji Paskal program koji računa obim i površinu pravougaonika. (Uputstvo: Sabiranje se označava znakom + kao što smo i navikli, množenje ima veći prioritet od sabiranja, a mogu se koristiti i obične zagrade tačno onako kako smo navikli u matematici.)

1.3. Napisati Paskal program koji učitava temperaturu iskazanu u Celzijusima (t_C) i ispisuje odgovarajuću vrednost temperature u Farenhajtima (t_F). Odnos je sledeći:

$$t_F = \gamma \cdot t_C + \delta$$

gde je $\gamma = 1.8$, a $\delta = 32$. Pri tome koristiti konstante i obratiti pažnju da na raspolaganju nemamo grčka slova kao ni indekse.

1.4. Napisati Paskal program koji učitava temperaturu iskazanu u Farenhajtima i ispisuje odgovarajuću vrednost temperature u Celzijusima.

1.5. Napisati Paskal program koji učitava cenu nekog proizvoda u koju nije uračunat PDV, i ispisuje cenu istog proizvoda sa uračunatim PDV. Uzeti da PDV iznosi 18%.

1.6. Napisati Paskal program koji učitava cenu nekog proizvoda sa uračunatim PDV, i ispisuje cenu istog proizvoda bez PDV. Uzeti da PDV iznosi 18%.

1.7. Napisati Paskal program koji od korisnika učitava realan broj r i potom računa i štampa zapreminu lopte čiji poluprečnik je r koristeći formulu $V = \frac{4}{3}r^3\pi$.

1.8. Napisati Paskal program koji od korisnika učitava realne brojeve r i H, nakon čega računa i štampa površinu i zapreminu pravilnog uspravnog valjka čiji poluprečnik je r, a visina H koristeći formule $P = 2r\pi(r+H)$ i $V = r^2\pi H$.

1.3 O razmacima i stilu

Paskal prevodilac (kompajler) ne vodi računa o broju razmaka koji se pojavljuju u programu (osim, naravno, onih koji se pojavljuju u porukama korisniku). Tačnije, ne pravi razliku između jednog i trideset sedam razmaka. Bitno je samo da na nekim mestima postoji bar jedan razmak. Na primer, važno je da se između `const` i `pi = 3.14;` nalazi bar jedan razmak (ili da je ovaj drugi deo u novom redu) zato što je nemoguće utvrditi šta bi tačno značilo `constpi=3.14;`. Prevodilac ne bi imao druge, nego da pretpostavi da je `constpi` jedna reč.

Ova lepa osobina (da broj razmaka nije bitan) se koristi da bi se naznačila struktura programa. Ostavljanjem nekoliko razmaka na početku reda, ali na sistematski i lukav način, program dobija na čitkosti.

Program `ObimKruga` koga smo analizirali u prethodnom odeljku se može napisati i ovako→ ali teško da bi neko pri zdravom razumu ovako zapisan program smatrao privlačnim.

```
program ObimKruga; const
pi = 3.1415; var r, Obim
: real ; begin writeln (
'Unesi  R ->') ; readln(
r); Obim := 2 * r * pi ;
writeln('Obim = ', Obim)
          end.
```

Postoje preporuke o tome kako se nazubljuje program da bi bio “lep” i “čitak”. Osnovna ideja je da se definicije `const` odeljka, deklaracije `var` odeljka i sve naredbe između `begin` i `end` uvuku za nekoliko razmaka (najčešće 2, 3 ili 4).

1.4 Vrste reči — leksika programskog jezika

Mada je Paskal veštački jezik, i on, po ugledu na prirodne jezike, ima vrste reči, pravila interpunkcije i gramatiku. Spisak propisa koji regulišu vrste reči, pravila interpunkcije (i još neke poslove) zove se *leksika* jezika. Spisak propisa koji regulišu gramatiku jezika zove se *sintaksa* jezika.

Rezervisane reči opisuju odeljke programa ili neke specijalne programske konstrukcije. Do sada smo videli sledeće rezervisane reči programskog jezika Paskal:

```
program    const    var    begin    end
```

Evo i spiska svih rezervisanih reči Paskala:

```
and      array    begin      case     const    div       do
downto   else     end        file     for      function  goto
if       in       label      mod      nil      not       of
or       packed   procedure  program  record   repeat    set
then     to       type       until    var      while     with
```

Imena. Pored rezervisanih reči postoje još neke reči koje su "rezervisane" time što imaju unapred određeno značenje, ali za njih ipak ne kažemo da su rezervisane reči. To su imena tipova i standardnih procedura i funkcija. Do sada smo se sreli sa tipom `real` i standardnim procedurama `readln` i `writeln`. Ove reči ne zovemo rezervisanim rečima zato što možemo da im promenimo značenje. Evo primera besmislenog programa koji deklariše promenljivu sa imenom `readln`, dodeljuje joj neku vrednost, koju potom ispisuje. Iako su ovakve stvari moguće, niko priseban to ne radi, uglavnom zato što tada gubimo prethodno značenje reči.

```
program BesmislenoAliRadi;
var
  readln : real;
begin
  readln := 1.14;
  writeln(readln)
end.
```

Druga vrsta imena koja se mogu javiti u Paskal programu su imena koja programer daje programu, konstantama, promenljivim i ostalim stvarima koje se mogu pojaviti u programu.

☞ *Ime je niz slova i brojeva koji mora početi slovom.*

Primeri ispravnih imena: `obimKruga`, `programbroj14`, `string2real`, `a1b2c3`.

Primeri neispravnih imena:

`2obim`	ime mora da počinje slovom,
`Obim@Kruga`	pojavljuje se znak koji nije ni slovo ni broj,
`Obim Kruga`	u imenu ne sme da se pojavi razmak

Velika i mala slova nemaju nikakav uticaj (osim unutar poruke korisniku). Kada čita imena i rezervisane reči, prevodilac ne obraća pažnju na veličinu slova, pa je za njega `Program`, `pRoGrAm` i `proGRAM` isto što i `program`. Takođe je `Pi`, `pI` i `PI` isto što i `pi`.

Znaci interpunkcije razdvajaju različite delove programa, ili manje delove unutar većih delova. Do sada smo se sreli sa četiri znaka interpunkcije: tačka, zarez, dvotačka i tačka-zarez.

Tačka se javlja na kraju programa i kod zapisa decimalnih brojeva kao decimalna tačka. Ona se koristi i za neke druge poslove, ali oni na red dolaze kasnije.

Zarez razdvaja argumente u pozivu funkcije i procedure.

Dvotačku koristimo u deklaraciji raznih stvari da bismo istakli njihov tip. Za sada nam se ova situacija javila samo kod deklaracije promenljive, a javiće se kasnije kod priče o procedurama i funkcijama.

```
program ObimKruga;
const
  pi = 3.14;
var
  r, Obim : real;
begin
  writeln('Unesi poluprecnik');
  readln(r);
  Obim := 2 * r * pi;
  writeln('Obim = ', Obim)
end.
```

Tačka-zarez ima dvojaku ulogu. Pri davanju imena programu, definisanju konstanti i deklaraciji promenljivih tačka-zarez se javlja kao *terminator*: označava kraj definicije/deklaracije. U glavnom delu programa tačka-zarez ima ulogu *separatora*: razdvaja naredbe. Zato iza poslednje naredbe ne pišemo `;` – iza nje nema naredbe od koje bi je trebalo razdvojiti. (Srećom, ako iza poslednje naredbe i napišemo `;` neće se ništa loše desiti: Paskal prevodilac će shvatiti da smo je razdvojili od *prazne naredbe*, koja, naravno, ne radi ništa.)

```
program ObimKruga;
const
    pi = 3.14;
var
    r, Obim : real;
begin
    writeln('Unesi poluprecnik');
    readln(r);
    Obim := 2 * r * pi;
    writeln('Obim = ', Obim)
end.
```

Komentari predstavljaju poslednji element leksike koga ćemo pomenuti. Komentar je tekst koji je namenjen čoveku i zato ga Paskal prevodilac ignoriše. Komentar se navodi između vitičastih zagrada { i }, ili između "specijalnih" zagrada (* i *) i može se nalaziti bilo gde u programu, osim unutar poruke korisniku.

Komentare ostavlja programer ili zato da bi sebi objasnio šta je hteo da postigne nekim delom programa, ili da pomogne onom ko posle njega bude čitao program u želji da shvati šta se tu dešava. Prava uloga komentara se ne može dobro videti na malim primerima. Kasnije, kada budemo pisali veće programe, videćemo da su dobri komentari ključni za razumevanje programa.

☞ *Programski jezik Paskal ne podržava ugnježdene komentare.*

Komentari u primeru koji sledi svakako nisu primer dobrih komentara. Ono čemu treba težiti je dobro komentarisan program, a ne programirani komentari. Iživljavanje u primeru je bilo neophodno da bi se pokazalo što više mogućnosti komentara.

```
{ Na pocetku programa se cesto navodi sta program
  radi, ko je autor itd }
program ObimKruga;

(* Najcesce se u celom programu koristi ista vrsta zagrada
   za komentare, da bi se ona druga vrsta mogla koristiti
   prilikom debagiranja *)
const
    pi = {na dve decimale} 3.14;

var
    r {poluprecnik}, Obim : real;

begin
    writeln('Unesi poluprecnik');
    readln(r);

    { Koristimo poznatu formulu }
    Obim := 2 * (* jesmo li vec pomenuli da * oznacava
        mnozenje? *) r * pi;

    writeln('Obim = ', Obim)
end.
```

Kviz.

1. U sledećoj tabeli označiti rezervisane reči

☐ `and`	☐ `beginning`	☐ `file`
☐ `Const`	☐ `goto`	☐ `cases`
☐ `ReservedWord`	☐ `record`	☐ `than`
☐ `end`	☐ `go to`	☐ `Not`
☐ `then`	☐ `RezervisanaRec`	☐ `Ende`
☐ `Xfiles`	☐ `cont`	☐ `records`
☐ `with`	☐ `reset`	☐ `real`
☐ `set`	☐ `without`	☐ `nil`

2. U sledećoj tabeli označiti one konstrukcije koje se mogu koristiti kao imena (promenljivih, konstanti itd):

☐ `Vojvodina`	☐ `1234A`	☐ `Torta`
☐ `A*b`	☐ `NeSme`	☐ `Ne Sme`
☐ `A1234`	☐ `Program`	☐ `$xyz`
☐ `555-2313`	☐ `END`	☐ `1end`
☐ `enda`	☐ `writeln`	☐ `real`
☐ `Don't`	☐ `Dont`	☐ `ime+prezime`
☐ `AAA`	☐ `A1B2C3`	☐ `1a2b3c`

3. Ako promenljiva x ima vrednost 3, a promenljiva y vrednost 5, koju vrednost imaju sledeći izrazi:

`x { + y }` = ☐

`x { + } - y` = ☐

`(* x *) y` = ☐

`2 (* x *) + 3 (* y *)` = ☐

`x * (* x *) x` = ☐

`3 { x } { + } { 4y }` = ☐

`{ - } x * y` = ☐

`5 * (* y * *) x` = ☐

4. Na koliko različitih načina se može zapisati ime promenljive `Obim`, a da prevodilac ništa ne primeti? (Napomena: jedina stvar koju prevodilac neće primetiti je razlika u veličina slova!)

1.5 Rezime

Leksika programskog jezika predstavlja spisak propisa koji regulišu vrste reči, pravila interpunkcije i komentare.

Reč programskog jezika Paskal je niz slova (`a`, `b`, `c`, ..., `z`, `A`, `B`, `C`, ..., `Z`) i cifara (`0`, ..., `9`) koji počinje slovom. Na primer, `program`, `begin`, `end`, `Obim` i `q12` su reči programskog jezika Paskal. Sve reči programskog jezika Paskal se dele u tri grupe:

- rezervisane reči,
- standardna imena, i
- ostala imena.

Rezervisane reči opisuju odeljke programa ili neke specijalne programske konstrukcije. Rezervisane reči programskog jezika Paskal su:

```
and      array    begin      case     const    div      do
downto   else     end        file     for      function goto
if       in       label      mod      nil      not      of
or       packed   procedure  program  record   repeat   set
then     to       type       until    var      while    with
```

Standardna imena su imena koja Paskal prevodilac dodeljuje unapred definisanim procedurama i funkcijama. Na primer, `writeln` i `readln` su standardna imena (a ima ih još). *Ostala imena* su reči koje programer dodeljuje elementima svog programa. Na primer, `Obim`, `pi` i `r` su imena koja nisu standardna imena.

Znaci interpunkcije razdvajaju različite delove programa, ili manje delove unutar većih delova. Do sada smo se sreli sa četiri znaka interpunkcije: `.` (tačka), `,` (zarez), `:` (dvotačka) i `;` (tačka-zarez).

Operatori predstavljaju akcije koje treba izvršiti nad nekim promenljivim ili konstantnim vrednostima. Operatori koje smo do sada sreli su `+`, `-`, `*`, `=` i `:=`.

Literali predstavljaju konstantne vrednosti koje navodimo u programu. Tipičan primer literala su brojevi koji se javljaju u algebraskim izrazima. Na primer, `3.14` ili `-57` su literali.

Komentar je tekst koji je namenjen čoveku i zato ga Paskal prevodilac ignoriše. Navodi se između vitičastih zagrada `{` i `}`, ili između "specijalnih" zagrada `(*` i `*)` i može se nalaziti bilo gde u programu, osim unutar poruke korisniku. Programski

jezik Paskal ne podržava ugnježdene komentare koji su ograđeni istim zagradama. Dozvoljeno je, međutim, da se unutar komentara jedne vrste (recimo `{ ... }`) nalazi komentar druge vrste (u ovom slučaju `(* ... *)`).

Sintaksa programskog jezika je spisak propisa koji regulišu gramatiku jezika, odnosno, način na koji reči, znaci interpunkcije, operatori, literali i ostali leksički elementi mogu da se kombinuju u ispravne *rečenice*.

Evo sada kako se formalno može opisati fragment programskog jezika Paskal sa kojim smo se do sada upoznali:

⟨*Paskal program*⟩ ≡ `program` ⟨*ime*⟩`;`
[⟨*Definicije konstanti*⟩]
[⟨*Deklaracije promenljivih*⟩]
⟨*Glavni deo programa*⟩

⟨*Definicije konstanti*⟩ ≡ `const`
⟨*ime*$_1$⟩ `=` ⟨*vrednost*$_1$⟩`;`
⋮
⟨*ime*$_k$⟩ `=` ⟨*vrednost*$_k$⟩`;`

⟨*Deklaracije promenljivih*⟩ ≡ `var`
⟨*Spisak*$_1$⟩ `:` ⟨*Tip*$_1$⟩`;`
⋮
⟨*Spisak*$_k$⟩ `:` ⟨*Tip*$_k$⟩`;`

⟨*Spisak*⟩ ≡ ⟨*ime*⟩ | ⟨*Niz imena*⟩

⟨*Niz imena*⟩ ≡ ⟨*ime*$_1$⟩`,` ⟨*ime*$_2$⟩`, ...,` ⟨*ime*$_k$⟩

⟨*Tip*⟩ ≡ `real` | ...(ima ih još mnogo!)

⟨*Glavni deo programa*⟩ ≡ `begin`
⟨*Naredba*$_1$⟩`;`
⋮
⟨*Naredba*$_k$⟩
`end.`

⟨*Naredba*⟩ ≡ ⟨*Naredba dodele*⟩ | ...(ima ih još mnogo!)

⟨*Naredba dodele*⟩ ≡ ⟨*ime*⟩ `:=` ⟨*izraz*⟩

Glava 2

Algebarski izrazi, “if” kontrolna struktura

U ovoj glavi upoznaćemo se sa načinom na koji se u programskom jeziku Paskal formiraju algebarski izrazi. Problem sa formiranjem algebarskih izraza nastaje zato što Paskal ne poseduje specijalne simbole kao što je $\sqrt{\ }$ i zato što algebraski izraz mora biti “spakovan u liniju”. Zato moramo biti pažljivi kada neki algebarski izraz zapisujemo u većini programskih jezika uključujući i Paskal: ponekad je potrebno dodati zagrade koje se u izrazu podrazumevaju!

Sledeći problem nastaje na semantičkom nivou: algebraske operacije nisu definisane za sve vrednosti koje se mogu pojaviti kao njihovi argumenti. Na primer, ne možemo deliti nulom, niti u skupu realnih brojeva možemo odrediti koren iz negativnog broja. Potrebno je, dakle, kontrolisati tok programa i tako razdvojiti slučajeve u kojima je vrednost algebarskog izraza definisana od slučajeva u kojima algebarski izraz nije određen. Tome služi “if” kontrolna struktura. Upoznaćemo se i sa blokom kao sa posebnim načinom da se niz naredbi programskog jezika Paskal grupiše u jednu sintaksnu celinu.

2.1 Realni brojevi

Promenljiva tipa `real` može da zapamti jedan realan broj. Realan broj se zapisuje kao decimalan broj, koji može biti i pozitivan i negativan. Primeri realnih brojeva:

```
1.4142    -0.618    2341.0    -26
```

☞ *Primetite još jednom da se decimalni brojevi pišu koristeći* ***decimalnu tačku,*** *prema engleskom pravopisu, a ne decimalni zarez kako nalaže naš pravopis!*

U fizici i tehnici se veoma često javljaju brojevi u tzv. *eksponencijalnoj* notaciji, kao što su, recimo brojevi $-0.61 \cdot 10^7$ i $0.2 \cdot 10^{-13}$. U programskom jeziku Paskal se "... puta deset na ..." piše pomoću jednog `e` (od *eksponent*). Primeri su dati pored.

Matematika	$\rightarrow$	Paskal
$1.45 \cdot 10^{12}$	$\rightarrow$	`1.45e12`
$-0.61 \cdot 10^{7}$	$\rightarrow$	`-0.61e7`
$0.2 \cdot 10^{-13}$	$\rightarrow$	`0.2e-13`
$-12.65 \cdot 10^{-6}$	$\rightarrow$	`-12.65e-6`

2.2 Algebarski izrazi

Programski jezik Paskal poznaje sledeće algebarske operacije nad realnim brojevima:

Operacija	Objašnjenje	Primer
`+`	sabiranje	`x + 1.14`
`-`	oduzimanje	`x - 1.14`
	promena znaka	`-x + 2`
`*`	množenje	`x * y`
`/`	deljenje	`x / 1.036`
`sqrt`	kvadratni koren	`sqrt(x / y)`
`sqr`	"na kvadrat"	`sqr(x + 1)`
`abs`	apsolutna vrednost	`abs(x - 2)`

Ime funkcije `sqrt` potiče od engleskog *square root* = kvadratni koren; ime funkcije `sqr` potiče od engleskog *squared* = kvadrirano (na kvadrat); ime funkcije `abs` potiče od engleskog *absolute value* = apsolutna vrednost.

Kao i u matematici, operacije `*` i `/` imaju veći prioritet od operacija `+` i `-`. To znači da će se prvo izvršiti operacije množenja i deljenja, pa tek onda operacije sabiranja i oduzimanja. Evo primera:

Paskal izraz	Odgovarajući matematički izraz
`1 - x / y`	$1 - \frac{x}{y}$
`x * y - 4`	$xy - 4$
`1.12 + x / y - 3 * z`	$1.12 + \frac{x}{y} - 3z$

Operacije sabiranja i oduzimanja imaju isti prioritet, što je i očekivano i ovo ne dovodi do čudnih situacija. Operacije množenja i deljenja imaju takođe isti prioritet, što je i očekivano, *ali ovo može da dovede do čudnih situacija.* Na primer:

Paskal izraz	Tačno	**Pogrešno**
`x / y * z`	$\frac{x}{y} \cdot z$	$\frac{x}{y \cdot z}$
`x / y / z`	$\frac{\frac{x}{y}}{z} = \frac{x}{yz}$	$\frac{x}{\frac{y}{z}}$

Zagrade se koriste kao u matematici. Ako nismo sigurni kako će Paskal prevodilac shvatiti neki izraz, postavljanjem zagrada možemo obezbediti da se izraz računa tačno onako kako želimo. Na primer:

Paskal izraz	Odgovarajući matematički izraz
`(1 - x) / y`	$\frac{1-x}{y}$
`x * (y - 4)`	$x(y-4)$
`(1.12 + x) / (y - 3 * z)`	$\frac{1.12+x}{y-3z}$
`x / (y * z)`	$\frac{x}{yz}$

Problem sa svim programskim jezicima je u tome što algebraski izraz mora biti "spakovan u liniju". Zato moramo biti pažljivi kada neki algebarski izraz zapisujemo u programskom jeziku: *ponekad je potrebno dodati zagrade koje se u izrazu podrazumevaju!* Primeri:

Algebarski izraz	Odgovarajući Paskal izraz	**Pogrešno**
$\frac{x-1}{2-x}$	`(x - 1) / (2 - x)`	`x - 1 / 2 - x`
$\frac{x}{3z}$	`x / (3 * z)`	`x / 3 * z`
$\frac{1-\frac{x}{y}}{3z-2}$	`(1 - x/y) / (3*z - 2)`	`1 - x/y / (3*z - 2)`

Pogledajmo još nekoliko primera upotrebe funkcija `abs`, `sqr` i `sqrt`:

Algebarski izraz	Odgovarajući Paskal izraz
$\sqrt{1-\frac{x}{y}+z}$	`sqrt(1 - x/y) + z`
$\sqrt{\frac{\lvert x-1\rvert}{x+1}}$	`sqrt(abs(x - 1) / (x + 1))`
$\frac{2-x}{y^2}$	`(2 - x) / sqr(y)`
$\frac{2-x}{(y-1)^2}$	`(2 - x) / sqr(y - 1)`
$\frac{\sqrt{2-x}}{\lvert y-1\rvert}$	`sqrt(2 - x) / abs(y - 1)`

Programski jezik Paskal poznaje još neke standardne matematičke funkcije koje ćete učiti kasnije. Zato sada navodimo samo spisak bez detaljnih objašnjenja.

Paskal	Matematika	Ime
`sin(x)`	$\sin x$	sinus
`cos(x)`	$\cos x$	kosinus
`tan(x)`	$\mathrm{tg}\, x$	tangens
`arctan(x)`	$\mathrm{arctg}\, x$	arkus tangens
`exp(x)`	e^x	$(e = 2.71\ldots)$
`ln(x)`	$\ln x$	prirodni logaritam

Primer. Napisati Paskal program koji računa vrednost izraza

$$\frac{x+y}{x-y},$$

gde su x i y realni brojevi.

```
program Izraz1;
var
    x, y, rez : real;
begin
    writeln('Unesi x i y');
    readln(x, y);
    rez := (x + y)/(x - y);
    writeln('Rez = ', rez)
end.
```

Primer. Napisati Paskal program koji računa vrednost izraza

$$-2x+\sqrt{\frac{1-x}{y(z^2+2)}},$$

gde su x, y i z realni brojevi.

```
program Izraz2;
var
    x, y, z, rez : real;
begin
    writeln('Unesi x, y, z');
    readln(x, y, z);
    rez := -2 * x + sqrt( (1 - x)/
             (y * (sqr(z) + 2)) );
    writeln('Rez = ', rez)
end.
```

☞ *Primetimo da je moguće jednom naredbom* `readln` *učitati više vrednosti.*

Ponekad se u većem izrazu neki manji izraz javi nekoliko puta. Rad programa se ubrzava ako se taj podizraz izračuna posebno i smesti u neku pomoćnu promenljivu, koju potom pozivamo u većem izrazu.

Primer. Napisati Paskal program koji računa vrednost izraza

$$\frac{-(x^2+xy+y^2)+2z}{x^2+xy+y^2-z}$$

gde su x, y i z realni brojevi.

```
program MaliTrik;
var
    x, y, z, d, rez : real;
begin
    writeln('Unesi x, y, z');
    readln(x, y, z);
    d := sqr(x) + x * y + sqr(y);
    rez := (-d + 2 * z)/(d - z);
    writeln('Rez = ', rez)
end.
```

2.3 Formatirani ispis realnih brojeva

Ako izvršimo program naveden pored, on će ispisati vrednost promenljive x ovako:

```
-3.666666666666667e+0000
```

što je korektno, ali ružno. Zato u programskom jeziku Paskal postoji način da se realni brojevi ispisuju na lepši i čitkiji način.

```
program RuzanIspis;
var
    x : real;
begin
    x := -11/3;
    writeln(x)
end.
```

Naredba `writeln` dozvoljava da se iza realne promenljive dodaju modifikatori koji kažu prevodiocu kako da formatira broj prilikom ispisa. Ispis realne promenljive sa modifikatorom izgleda ovako:

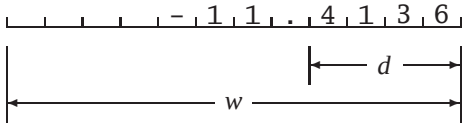
`writeln(x :` w `:` d`)`

gde w predstavlja *širinu polja*, odnosno, ukupan broj mesta koji sme biti zauzet za ispis broja, a d je broj decimalnih mesta. Unutar polja širine w broj će biti poravnat po desnoj ivici. Dakle, ukoliko je w veći od stvarno potrebnog broja mesta, praznine će biti dodate sleva. Broj će biti zaokružen na dati broj decimalnih mesta.

Na primer, ako želimo da ispišemo broj -11.4135824 uz parametre (modifikator) $w = 12$, $d = 4$, to možemo učiniti komandom

```
writeln(-11.4135824 : 12 : 4)
```

Na monitoru ćemo dobiti (naravno, sve prateće strelice i tarabe se neće videti):



Evo još nekoliko primera. Sledeća tabela pokazuje kako izgleda ispis vrednosti promenljive x uz razne modifikatore, nakon komande `x := -11/3`:

```
writeln(x)        -3.666666666666667e+0000
writeln(x:10:3)   _____-3.667
writeln(x:5:1)    __-3.7
writeln(x:2:0)    -4
writeln(x:1:0)    -4
writeln(x:0:0)    -3.666666666666667e+0000
```

Kviz.

1. Koju vrednost će imati promenljive x, m, n nakon izvršavanja sledećih naredbi:

(*a*) `x := 5.5; x := x + 3.5`

(*b*) `m := 35; m := m + 1; m := m - 32`

(*c*) `m := 5; n := 6; m := m*n; n := n*m`

2. Koju vrednost će imati promenljive a, b, c nakon izvršavanja sledećih naredbi:

```
a := 1; b := 2; c := 3;
a := b; b := c; c := a
```

3. Koju vrednost će imati promenljive x, y nakon izvršavanja sledećih naredbi:

```
x := 5; y := 7;
t := x; x := y; y := t
```

4. Šta ispisuje program pored kada je
(*a*) `x = 3, y = 7;`
(*b*) `x = 9, y = 2`?
Šta radi taj program?

```
program Kviz2;
var
    x, y : real;
begin
    readln(x); readln(y);
    x := x + y;
    y := x - y;
    x := x - y;
    writeln(x); writeln(y)
end.
```

Zadaci.

2.1. Napisati Paskal program koji računa rastojanje dve tačke u ravni. Ako tačka A ima koordinate (x_1, y_1), a tačka B koordinate (x_2, y_2), onda je rastojanje tačaka A i B dato sa

$$d(A,B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

2.2. Napisati program koji računa intenzitet sile privlačenja dva tela čije mase su m_1 i m_2, a koja se nalaze na rastojanju r. Dakle, od korisnika treba učitati realne brojeve m_1, m_2 i r i izračunati F koristeći se poznatim Njutnovim obrascem

$$F = \gamma \frac{m_1 m_2}{r^2},$$

gde je $\gamma = 6.67 \cdot 10^{-11}$ (nekih jedinica).

2.3. Napisati Paskal program koji računa vrednosti sledećih algebarskih izraza:

(a) $\dfrac{-x+y}{2}$ (c) $2xyz + \dfrac{(x+y)(x-2y)}{x^2+y^2+3}$

(b) $\dfrac{1}{\frac{1}{x}+\frac{1}{y}}$ (d) $\dfrac{(x+y)(y+z)(z+x)}{(2x-y)(3y-z)(4z-x)}$

2.4. Napisati Paskal program koji učitava realne brojeve a_1, b_1, c_1, a_2, b_2, c_2 i potom računa i ispisuje rešenja x i y sledećeg sistema linearnih jednačina

$$\begin{aligned} a_1 x + b_1 y &= c_1 \\ a_2 x + b_2 y &= c_2. \end{aligned}$$

(Uputstvo: sistem prvo rešiti "peške", srediti dobijene izraze, i onda samo napisati program koji računa rešenja po dobijenim obrascima.)

2.5. Napisati Paskal program koji računa vrednost sledećeg izraza:

$$\frac{1}{\frac{-x}{x^2+y^2+1} + \frac{y}{x^2+y^2+1}} \cdot \sqrt{x^2+y^2+1}.$$

2.6. Napisati Paskal program koji računa vrednost sledećeg izraza:

$$\frac{(x^2-yz)^2 + \sqrt{1-x^2+yz}}{|x^2-yz|+1}.$$

2.7. Kvadratna jednačina je jednačina oblika $ax^2+bx+c=0$, gde su $a\neq 0$, b i c proizvoljni realni brojevi. Svaka kvadratna jednačina ima dva rešenja koja se računaju ovako:

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a} \quad \text{i} \quad x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}.$$

Napisati Paskal program koji od korisnika učitava relane brojeve a, b, c i računa rešenja kvadratne jednačine.

2.4 "If" kontrolna struktura

Podsetimo se Paskal programa koji računa vrednost izraza

$$\frac{x+y}{x-y},$$

gde su x i y realni brojevi (videti pored). Ako su x i y različiti, program radi lepo i računa sve kako treba.

```
program Izraz1;
var
  x, y, rez : real;
begin
  writeln('Unesi x');
  readln(x);
  writeln('Unesi y');
  readln(y);
  rez := (x + y)/(x - y);
  writeln('Rez = ', rez)
end.
```

Međutim, ako programu damo isti broj za x i y, prilikom računanja vrednosti promenljive `rez` dolazi do deljenja nulom. Računar tada prijavljuje grešku (*run time error*) i prekida sa radom. U žargonu se kaže da je program "pukao". Primer jednog "pucanja" je dat pored.

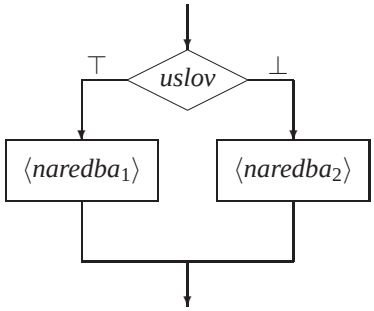
MONITOR:

```
Unesi x
3
Unesi y
3
Runtime error 107: division by zero
```

Nijedan Pravi Programer ne sme da dozvoli da mu program puca. Zato svaki programski jezik ima način da se proveri da li je ispunjen uslov za izvršenje neke operacije. U našem slučaju, pre računanja vrednosti promenljive `rez` treba proveriti da li je $x=y$. Ako jeste onda programer treba da se pobuni, a ako nije onda se može preći na računanje. Taj mehanizam se zove *"if" kontrolna struktura*.

"If" kontrolna struktura se može javiti u raznim oblicima, a mi ćemo krenuti od najjednostavnijeg. Osnovni oblik "if" kontrolne strukture izgleda ovako:

⟨*If naredba – osnovni oblik*⟩ ≡
```
if ⟨uslov⟩ then
    ⟨naredba1⟩
else
    ⟨naredba2⟩;
```



a tok programa koji sadrži "if" predstavljen je slikovito pored.

Kada naiđe na "if":

- program prvo proveri uslov;
- ako je uslov ispunjen, izvrši ⟨*naredbu*$_1$⟩, preskoči ⟨*naredbu*$_2$⟩ i nastavi sa radom;
- ako uslov nije ispunjen, preskoči ⟨*naredbu*$_1$⟩, izvrši ⟨*naredbu*$_2$⟩ i nastavi sa radom.

☞ *Ovo se zove kontrolna struktura zato što kontroliše tok programa: ako je ispunjen uslov, program radi na jedan način, a ako uslov nije ispunjen onda program radi na drugi način.*

Sada ćemo program `Izraz1` napisati pošteno. U novoj varijanti se ništa nije promenilo. Jedina novina je da računanje radimo onda kada to ima smisla. U ostalim slučajevima *program sam* prijavi grešku, a ne računar od koga smo zatražili da deli nulom. Kada učitamo `x` i `y`, prvo ćemo proveriti da li su jednaki. Ako jesu, prijavićemo da nešto ne valja i ispisati bilo šta kao rezultat, a ako nisu jednaki, izračunaćemo vrednost izraza.

```
program PonovoIzraz1;
var
  x, y, rez : real;
begin
  writeln('Unesi x');
  readln(x);
  writeln('Unesi y');
  readln(y);
  rez := 0;
  if x = y then
    writeln('GRRRESKA!')
  else
    rez := (x + y)/(x - y);
  writeln('Rez = ', rez)
end.
```

Ono što nam malo kvari sreću u ovom trenutku je činjenica da program svakako ispisuje neki rezultat, nezavisno od toga da li je izraz definisan za unete vrednosti ili ne. Sa ovim problemom ćemo se ubrzo uhvatiti u koštac!

2.5 Jednostavni uslovi

Uslov koji se javlja u "if" može biti jednostavan ili složen. Za sada ćemo koristiti samo jednostavne uslove.

Jednostavnim uslovima u "if" kontrolnoj strukturi proveravaju se jednostavniji odnosi među veličinama: da li su dva broja jednaka, različita, da li je neki od njih manji ili veći od onog drugog, manji ili jednak, veći ili jednak. Tabelica je data pored.

Matematika	Paskal	Primer
$=$	`=`	`x - y = 0`
$\neq$	`<>`	`x <> y`
$<$	`<`	`x < 5`
$>$	`>`	`y > x + 2`
$\leqslant$	`<=`	`x <= 5`
$\geqslant$	`>=`	`y >= x + 2`

Primer. Napisati Paskal program koji računa vrednost sledeće funkcije:

$$f(x) = \begin{cases} 1 - x + x^3, & x \leqslant 5 \\ \dfrac{2x-3}{x-5}, & \text{inače} \end{cases}$$

Za ispis rezultata koristiti ukupno 10 mesta, a rezultat zaokružiti na dve decimale.

```
program KomplikovanaFja;
var
   x, f : real;
begin
   readln(x);
   if x <= 5 then
      f := 1 - x + x*x*x
   else
      f := (2*x - 3)/(x - 5);
   writeln('f(x) = ', f:10:2)
end.
```

Primer. Napisati Paskal program koji određuje i štampa minimum dva realna broja.

```
program MinDvaBroja;
var
   a, b, min : real;
begin
   readln(a, b);
   if a < b then
      min := a
   else
      min := b;
   writeln('min = ', min)
end.
```

Zadaci.

2.8. Napisati Paskal program koji računa vrednost sledećih funkcija:

$$(a)\quad f(x,y) = \begin{cases} |x| + |y| - 6xy, & x < y, \\ x^2 + y^2, & \text{inače;} \end{cases} \qquad (b)\quad \max(|x-y|, |u-v|).$$

2.9. Napisati Paskal program koji računa vrednost sledeće funkcije:

$$(a)\quad \frac{\min(x,y)+0.5}{1+\min(x,y)^2}; \qquad (b)\quad \frac{\min(x,y)+0.5}{1+\max(x,y)^2}.$$

2.10. Napisati Paskal program koji računa vrednost sledeće funkcije:

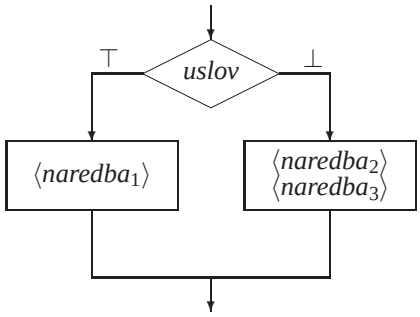
$$f(x,y)=\begin{cases} \min(x,y), & y \geqslant 0, \\ \max(x^2,|y|), & \text{inače.} \end{cases}$$

2.11. Napisati Paskal program koji od korisnika učitava tri realna broja i određuje i štampa njihov minimum.

2.12. Napisati Paskal program koji od korisnika učitava četiri realna broja i određuje i štampa njihov minimum.

2.6 Blok

Često imamo potrebu da u jednoj ili obe grane "if" kontrolne strukture izvršimo više naredbi. Ta situacija je prikazana na slici pored kada u slučaju da uslov nije tačan želimo da izvršimo dve naredbe. Mehanizam programskog jezika Paskal koji nam to omogućuje zove se *blok*.



Blok je niz komandi razdvojenih tačka-zarezima kome na početku stoji rezervisana reč `begin`, a na kraju rezervisana reč `end`. Ovu konstrukciju smo već koristili, samo što nismo znali da se to tako zove! Blok smo do sada videli mnogo puta, na primer, glavni deo svakog programa je jedan blok. Kao usputnu napomenu možemo sada reći da Paskal program, zapravo, ima strukturu koja je opisana pored →

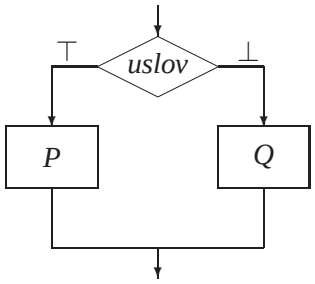
⟨*Blok*⟩ ≡ `begin`
⟨*Naredba*$_1$⟩`;`
⟨*Naredba*$_2$⟩`;`
⋮
⟨*Naredba*$_k$⟩
`end`

⟨*Paskal program*⟩ ≡
`program` ⟨*ime*⟩`;`
[⟨*Definicije konstanti*⟩]
[⟨*Deklaracije promenljivih*⟩]
⟨*Blok*⟩ `.`

Blok se može pojaviti i u "if" kontrolnoj strukturi kada je unutar jedne "grane" potrebno izvršiti više naredbi. Opštiji oblik "if" kontrolne strukture izgleda ovako

⟨*If naredba – opštiji oblik*⟩ ≡

```
if ⟨uslov⟩ then
    P
else
    Q ;
```



gde svaki od *P* i *Q* može da bude *ili tačno jedna naredba ili blok.*

Primer. Napisati Paskal program koji računa vrednost izraza

$$x + y\sqrt{x^2 - y^2 + 1}$$

gde su *x* i *y* neki realni brojevi.

```
program Primer;
var
   x, y, s, rez : real;
begin
   readln(x, y);
   s := sqr(x) - sqr(y) + 1;
   if s < 0 then
      writeln('Koren iz neg broja!')
   else
      begin
         rez := x + y * sqrt(s);
         writeln('Rezultat: ', rez)
      end
end.
```

Naredbe koje se javljaju u bloku mogu, naravno, biti proizvoljne naredbe, pa i druge "if" kontrolne strukture. Na taj način se može postići finija kontrola programa što ćemo koristiti za računanje komplikovanijih izraza.

Primer. Napisati Paskal program koji računa vrednost izraza

$$1 + \sqrt{\frac{x+y}{x-y}}$$

gde su *x* i *y* neki realni brojevi.

```
program Primer;
var
  x, y, s, rez : real;
begin
  readln(x, y);
  if x = y then
    writeln('Deljenje nulom!')
  else
    begin
      s := (x + y)/(x - y);
      if s < 0 then
        writeln('Koren iz neg br!')
      else
        begin
          rez := 1 + sqrt(s);
          writeln('Rez: ', rez)
        end
    end
end.
```

Ovo nam je trebalo još od samog početka priče o "if" kontrolnoj strukturi kada smo pisali program `Izraz1`! Program `PonovoIzraz1` je malo popravio stvari tako da više nije postojala opasnost da će pući, ali nije bio elegantan zato što je uvek ispisivao nešto, čak i u slučajevima kada izraz nije bio definisan. Ono što smo želeli od samog početka je zapravio program `NajzadIzraz1` koji je dat na Sl. 2.1.

Zadaci.

2.13. (*a*) Šta će program ispod ispisati ako mu se kao vrednost za `x` unese 9? A ako se unese -8?

```
program MojPrviBlok;
var
  x, y, z : real;
begin
  writeln('Unesi x');
  readln(x);
  if x > 0 then
    begin
      y := sqrt(x);
      z := 2 * y - 1
    end
  else
    z := 3 * x;
  writeln(z)
end.
```

(*b*) Koju vrednost će imati promenljiva `x` nakon sledećeg programskog fragmenta:

```
x := 1; y := -5;
if abs(y) > 2 then
    begin
        if y < 0 then
            x := 0
        else
            x := 2
    end
else
    x := 1;
```

2.14. Napisati ponovo program koji računa rešenja sistema dve linearne jednačine sa dve nepoznate (Zadatak 2.4). Rešenja sistema ispisati na tri decimale.

2.15. Napisati ponovo program koji računa rešenja kvadratne jednačine (Zadatak 2.7). Prilikom ispisa rešenja zaokružiti na dve decimale.

2.16. Napisati Paskal program koji računa vrednost sledećeg izraza:

$$(a)\quad \frac{x^2+4y}{x+y} \qquad (c)\quad 2+\sqrt{z+\frac{x-z}{y-z}}$$

$$(b)\quad \sqrt{1+x-y^2} \qquad (d)\quad \sqrt{x+\sqrt{y}}$$

Verzija koja puca:

```
program Izraz1;
var
  x, y, rez : real;
begin
  writeln('Unesi x');
  readln(x);
  writeln('Unesi y');
  readln(y);
  rez := (x + y)/(x - y);
  writeln('Rez=', rez)
end.
```

Neelegantna verzija:

```
program PonovoIzraz1;
var
  x, y, rez : real;
begin
  writeln('Unesi x');
  readln(x);
  writeln('Unesi y');
  readln(y);
  rez := 0;
  if x = y then
    writeln('GRRRESKA!')
  else
    rez := (x + y)/(x - y);
  writeln('Rez = ', rez)
end.
```

Pravo rešenje:

```
program NajzadIzraz1;
var
  x, y, rez : real;
begin
  writeln('Unesi x');
  readln(x);
  writeln('Unesi y');
  readln(y);
  if x = y then
    writeln('GRRRESKA!')
  else
    begin
      rez := (x + y)/(x - y);
      writeln('Rez = ', rez)
    end
end.
```

Slika 2.1: Tri verzije programa koji računa izraz $\frac{x+y}{x-y}$

2.17. Napisati Paskal program koji računa vrednost sledeće funkcije:

$$(a)\quad f(x)=\begin{cases} x^2+4x, & x\leqslant x^2 \\ \dfrac{1}{x-x^2}, & \text{inače} \end{cases} \qquad (b)\quad g(x)=\begin{cases} \dfrac{2x}{x-4}, & x>0 \\ \dfrac{1+\sqrt{-x}}{x+6}, & \text{inače} \end{cases}$$

2.18. Napisati Paskal program koji računa vrednost sledeće funkcije:

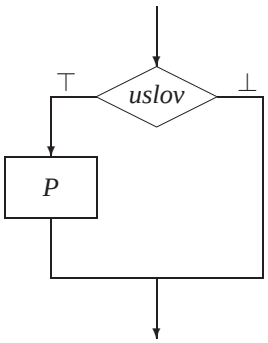
$$\frac{\min(x+y,\sqrt{x})+2y}{16-\max(x,y)^2}$$

2.7 Razni oblici "if" kontrolne strukture

Osim osnovnog oblika, "if" kontrolna struktura se može javiti i u drugim oblicima koji nastaju ili skraćivanjem osnovnog oblika, ili nadovezivanjem jednostavnijih oblika.

Skraćeni oblik "if" kontrolne strukture izgleda ovako:

`if` ⟨*uslov*⟩ `then`
P ;



Kao i ranije, *P* može da bude ili tačno jedna naredba ili blok. Kada naiđe na "kratki if", program se ponaša na sledeći način:

- proveri uslov;
- ako je uslov ispunjen izvrši *P* i nastavi sa radom;
- ako uslov nije ispunjen, preskoči *P* i nastavi sa radom.

Višestruki "if" u osnovnom obliku je prikazan pored. Svaki P_i je ili tačno jedna komanda ili blok. Evo kako radi višestruki "if". Računar proveri prvi uslov. Ako je on ispunjen, izvrši P_1, preskoči sve ostale P_i-ove, preskoči Q i nastavi sa naredbom iza "if"-a. Ako prvi uslov nije ispunjen, proverava se drugi uslov. Ako je on ispunjen, izvrši se P_2, preskoče se ostali P_i-ovi, preskoči se Q i nastavi se sa izvršavanjem naredbe iza "if"-a. I tako dalje. Ako nijedan uslov nije ispunjen, izvrši se Q.

```
if ⟨uslov1⟩ then
      P1
else if ⟨uslov2⟩ then
      P2
  ⋮
else if ⟨uslovk⟩ then
      Pk
else
      Q ;
```

Višestruki "if" u skraćenom obliku je prikazan pored. Prilikom izvršavanja ove komande računar se ponaša na isti način kao u prethodnom slučaju. Jedina razlika je u tome da ukoliko nijedan uslov nije ispunjen, ne izvrši se ništa, već se nastavi sa radom od naredbe koja je iza "if"-a. Tok programa prilikom izvršavanja ova dva oblika "if" kontrolne strukture prikazan je na Sl. 2.2.

```
if ⟨uslov1⟩ then
      P1
else if ⟨uslov2⟩ then
      P2
  ⋮
else if ⟨uslovk⟩ then
      Pk ;
```

Primer. Napisati Paskal program koji računa vrednost sledeće funkcije:

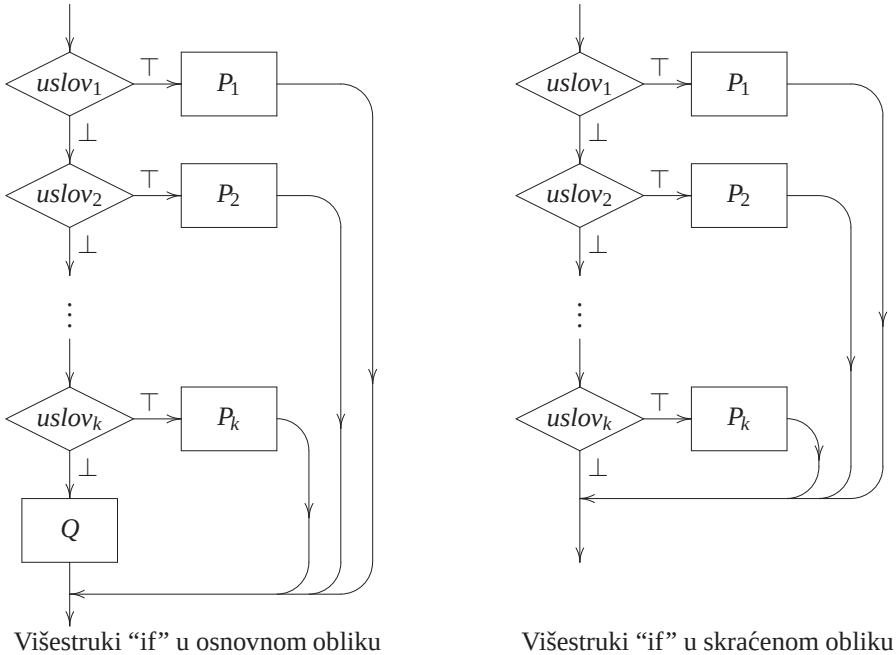
$$f(x) = \begin{cases} \dfrac{2|x|+1}{4x-5}, & x < \dfrac{5}{4} \\ 0, & x = \dfrac{5}{4} \\ \dfrac{2x-1}{4x-5}, & x > \dfrac{5}{4} \end{cases}$$

```
program PrimerVisestrukogIf;
var
  x, f : real;
begin
  readln(x);
  if x < 5/4 then
    f := (2*abs(x) + 1)/(4*x - 5)
  else if x = 5/4 then
    f := 0
  else
    f := (2*x - 1)/(4*x - 5);
  writeln('f(x) = ', f)
end.
```

2.8 "Dangling else"

Postojanje kratkog "if"-a je veoma korisno, ali dovodi do jedne anomalije koja zahteva da se posebno prodiskutuje i da se reguliše posebnim pravilom. Taj fenomen se na engleskom zove *dangling else*[1].

[1] viseće else, ili else-visuljak



Višestruki “if” u osnovnom obliku

Višestruki “if” u skraćenom obliku

Slika 2.2: Višestruki “if” u osnovnom i skraćenom obliku

U kratkom “if” se nakon ključne reči `then` može pojaviti jedna naredba koja ponovo može biti kratki “if”. Ako ugnezdimo nekoliko takvih kratkih “if”-ova i završimo jednim potpunim “if”-om, dobijamo jednu komplikovanu naredbu. Primer jedne takve naredbe je dat pored.

```
if a > 0 then
  if b > 0 then
    if c > 0 then
      writeln('P')
    else
      writeln('Q');
```

Sada se postavlja pitanje: kom “if”-u pripada `else` na kraju? Gornja naredba se može interpretirati na razne načine, recimo:

```
if a > 0 then
  if b > 0 then
    if c > 0 then
      writeln('A')
    else {za 3. if}
      writeln('B');
```

```
if a > 0 then
  if b > 0 then
    if c > 0 then
      writeln('A')
  else {za 2. if}
    writeln('B');
```

```
if a > 0 then
  if b > 0 then
    if c > 0 then
      writeln('A')
else {za 1. if}
  writeln('B');
```

Naravno, Paskal prevodilac ne može da brojeći razmake pogodi šta je programer

imao na umu! Zato se *dangling else* fenomen reguliše posebnim pravilom:

☞ *U ovoj situaciji,* `else` *pripada poslednjem (njemu najbližem)* `if`*-u.*

Dakle, u prethodnom primeru ispravna interpretacija je prva, kod koje `else` pripada trećem `if`-u.

Kviz.

1. Posmatrajmo program pored. Šta će program ispisati ako mu se kao vrednost za x unese 9? A ako se unese -4?

```
program MojPrviKratkiIf;
var
  x, y, z : real;
begin
  readln(x);
  if x > 0 then
    begin
      y := sqrt(x);
      x := 2 * y - 1
    end;
  z := 3 * x;
  writeln(z)
end.
```

2. Koju vrednost će imati promenljiva x nakon izvršenja programskog fragmenta koji je dat pored?

```
 x := 10; y := -4;
 if abs(y) > 3 then
   begin
     if y < 0 then
       x := 3
   end;
```

3. Šta će ispisati sledeći programski fragment kada promenljive x, y, z, u imaju sledeće vrednosti, redom:
(a) 0, 3, 5, 7;
(b) 2, -1, 6, 6;
(c) 2, 2, 6, 2;
(d) 2, 2, 2, 2;
(e) 2, 2, 2, 4?

```
if x > 1 then
  if y > 0 then
    if z < 5 then
      if u > 3 then
        writeln('A')
else
  writeln('B');
writeln('C');
```

Zadaci.

2.19. Sledeći programski fragment napisati pomoću samo jedne `if` naredbe:

```
if a > b then c := 1;
if a > b then d := 2;
if a <= b then c := 3;
if a <= b then d := 4;
```

2.20. Napisati Paskal program koji računa vrednost sledeće funkcije:

$$f(x,y) = \begin{cases} \dfrac{-4xy}{\sqrt{x^2-y^2}}, & |x| > |y| \\ \dfrac{x^2-4y}{\sqrt{y^2-x^2}}, & |x| < |y| \\ \dfrac{-x}{\sqrt{1+x^2+y^2}}, & \text{inače} \end{cases}$$

2.21. Napisati Paskal program koji računa vrednost sledeće funkcije:

$$g(x,y) = \begin{cases} \dfrac{8-xy}{\sqrt{\dfrac{x-1}{1-y}}}, & \dfrac{x-1}{1-y} > 0 \\ 5x, & y = 1 \\ 2-y, & x = 1, y \neq 1 \\ -\dfrac{8-xy}{\sqrt{\dfrac{1-x}{1-y}}}, & \dfrac{x-1}{1-y} < 0 \end{cases}$$

2.22. Napisati ponovo program koji računa rešenja kvadratne jednačine (Zadatak 2.7). Ako je $b^2-4ac<0$ ispisati poruku "Rešenja nisu realna". Ako je $b^2-4ac=0$ tada su oba rešenja jednaka. Ispisati poruku "Rešenja su jednaka" i potom ispisati vrednost rešenja. Ako je $b^2-4ac>0$ ispisati oba rešenja. U svim slučajevima rešenja zaokružiti na dve decimale.

2.23. (Progresivno oporezivanje) U jednoj državi se porez na zarade obračunava na sledeći način: na godišnje zarade koje su manje ili jednake od donje granice (`dgr`) se ne plaća nikakav porez; na godišnje zarade koje su strogo veće od donje granice (`dgr`) i manje su ili jednake od gornje granice (`ggr`) obračunava se porez po nižoj kamatnoj stopi (`nks`), ali samo na deo koji je

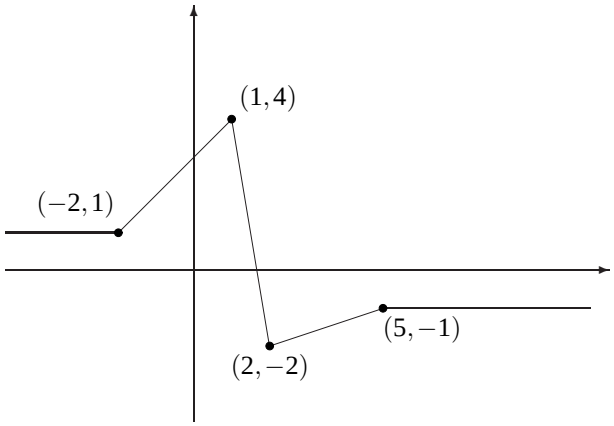
iznad dgr; a na godišnje zarade koje su veće od gornje granice (ggr) porez se obračunava tako što se na deo zarade ggr − dgr porez obračunava po nižoj kamatnoj stopi (nks), dok se na deo zarade koji je iznad ggr porez obračunava po višoj kamatnoj stopi (vks).

Preciznije, ako je z godišnja zarada, onda se porez p obraŭnava ovako:

- ako je $\mathtt{z} \leqslant \mathtt{dgr}$ onda je $\mathtt{p} = 0$;
- ako je $\mathtt{dgr} < \mathtt{z} \leqslant \mathtt{ggr}$ onda je $\mathtt{p} = \dfrac{(\mathtt{z} - \mathtt{dgr}) * \mathtt{nks}}{100}$;
- ako je $\mathtt{z} > \mathtt{ggr}$ onda je $\mathtt{p} = \dfrac{(\mathtt{ggr} - \mathtt{dgr}) * \mathtt{nks} + (\mathtt{z} - \mathtt{ggr}) * \mathtt{vks}}{100}$.

Napisati Paskal program koji od korisnika učtava parametre poreskog sistema dgr, ggr, nks i vks, potom zaradu na godišnjem nivou z nekog radnika, i onda računa i štampa iznos poreza p.

2.24. Napisati Paskal program koji od korisnika učitava realan broj x i računa vrednost funkcije $f(x)$ koja je data grafikom:



2.9 Složeni uslovi

Često je potrebno ispitati da li važe dva uslova istovremeno pre nego što se pređe na računanje vrednosti nekog izraza. Da se ne bismo petljali sa ugnježdenim "if"-ovima, postoji način da se od dva ili više jednostavnih uslova napravi jedan složeni uslov: jednostavni uslovi se stave u zagrade i povežu se logičkim veznicima `and`, `or` ili `not`. Pogledajmo par primera:

Matematika	Paskal
$x < 5 \wedge y \geqslant 0$	`(x < 5) and (y >= 0)`
$x < 5 \vee x > 9$	`(x < 5) or (x > 9)`
$\neg(x > 3) \wedge y = 5$	`not(x > 3) and (y = 5)`
$(x < 2 \wedge y = 4) \vee (x > 3 \wedge y > 3)$	`((x<2) and (y=4)) or ((x>3) and (y>3))`
$(x < 2 \vee y = 4) \wedge (x > 3 \vee y > 3)$	`((x<2) or (y=4)) and ((x>3) or (y>3))`

Logički veznici u programskom jeziku Paskal imaju različite prioritete. Veznik `not` ima najviši prioritet zato što je on unarni operator (napada samo jedan izraz). Osim toga, veznik `and` ima viši prioritet od veznika `or` (`and` se ponaša kao neka vrsta množenja, a `or` se ponaša kao neka vrsta sabiranja). Ako želimo da promenimo redosled dejstva logičkih veznika, slobodno možemo da koristimo zagrade, kako je to pokazano u poslednjim primeru iz prethodne tabele.

Posebno treba obratiti pažnju na sledeće situacije, kod kojih se (matematički korektan) produženi zapis mora predstaviti u obliku konjunkcije dva prosta izraza:

Matematika	Paskal ispravno	Paskal **neispravno**
$x = y = z$	`(x = y) and (y = z)`	`x = y = z`
$1 < x < 5$	`(1 < x) and (x < 5)`	`1 < x < 5`

Primer. Napisati Paskal program koji računa vrednost izraza

$$\sqrt{x-3} + \sqrt{|y|-z},$$

gde su x, y i z realni brojevi.

```
program Izraz;
var
  x, y, z, rez : real;
begin
  writeln('Unesi x, y i z');
  readln(x, y, z);
  if (x-3 >= 0) and (abs(y)-z >= 0) then
    begin
      rez := sqrt(x-3) + sqrt(abs(y)-z);
      writeln('Rezultat: ', rez)
    end
  else
    writeln('Izraz nije definisan')
end.
```

Primer. Da li su sledeća dva programska fragmenta ekvivalentna, odnosno, da li se za iste ulazne vrednosti ponašaju na isti način?

```
readln(x);
readln(y);
rez := 0;
if y <> 0 then
  if x/y >= 0 then
    rez := sqrt(x/y);
writeln(rez);
```

```
readln(x);
readln(y);
rez := 0;
if (y <> 0) and (x/y >= 0) then
  rez := sqrt(x/y);
writeln(rez);
```

Odgovor: NE! Kada je `y <> 0` oni se zaista ponašaju na isti način. Međutim, kada je `y = 0`, drugi program puca nezavisno od vrednosti `x`, dok prvi program radi besprekorno. Drugi program puca za `y = 0` zato što će Paskal prevodilac pokušati da izračuna oba izraza u složenom uslovu pre nego što primeni veznik `and`, pa će pući kada pokuša da utvrdi da li je `x/y >= 0` (deljenje nulom). To što je prvi izraz netačan, tj. `y = 0`, neće ga sprečiti da nastavi sa računanjem drugog izraza.

Računar je ipak samo jedna mašina koja radi *tačno* ono što joj je rečeno da uradi. Ako korisnik želi da proveri uslov `(y <> 0) and (x/y >= 0)` računar će doslovno ispoštovati njegov nalog, pa makar i po cenu samoubistva!

☞ *Ako treba proveriti neki komplikovani uslov koji u sebi sadrži deljenje, korenovanje, ili neku drugu "opasnu" operaciju koja može da izazove pucanje programa, tri puta razmisliti da li se može upotrebiti složeni uslov, ili je možda ipak bolje koristiti ugnježdene "if"-ove!*

Zadaci.

2.25. Napisati Paskal program koji računa vrednost sledećeg izraza:

$$(a)\quad \sqrt{(x-5)(y-3)}+\sqrt{z+9} \qquad (b)\quad \sqrt{(x-5)(4-y)}+\sqrt{(9-x)yz}$$

$$(c)\quad \sqrt{|x|-\sqrt{2x-y}} \qquad (d)\quad \sqrt{\sqrt{\frac{|x|+z}{x^2+xy+y^2}}-\sqrt{\frac{|y|+z}{x^2+xy+y^2}}}$$

2.26. Napisati Paskal program koji od korisnika učitava tri realna broja, `a`, `b` i `c`, i utvrđuje da li postoji trougao kome su to merni brojevi dužina stranica. Ukoliko takav trougao postoji, izračunati njegovu površinu koristeći Heronov obrazac $P=\sqrt{s(s-a)(s-b)(s-c)}$ gde je s poluobim trougla.

2.27. Napisati Paskal program koji od korisnika učitava tri realna broja, `a`, `b` i `c`, i utvrđuje da li postoji trougao kome su to merni brojevi dužina stranica. Ukoliko takav trougao postoji, odrediti da li je taj trougao oštrougli, pravougli ili tupougli.

2.28. Napisati Paskal program koji od korisnika učitava koordinate (x,y) donjeg levog temena kvadrata čije stranice su paralelne koordinatnim osama, dužinu a stranice kvadrata kao i koordinate (px,py) tačke P, i utvrđuje da li se tačka P nalazi u unutrašnjosti kvadrata.

2.29. Napisati Paskal program koji od korisnika učitava koordinate (x,y) donjeg levog temena kvadrata čije stranice su paralelne koordinatnim osama, dužinu a stranice kvadrata kao i koordinate (px,py) tačke P za koju se zna da se nalazi izvan kvadrata, i određuje rastojanje tačke P od kvadrata. (Rastojanje tačke od kvadrata se računa tako što se nađe tačka Q na rubu kvadrata koja je najbliža tački P, pa se izračuna rastojanje tačaka P i Q.)

2.30. Napisati Paskal program koji od korisnika učitava četiri realna broja i potom računa i ispisuje najveći od njih.

2.10 Rezime

U ovoj glavi smo uveli blok kao posebnu konstrukciju programskog jezika Paskal koja omogućuje da se niz naredbi organizuje u jednu celinu:

⟨*Blok*⟩ ≡ `begin`
⟨*Naredba*$_1$⟩`;`
⟨*Naredba*$_2$⟩`;`
⋮
⟨*Naredba*$_k$⟩
`end`

Ključne reči `begin` i `end` služe kao zagrade koje ograđuju niz naredbi bloka. Blok može da sadrži i samo jednu naredbu, ili čak i da bude prazan (`begin end`). Blok može da se pojavi samo na strogo određenim mestima u programu. Na primer, glavni deo programa nije ništa drugo do blok, tako da struktura Paskal programa može da se opiše i na sledeći način:

⟨*Paskal program*⟩ ≡ `program` ⟨*ime*⟩`;`
[⟨*Definicije konstanti*⟩]
[⟨*Deklaracije promenljivih*⟩]
⟨*Blok*⟩ `.`

Osim bloka uveli smo još jednu važnu sintaksnu konstrukciju: “if” kontrolnu strukturu. “If” kontrolna struktura ima sledeći oblik:

⟨*If-naredba*⟩ ≡ `if` ⟨*uslov*⟩ `then`
⟨*Blok ili Naredba*⟩
[`else`
⟨*Blok ili Naredba*⟩]

⟨*Blok ili Naredba*⟩ ≡ ⟨*Blok*⟩ | ⟨*Naredba*⟩

Primetimo da su ovim jednostavnim pravilom obuhvaćeni svi specijalni slučajevi “if” kontrolne strukture koje smo pominjali u tekstu.

Glava 3

Celobrojni tip, “for” ciklus

Pred nama je glava koja otvara vrata ka pravom programiranju. U njoj prvo uvodimo celobrojni tip podataka, nakon čega sledi prva ciklička struktura sa kojom ćemo se sresti u kursu: “for” ciklus. Uvođenje “for” ciklusa nam omogućuje da se po prvi put uhvatimo u koštac sa ozbiljnijim programerskim problemima: određivanje delilaca datog broja, računanje suma i proizvoda, i računanje elemenata rekurzivno zadatih nizova. U ovoj glavi ćemo detaljno upoznati i standardne procedure za ispis poruka korisniku (`write` i `writeln`).

3.1 Celobrojni tip

Osim realnih brojeva, programski jezik Paskal poznaje i cele brojeve kao poseban tip. Celi brojevi postoje kao poseban tip zato što zauzimaju manje mesta u memoriji i zato sto se sa njima lakše i brže računa.

Postoje dva celobrojna tipa: jedan se zove `integer`, a drugi `longint`. Promenljiva tipa `integer` može da zapamti jedan ceo broj iz skupa

$$\{-32768, \ldots, -1, 0, 1, \ldots, 32767\},$$

dok promenljiva tipa `longint` može da zapamti jedan ceo broj iz skupa

$$\{-2147483648, \ldots, -1, 0, 1, \ldots, 2147483647\}.$$

Odgovarajuće promenljive se deklarišu na uobičajeni način (videti primer pored).

```
var
    x : integer;
    N : longint;
```

Programski jezik Paskal poznaje sledeće operacije nad celim brojevima:

Operacija	Objašnjenje	Primer
`+`	sabiranje	`x + 14`
`-`	oduzimanje	`x - 27`
`-`	promena znaka	`-x + 2`
`*`	množenje	`x * y`
`div`	celobrojno deljenje	`x div 7`
`mod`	ostatak pri deljenju	`x mod 7`
`sqr`	"na kvadrat"	`sqr(x)`
`abs`	apsolutna vrednost	`abs(x - 2)`
`odd`	da li je broj neparan?	`if odd(n) then...`

Operacija `div` daje količnik, a operacija `mod` ostatak pri celobrojnom deljenju. Funkcija `odd` se koristi uglavnom unutar "if" konstrukcije, kako je to i pokazano. U stvari, `odd` ne računa ništa već ispituje da li je broj neparan. Takve funkcije se zovu *testovi*. Imena operacija `div`, `mod` i `odd` potiču od sledećih engleskih reči: *div*ide (podeli), *mod*ulo (po modulu) i *odd* (čudan; neparan).

Primer. Napisati program koji od korisnika učitava pozitivan ceo broj n i potom računa i štampa zbir poslednje tri cifre broja n.

U programu koristimo da je

$$\boxed{\text{poslednja cifra broja } n} = n \bmod 10$$

$$\boxed{\text{broj } n \text{ bez poslednje cifre}} = n \text{ div } 10$$

Sledeća tabela ilustruje rad programa kada je $n = 15384$:

n	$c2$	$c1$	$c0$
15384	–	–	–
1538	–	–	4
153	–	8	4
15	3	8	4

```
program Posl3Cif;
var
  n, z : integer;
  c0, c1, c2 : integer;
begin
  writeln('n = ');
  readln(n);
  if n <= 0 then
    writeln('Greska')
  else
    begin
      { poslednja cifra }
      c0 := n mod 10;
      n  := n div 10;
      { pretposlednja cifra }
      c1 := n mod 10;
      n  := n div 10;
      { treca s kraja }
      c2 := n mod 10;
      z  := c0 + c1 + c2;
      writeln('zbir je ', z)
    end
end.
```

Primer: Prestupne godine. Godine 1582. papa Grgur XIII je odlučio da modifikuje kalendar jer je primećeno da tadašnji kalendar, koga je bio ustanovio još Julije Cezar, nije više precizan. Tada je dogovoreno da se zadrži osnovni princip julijanskog kalendara, a da se samo promeni način računanja prestupnih godina. Naime, po julijanskom kalendaru svaka četvrta godina je bila prestupna. Ispostavilo se da je to previše, jer je julijanski kalendar za nekih 1500 godina koliko je bio na snazi požurio oko 14 dana. Tako je dekretom kalendar vraćen unatrag i ustanovljeno je novo pravilo za računanje prestupnih godina:

- ako je godina deljiva sa 400 prestupna je;
- ako nije deljiva sa 400, ali je deljiva sa 100, onda nije prestupna;
- ako nije deljiva sa 100, ali je deljiva sa 4, onda je prestupna;
- a ako nije deljiva sa 4, onda nije prestupna.

Na primer: 2000. i 1968. godina su prestupne, dok 1900. i 1969. nisu.

Napisati program koji od korisnika učitava ceo broj iz skupa $\{1582,\ldots,9999\}$ i utvrđuje da li je odgovarajuća godina prestupna prema pravilu za računanje prestupnih godina u gregorijanskom kalendaru.

Rešnje. Da bismo napisali program potrebno je proveriti da li je neki ceo broj deljiv nekim drugim celim brojem. Srećom, to se lako može proveriti:

$$\boxed{x \text{ je deljivo sa } y \text{ ako i samo ako je } x \bmod y = 0.}$$

Program izgleda ovako:

```
program PrestupnaGod;
var
   g : integer;
begin
   writeln('Unesi godinu');
   readln(g);
   if g < 1582 then
      writeln('Greska')
   else if g mod 400 = 0 then
      writeln('Prestupna')
   else if g mod 100 = 0 then
      writeln('Nije prestupna')
   else if g mod 4 = 0 then
      writeln('Prestupna')
   else
      writeln('Nije prestupna')
end.
```

Primer: Razmena novca. Na raspolaganju imamo kovanice od 1, 5 i 10 dinara. Program od korisnika učitava neki pozitivan ceo broj koji predstavlja količinu novca, i “isplaćuje” mu taj iznos koristeći najmanji mogući broj kovanica.

Kviz.

1. Kako program radi za sledeće iznose: 43, 96, 158, 189?
2. Koliko najviše kovanica od 1 din može biti “isplaćeno”? Navesti primer.
3. Koliko najviše kovanica od 5 din može biti “isplaćeno”? Navesti primer.

```
program RazmenaNovca;
var
  iznos, k : integer;
begin
  writeln('Iznos?');
  readln(iznos);
  if iznos <= 0 then
    writeln('Greska')
  else
    begin
    { prvo apoeni od 10 din }
      k := iznos div 10;
      iznos := iznos mod 10;
      if k > 0 then
        writeln(k, ' po 10 din');
    { potom apoeni od 5 din }
      k := iznos div 5;
      iznos := iznos mod 5;
      if k > 0 then
        writeln(k, ' po 5 din');
    { apoeni od 1 din }
      if iznos > 0 then
        writeln(iznos, ' po 1 din')
    end
end.
```

3.2 Algebarski izrazi i konverzija tipova

Videli smo da Paskal poznaje različite vrste brojeva: realne brojeve (tip `real`) i cele brojeve (tipovi `integer` i `longint`). Njihov odnos je isti kao odnos celih i realnih brojeva u matematici:

$$\texttt{integer} \subset \texttt{longint} \subset \texttt{real}$$

i zato unutar izraza možemo mešati cele i realne brojeve.

Prilikom rada sa celim brojevima zagrade se koriste na isti način kao pri radu sa realnim brojevima. Operatori `*`, `div` i `mod` imaju isti prioritet, i on je veći od prioriteta koga imaju `+` i `-`, a isti je kao prioritet operacija `*` (množenje realnih brojeva) i `/` (deljenje realnih brojeva).

Ukoliko se u nekom algebarskom izrazu javljaju i realni i celi brojevi, svi brojevi se automatski konvertuju u realne brojeve, i rezultat je realan broj. Primeri:

Izraz	Tip rezultata
`10 * 2 + 9`	`integer`
`10 * 2 + 9.12`	`real`
`12 * 3.71`	`real`
`12 / 4`	`real` (primenjeno je deljenje realnih brojeva)
`12 div 4`	`integer`
`360 / 10 + 5`	`real`
`11 + 0`	`integer`
`11 + 0.0`	`real`
`sqrt(9)`	`real` (`sqrt` uvek vraća realan broj)
`abs(-9)`	`integer`
`abs(-9.0)`	`real`

Promenljivoj realnog tipa možemo dodeliti i realne i celobrojne vrednosti. Pri tome, celobrojne vrednosti će biti automatski konvertovane u ekvivalentan realni zapis. Međutim, celobrojnoj promenljivoj *ne možemo* dodeliti realan broj. Pored se nalazi jedan *nekorektan primer!*

Prilikom prevođenja, Paskal prevodilac će na ovom mestu prijaviti grešku (neusklađeni tipovi).

```
program POGRESNO;
var
   k : integer;
begin
   k := 3.75
end.
```

Razlozi za prijavljivanje greške su jasni: prevodilac ne zna šta da radi sa decimalama. Da li da ih otseče, ili da zaokruži broj? S obzirom da ne sme da dopusti da dođe do gubitka informacija, od programera se očekuje da navede koji oblik konverzije želi.

Zato su u programski jezik Paskal uvedene funkcije `trunc` i `round` (od engleskih reči *truncate* – otseci, i *round* – zaokruži). One uzmu jedan realan broj, a vrate ceo broj koji se od polaznog broja dobija otsecanjem decimala, odnosno zaokruživanjem na najbliži ceo broj. Primeri se mogu naći u tabeli pored (rezultat je uvek tipa `integer`).

Izraz	Rezultat
`trunc(3.14)`	3
`trunc(3.99)`	3
`trunc(-3.01)`	-3
`round(3.14)`	3
`round(3.99)`	4
`round(-3.01)`	-3

Primer: Potpun kvadrat. Ceo broj n je *potpun kvadrat* ako postoji ceo broj k takav da je

$$n = k^2.$$

Napisati Paskal program koji od korisnika učitava ceo broj n i utvrđuje da li je n potpun kvadrat.

```
program PotpunKvadrat;
var
  n, k : integer;
begin
  writeln('Unesi n');
  readln(n);
  k := round(sqrt(n));
  if sqr(k) = n then
    writeln('Jeste')
  else
    writeln('Nije')
end.
```

Kviz.

1. Izračunati vrednosti sledećih izraza:

(a) `(19 div 6) * 4 div 5 + 1`
(b) `(25 div 3) mod (25 mod 3)`
(c) `96 div 7 * 7`
(d) `426 div 10 div 6 mod 3`

2. Izračunati vrednost sledećih izraza i odrediti tip rezultata:

(a) `trunc(360 / 7) mod 11`
(b) `round(412 / 9) / 5`
(c) `round(19 / 4) / trunc(19 / 4)`
(d) `round(19 / 4) div trunc(9 / 4)`

3. (Da/Ne pitalice) Neka je `k` celobrojna promenljiva, a `r` realna promenljiva. Za koju od sledećih naredbi će prevodilac prijaviti grešku?

Naredba	Prevodilac se buni?	
`k := 14 * 3 - 6`	☐ Da	☐ Ne
`r := 14 * 3 - 6`	☐ Da	☐ Ne
`k := 15 / 3 + 15 div 6`	☐ Da	☐ Ne
`r := 15 / 3 + 15 div 6`	☐ Da	☐ Ne
`k := trunc(15 / 3) + 15 div 6`	☐ Da	☐ Ne
`r := round(15 / 4)`	☐ Da	☐ Ne

Zadaci.

3.1. Jedan molekul sumporne kiseline (H_2SO_4) se sastoji od dva atoma vodonika, jednog atoma sumpora i četiri atoma kiseonika. Napisati Paskal

program koji od korisnika učitava cele brojeve `NH`, `NS` i `NO`, koji predstavljaju broj atoma vodonika, sumpora i kiseonika, tim redom, koji nam stoje na raspolaganju, i potom računa i štampa maksimalan broj molekula sumporne kiseline koji se mogu formirati od datih atoma.

3.2. Molekul etanola ima formulu C_2H_5OH. Napisati Paskal program koji od korisnika učitava cele brojeve `NC`, `NH` i `NO`, koji predstavljaju broj atoma ugljenika, vodonika i kiseonika, tim redom, koji nam stoje na raspolaganju, i potom računa i štampa maksimalan broj molekula etanola koji se mogu formirati od datih atoma.

3.3. *Sudbinski broj* osobe je jednocifreni broj koji se računa na osnovu godine njenog rođenja na sledeći način: saberu se sve cifre u godini rođenja; ako se dobije broj koji ima više od jedne cifre, ponovo se saberu cifre dobijenog broja, i tako dalje, dok se ne dobije jednocifren broj. Na primer,

$$1990 \xrightarrow{1+9+9+0} 19 \xrightarrow{1+9} 10 \xrightarrow{1+0} 1,$$
$$1984 \xrightarrow{1+9+8+4} 22 \xrightarrow{2+2} 4,$$
$$2000 \xrightarrow{2+0+0+0} 2.$$

Napisati Paskal program koji od korisnika učitava ceo broj g, $1000 \leqslant g \leqslant 9999$, koji predstavlja godinu rođenja neke osobe i potom računa i ispisuje sudbinski broj te osobe.

3.4. (*Hronometar*) Napisati program koji od korisnika učitava tri cela broja, `h`, `m`, `s`, i proverava da li oni predstavljaju korektno zapisano vreme (sati, minuti, sekunde), odnosno, da li je $\texttt{h} \geqslant 0$ i $0 \leqslant \texttt{m}, \texttt{s} \leqslant 59$. (Pažnja: ovo nije časovnik, već hornometar, dakle uređaj koji meri koliko je vremena proteklo!) Ukoliko se radi o korektno zapisanom vremenu, učitati pozitivan ceo broj `q` koji predstavlja neki broj sekundi, i potom ispisati novo vreme u obliku `h:m:s` koje se dobija tako što se na učitano vreme dodaju učitane sekunde.

Na primer, ako je `h = 31`, `m = 58`, `s = 30` i `q = 1251`, računar treba da ispiše `32:19:21`.

3.5. Ugao meren stepenima može se predstaviti pomoću tri celobrojne promenljive `d`, `m`, `s`, gde `d` sadrži broj stepeni, `m` broj minuta, a `s` broj sekundi. Ovakve tri promenljive predstavljaju korektan zapis mere ugla ako je $0 \leqslant d \leqslant 359$, $0 \leqslant m \leqslant 59$ i $0 \leqslant s \leqslant 59$.

(a) Napisati Paskal program koji od korisnika učitava dva ugla α i β merena stepenima i određuje $\alpha + \beta$. Svi podaci su uneti u korektnom formatu i to ne treba proveravati.

(b) Napisati Paskal program koji od korisnika učitava dva ugla α i β merena stepenima i određuje $\alpha - \beta$. Svi podaci su uneti u korektnom formatu, zna se da je $\alpha > \beta$ i to ne treba proveravati.

3.6. Napisati program koji od korisnika učitava tri cela broja, `d`, `m`, `g`, i proverava da li oni predstavljaju korektan datum, pri čemu je `d` redni broj dana, `m` redni broj meseca, a `g` redni broj godine. (Uzeti da je $1582 \leqslant$ `g` $\leqslant 9999$.)

Na primer, za `d = 26, m = 12, g = 1735` računar treba da prijavi da se radi o korektnom datumu, dok za `d = 31, m = 4, g = 2001` i `d = 30, m = 2, g = 1848` treba da prijavi da se radi o nekorektnom datumu.

3.7. Napisati program koji od korisnika učitava tri cela broja, `d`, `m`, `g`, za koje se zna da predstavljaju *korektan* datum, i utvrditi redni broj tog dana u godini.

Na primer, za `d = 1, m = 1, g = 1875` program treba da ispiše 1, dok za `d = 7, m = 10, g = 2000` program treba da ispiše 281.

3.8. Milutin Milanković je 1923. godine predložio pravilo za određivanje prestupnih godina koje bi garantovalo da kalendar ostane precizan narednih 43000 godina. Milankovićevo pravilo glasi ovako:

- ako godina *nije sekularna* (tj. nije deljiva sa 100), ona je prestupna ako je deljiva sa 4, kao i kod julijanskog i gregorijanskog kalendara; na primer, 2004. godina bi bila prestupna, a 2005. ne bi bila prestupna;
- *sekularna godina* je prestupna ako *broj vekova* u toj godini pri deljenju sa 9 daje ostatak 2 ili 6; na primer

Godina	Broj vekova	Ostatak pri deljenju sa 9	Prestupna?
2000	20	2	da
2100	21	3	ne
2400	24	6	da
2700	27	0	ne

Napisati Paskal program koji od korisnika učitava ceo broj g, $g \geqslant 1923$, i proverava da li je ta godina prestupna prema Milankovićevom pravilu.

3.9. Napisati Paskal program koji računa vrednost

(a) funkcije koja je data na Sl. 3.1 (a);

(b) funkcije koja je data na Sl. 3.1 (b);

(c) funkcije koja je data na Sl. 3.1 (c).

Sve tri funkcije su definisane samo za $x \geqslant 0$. Program zato treba da učita neki realan broj $x \geqslant 0$, izračuna vrednost funkcije za to x i potom ispiše dobijenu vrednost. U slučaju da je $x < 0$ program treba da ispiše poruku o greški.

3.10. Napisati Paskal program koji od korisnika učitava neki realan broj i potom određuje i štampa prve dve decimale tog broja.

3.3 "For" ciklus

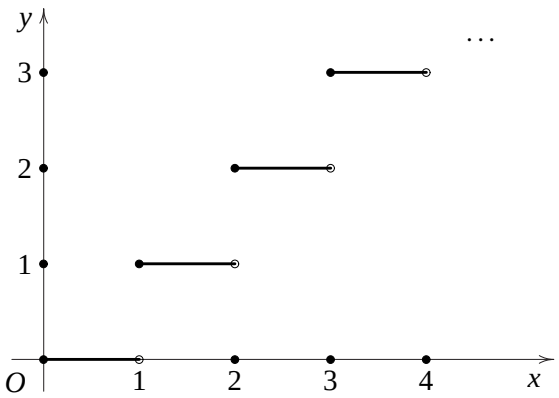
Često se tokom programiranja javlja potreba da se jedan isti deo programa izvrši više puta (20 puta, ili n puta, gde je n vrednost neke promenljive). Kontrolne strukture koje omogućuju da se program zavrti zovu se *ciklusi* ili *petlje*. Programski jezik Paskal poznaje nekoliko vrsta ciklusa i sada ćemo se upoznati sa jednom od njih.

"For" ciklus je konstrukcija programskog jezika Paskal koja nam omogućuje da jedan deo programa izvršimo unapred zadat broj puta. Postoje dve varijante "for" ciklusa: rastući "for" ciklus, i opadajući "for" ciklus.

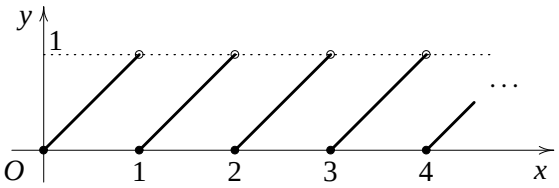
Rastući "for" ciklus sistematski uvećava vrednost neke celobrojne promenljive u zadatom intervalu i za svaku moguću vrednost izvrši dati spisak instrukcija. Opšti oblik rastućeg "for" ciklusa izgleda ovako:

```
for γ := α to β do
    P
```

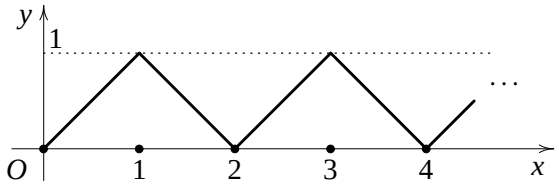
gde je γ neka celobrojna promenljiva, α i β su neki celobrojni izrazi, a P tačno jedna naredba ili blok. Celobrojnu promenljivu γ zovemo *kontrolna promenljiva*, izrazi α i β su *granice ciklusa*, dok P zovemo *telo ciklusa*.



(*a*)



(*b*)



(*c*)

Slika 3.1: Funkcije iz Zadatka 3.9

Primer. Program koji 10 puta ispisuje "Hello" može se napisati ovako:

```
program RastuciFor;
var
  i : integer;
begin
  for i := 1 to 10 do
    writeln(i, ' Hello')
end.
```

i ekvivalentan je programu pored→

```
program RaspisanRastuciFor;
var
  i : integer;
begin
  i := 1; writeln(i, ' Hello');
  i := 2; writeln(i, ' Hello');
  i := 3; writeln(i, ' Hello');
  i := 4; writeln(i, ' Hello');
  i := 5; writeln(i, ' Hello');
  i := 6; writeln(i, ' Hello');
  i := 7; writeln(i, ' Hello');
  i := 8; writeln(i, ' Hello');
  i := 9; writeln(i, ' Hello');
  i := 10; writeln(i, ' Hello')
end.
```

Prethodni program na monitor 10 puta ispisuje 'Hello', kako je to pokazano pored.

MONITOR:

```
1 Hello
2 Hello
3 Hello
4 Hello
5 Hello
6 Hello
7 Hello
8 Hello
9 Hello
10 Hello
```

☞ *Ako u rastućem "for" ciklusu*

for γ := α to β do
  P

imamo $\alpha > \beta$, onda se ciklus neće nijednom izvršiti!

Primer. Napisati Paskal program koji od korisnika učitava prirodan broj *n* i potom ispisuje prvih *n* kvadrata.

```
program Kvadrati;
var
  k, n : integer;
begin
  readln(n);
  for k := 1 to n do
    writeln(sqr(k))
end.
```

Opadajući "for" ciklus, analogno rastućoj varijanti, sistematski umanjuje vrednost kontrolne promenljive u zadatom intervalu i za svaku moguću vrednost izvršava dati spisak instrukcija. Opšti oblik opadajućeg "for" ciklusa izgleda ovako:

```
for γ := α downto β do
    P
```

gde je γ neka celobrojna promenljiva koja se zove *kontrolna promenljiva*, α i β su neki celobrojni izrazi – *granice ciklusa*, a P je *telo ciklusa*, što može biti tačno jedna naredba ili blok.

Primer. Program koji 10 puta ispisuje "Hello" može se napisati i ovako:

```
program OpadajuciFor;
var
  i : integer;
begin
  for i := 10 downto 1 do
    writeln(i, ' Hello')
end.
```

MONITOR:

```
10 Hello
9 Hello
8 Hello
7 Hello
6 Hello
5 Hello
4 Hello
3 Hello
2 Hello
1 Hello
```

Sada odbrojavanje ide od 10 do 1, a program je ekvivalentan programu pored →

```
program RaspisanOpadajuciFor;
var
  i : integer;
begin
  i := 10; writeln(i, ' Hello');
  i := 9; writeln(i, ' Hello');
  i := 8; writeln(i, ' Hello');
  i := 7; writeln(i, ' Hello');
  i := 6; writeln(i, ' Hello');
  i := 5; writeln(i, ' Hello');
  i := 4; writeln(i, ' Hello');
  i := 3; writeln(i, ' Hello');
  i := 2; writeln(i, ' Hello');
  i := 1; writeln(i, ' Hello')
end.
```

☞ *Ako u opadajućem "for" ciklusu*

```
for γ := α downto β do
    P
```

imamo $\alpha < \beta$, onda se ciklus neće nijednom izvršiti!

Kviz.

1. Šta radi svaki od sledećih programa?

```
program X;
var
  i, n : integer;
begin
  readln(n);
  for i := -n to n do
    writeln(i)
end.
```

```
program Y;
var
  i, n : integer;
begin
  readln(n);
  for i := n downto 1 do
    if odd(i) then
      writeln(i)
end.
```

```
program Z;
var
  x, n : integer;
begin
  readln(n);
  for x := -n to n do
    if sqr(x) + 3*x - 10 = 0 then
      writeln(x)
end.
```

2. Koliko puta će svaki od narednih ciklusa ispisati reč "Hello" na monitoru:

(a)
```
for i := -1 to 1 do
  writeln('Hello')
```

(b)
```
for i := 10 to 20 do
  writeln('Hello')
```

(c)
```
a := 1; b := 7;
for i := a+1 to b-a do
  writeln('Hello')
```

(d)
```
for i := 1 downto -3 do
  writeln('Hello')
```

(e)
```
for i := 10 downto 20 do
  writeln('Hello')
```

(f)
```
a := 9; b := -3;
for i := a+b downto -a-b do
  writeln('Hello')
```

Zadaci.

3.11. Napisati Paskal program koji od korisnika učitava cele brojeve g_1, g_2 i ako je $1582 \leqslant g_1 \leqslant g_2 \leqslant 9999$ ispisuje sve prestupne godine u intervalu $[g_1, g_2]$. U ostalim slučajevima ispisuje poruku o greški.

3.12. Napisati Pascal program koji od korisnika učitava dva cela broja g_1, g_2 tako da je $2001 \leqslant g_1 \leqslant g_2 \leqslant 9999$ i potom štampa sve godine iz intervala $[g_1, g_2]$ kojima je 1. januar ponedeljak. Zna se da je 2001. godine 1. januar bio ponedeljak.

3.13. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj n i ispisuje sve njegove pozitivne delioce.

3.14. Napisati Paskal program koji od korisnika učitava cele brojeve $d \geqslant 0$ i $n \geqslant 10$, i potom određuje i štampa sve brojeve iz skupa $\{10, 11, \ldots, n\}$ čiji zbir poslednje dve cifre je jednak sa d.

3.15. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 1$, potom n celih brojeva i određuje najveći od njih. Na primer, za $n = 5$ i za brojeve 2, 7, 2, 9, 7 program ispisuje 9.

3.16. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 1$, potom n celih brojeva i određuje najmanji od njih. Na primer, za $n = 5$ i za brojeve 2, 7, 2, 9, 7 program ispisuje 2.

3.17. Napisati Paskal program koji učitava pozitivne cele brojeve m, n i k, i potom ispisuje količnik brojeva m i n na k decimala.

3.18. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj p i potom određuje pozitivne cele brojeve a i b tako da to budu dužine stranica pravougaonika najmanjeg obima čija površina je p.

3.19. Napisati Paskal program koji štampa sve trocifrene brojeve $\overline{abc}$ za koje je $\overline{abc} = (\overline{ab})^2 - c^2$. Na primer, $147 = 14^2 - 7^2$.

3.20. Napisati Paskal program koji štampa sve četvorocifrene brojeve $\overline{abcd}$ za koje je $\overline{abcd} = (a+b+c+d)^2$.

3.21. Napisati Paskal program koji traži sve moguće načine da se dešifruje jednakost

$$(***)^2 = *00**,$$

gde zvezdice označavaju proizvoljne cifre. Pri tome, prva cifra u broju nije nula.

†**3.22.** Napisati Pascal program koji ispisuje sve moguće načine da se dešifruje sabiranje

$$\begin{array}{cccccc} & & \mathrm{B} & \mathrm{O} & \mathrm{R} \\ + & & \mathrm{B} & \mathrm{O} & \mathrm{R} \\ \hline & \check{\mathrm{S}} & \mathrm{U} & \mathrm{M} & \mathrm{A} \end{array}$$

gde različitim slovima odgovaraju različite cifre, i pri tome $\check{\mathrm{S}} \neq 0$ i $\mathrm{B} \neq 0$.

3.4 Brojač

Brojač je celobrojna promenljiva koju u programu koristimo da prebrojimo nešto, na primer, delioce nekog broja, ili parne brojeve među brojevima koje je uneo korisnik. Brojač se deklariše kao i svaka druga celobrojna promenljiva. Pre svake upotrebe brojač mora da se *inicijalizuje* – postavi na početnu vrednost, najčešće 0. Vrednost brojača se potom može uvećati prema potrebi. Ako želimo da brojač k uvećamo za 1, recimo, to možemo učiniti naredbom `k := k + 1`.

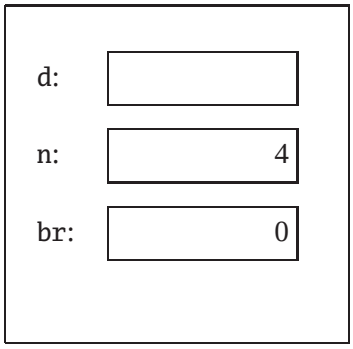
Primer. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj *n* i potom određuje i ispisuje broj njegovih delilaca.

Rešenje je dato pored. Promenljiva `br` broji delioce broja `n` i to je brojač.

```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Pogledajmo sada detaljnije kako program radi. Prvo korisnik unese vrednost promenljive `n`, neka je to 4 za potrebe ovog primera, a onda se vrednost promenljive `br` postavi na 0.

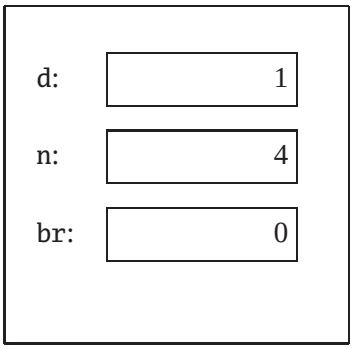
MEMORIJA:




```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Sada krećemo sa izvršavanjem "for" ciklusa. Vrednost promenljive d se postavi na 1 zato što je to početna vrednost ciklusa:

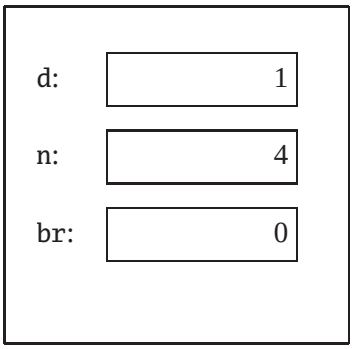
MEMORIJA:




```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Potom proverimo da li je d delilac broja n:

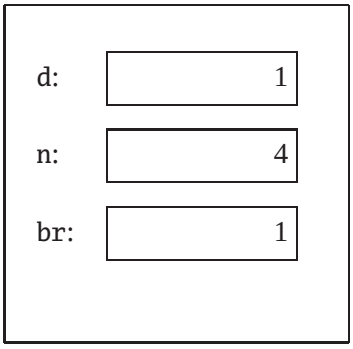
MEMORIJA:




```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

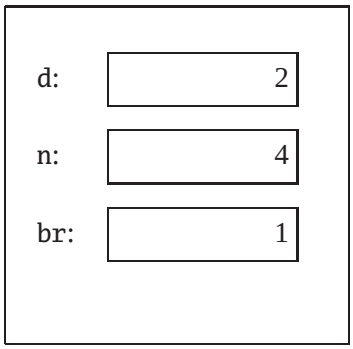
Pošto to jeste slučaj, vrednost promenljive br se uveća za 1:

MEMORIJA:



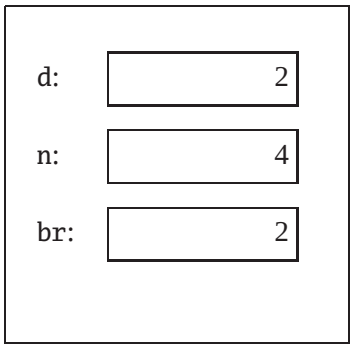

```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Sada se vraćamo na početak “for” ciklusa i tom prilikom promenljiva d dobija sledeću vrednost u nizu:



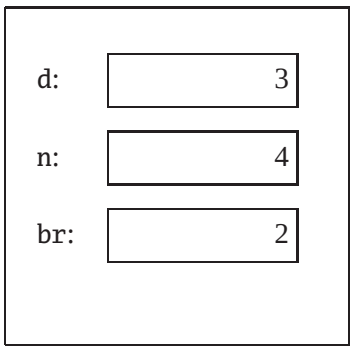

```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Pošto je d delilac broja n, promenljiva br se opet uveća za 1:




```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

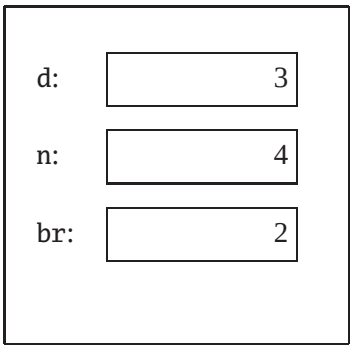
Zato što nismo dostigli gornju granicu “for” ciklusa, ponovo se vraćamo na njegov početak i promenljiva d dobija narednu vrednost:




```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Ovaj put promenljiva d nije delilac promenljive n, pa se naredba dodele iza then ne izvršava:

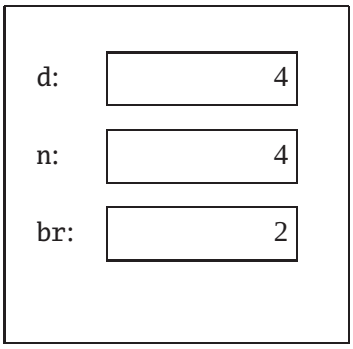
MEMORIJA:




```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Povratak na početak ciklusa i sledeća vrednost za d:

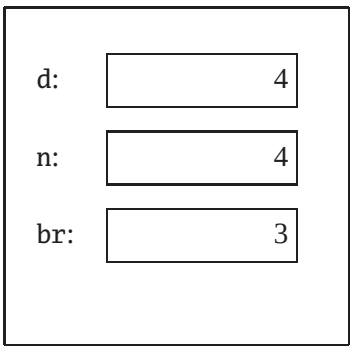
MEMORIJA:




```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Promenljiva d ponovo deli bez ostatka promenljivu n i zato se vrednost promenljive br uveća za 1:

MEMORIJA:




```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Pošto je sada d jednako sa n, gornja granica ciklusa je dostignuta i ciklus se završava. Na kraju ispišemo vrednost promenljive br.

Napomenimo još da program BrojDel radi korektno za pozitivne cele brojeve, dok za ostale cele brojeve ispisuje 0 (zašto?).

```
program BrojDel;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  writeln(br)
end.
```

Primer. Od programa BrDel se malom modifikacijom lako dobija program koji ispituje da li je dati broj prost. Podsetimo se: pozitivan broj je prost ako ima tačno dva delioca.

Program Prost koji je naveden pored broji delioce broja n i proverava da li je dobijeni broj jednak 2. *Ovo je najsporiji mogući način* da se proveri da li je broj prost. Znatno bolji algoritam ćemo videti u Glavi 5.

```
program Prost;
var
  d, n, br : integer
begin
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  if br = 2 then
    writeln('Prost')
  else
    writeln('Nije prost')
end.
```

Primer. Napisati program koji od korisnika učitava ceo broj *n*, potom *n* celih brojeva i utvrđuje koliko je među njima parnih.

```
program Parni;
var
  k, n, a, br : integer
begin
  readln(n);
  br := 0;
  for k := 1 to n do
    begin
      readln(a);
      if not odd(a) then
        br := br + 1
    end;
  writeln(br)
end.
```

Zadaci.

3.23. Napisati Paskal program koji od korisnika učitava ceo broj n i potom utvrđuje koliko n ima delilaca koji su oblika $4k+1$.

3.24. Napisati Paskal program koji od korisnika učitava cele brojeve g_1, g_2 i ako je $1582 \leqslant g_1 \leqslant g_2 \leqslant 9999$ određuje broj prestupnih godina u intervalu $[g_1, g_2]$. U ostalim slučajevima ispisuje poruku o greški.

3.25. Napisati Paskal program koji od korisnika učitava ceo broj n, potom n celih brojeva i utvrđuje koliko je među njima neparnih.

3.26. Napisati Paskal program koji od korisnika učitava ceo broj n, potom n celih brojeva i utvrđuje koliko je među njima deljivih sa 3, koliko deljivih sa 5, a koliko nije deljivo ni sa 3 ni sa 5.

3.27. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 1$, potom n celih brojeva i određuje najveći od njih, kao i koliko puta se on pojavio u nizu. Na primer, za $n = 6$ i za brojeve 2, 9, 2, 9, 7, 9 program ispisuje 9 i 3.

3.28. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 1$, potom n celih brojeva i određuje najveći od njih, kao i

(a) mesto na kome se on prvi put pojavio u nizu (na primer, za $n = 6$ i za brojeve 2, 9, 2, 9, 7, 9 program ispisuje 9 i 2);

(b) mesto na kome se on poslednji put pojavio u nizu (na primer, za $n = 6$ i za brojeve 2, 9, 2, 9, 7, 9 program ispisuje 9 i 6).

3.29. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 1$, potom n celih brojeva i određuje najmanji i najveći od njih, kao i koliko puta su se najmanji i najveći broj pojavili u nizu. Na primer, za $n = 6$ i za brojeve 2, 9, 2, 9, 7, 9 program ispisuje 2, 2, 9, 3 zato što je 2 najmanji broj u nizu i pojavljuje se 2 puta, dok je 9 najveći broj u nizu i pojavljuje se 3 puta.

3.30. Dresirani žabac se nalazi na livadi. Kada dobije komandu "1" on skoči 1m na sever; kada dobije komandu "2" skoči 1m na istok; kada dobije komandu "3" skoči 1m na jug; a kada dobije komandu "4" skoči 1m na zapad.

Napisati Paskal program koji od korisnika učitava ceo broj n, potom n komandi (dakle, brojeve iz skupa $\{1,2,3,4\}$) i utvrđuje da li će se nakon tog niza komandi žabac naći u polaznoj tački.

3.31. Na na beskonačnoj, idealno ravnoj platformi nalazi se robot koji razume sledeće tri komande:

- 1 — okreni se levo za 90° u mestu,
- 2 — okreni se desno za 90° u mestu,
- 3 — idi napred $1m$.

Napisati Paskal program koji od korisnika učitava ceo broj n, potom n komandi i utvrđuje da li se robot posle izvršenih n komandi našao na mestu na kome je stajao pre početka izvršavanja komandi.

3.5 Sume i proizvodi

Veoma često se u programiranju javlja problem da se izračuna suma

$$a_1 + a_2 + \ldots + a_n$$

ili proizvod

$$a_1 \cdot a_2 \cdot \ldots \cdot a_n$$

gde su a_i neki brojevi. Na primerima ćemo pokazati kako se to radi. Prvo ćemo videti najjednostavniju ideju, a potom i jedan mali trik kojim se sume i proizvodi ponekad mogu inteligentnije izračunati.

Primer. Pored je dat program koji računa sumu

$$1^2 + 2^2 + 3^2 + \ldots + n^2.$$

Tokom rada programa na pomoćnu promenljivu `sum` polako dodajemo sabirke jedan po jedan. Na kraju, promenljiva `sum` sadrži vrednost cele sume.

```
program PrvaSuma;
var
  i, n, sum : integer;
begin
  readln(n);
  sum := 0;
  for i := 1 to n do
    sum := sum + sqr(i);
  writeln('suma = ', sum)
end.
```

Program radi korektno samo za $n > 0$. (Šta program ispisuje za $n \leqslant 0$?)
Evo kako se menja stanje promenljive `sum` u raznim fazama rada programa:

Pre ulaska u petlju:	$\texttt{sum} = 0$
1. prolaz koz petlju:	$\texttt{sum} = 1^2$
2. prolaz koz petlju:	$\texttt{sum} = 1^2 + 2^2$
3. prolaz koz petlju:	$\texttt{sum} = 1^2 + 2^2 + 3^2$
$\vdots$	
n-ti prolaz koz petlju:	$\texttt{sum} = 1^2 + 2^2 + 3^2 + \ldots + n^2$

Primer. Napisati Paskal program koji za dati prirodan broj n računa

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n.$$

(Napomena: i ovaj program radi korektno samo za $n > 0$.)

```
program PrviProizvod;
var
  i, n : integer;
  fakt : longint;
begin
  readln(n);
  fakt := 1;
  for i := 1 to n do
    fakt := fakt * i;
  writeln('n! = ', fakt)
end.
```

Primer. Napisati Paskal program koji za dati realan broj x i dati prirodan broj n računa x^n.

(Napomena: i ovaj program radi korektno samo za $n > 0$.)

```
program Stepen;
var
  i, n : integer;
  x, p : real
begin
  readln(x, n);
  p := 1;
  for i := 1 to n do
    p := p * x;
  writeln('x^n = ', p)
end.
```

Zadaci.

3.32. Napisati Paskal program koji računa vrednost sledećeg izraza:

(a) $3^3 + 4^3 + 5^3 + \ldots + n^3$

(b) $1^2 + 3^2 + 5^2 + \ldots + (2n-1)^2$

(c) $\left(\frac{1}{x-1} - \frac{1}{x}\right) \cdot \left(\frac{1}{x-2} - \frac{1}{x-1}\right) \cdot \ldots \cdot \left(\frac{1}{x-n} - \frac{1}{x-(n-1)}\right)$

(x je realan broj i $x > n$)

3.33. Napisati Paskal program koji od korisnika učitava prirodan broj n, potom n realnih brojeva $a_1, \ldots, a_n$ i računa

(a) njihovu aritmetičku sredinu $A = \frac{a_1 + a_2 + \ldots + a_n}{n}$;

(b) njihovu kvadratnu sredinu $K = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$;

(c) njihovu harmonijsku sredinu $H = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \ldots + \frac{1}{a_n}}$.

3.34. Napisati Paskal program koji od korisnika učitava prirodne brojeve n i k i potom ispisuje sumu poslednjih k cifara broja n.

3.35. Za prirodan broj n sa $\sigma(n)$ označavamo sumu svih brojeva iz skupa $\{1,2,\ldots,n-1\}$ koji su delioci broja n. Napisati Paskal program koji od korisnika učitava ceo broj n i ako je $n \geqslant 3$ računa $\sigma(n)$.

3.36. Prirodan broj n je *savršen* ako je $n = \sigma(n)$ (videti Zadatak 3.35). Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 3$ i utvrđuje da li je on savršen.

3.37. Prirodni brojevi m i n su *prijateljski brojevi* ako je $n = \sigma(m)$ i $m = \sigma(n)$ (videti Zadatak 3.35). Napisati Paskal program koji od korisnika učitava cele brojeve $m,n \geqslant 3$ i utvrđuje da li su oni prijateljski brojevi.

3.38. Napisati Paskal program koji računa x^n za sve realne brojeve x i sve cele brojeve n. Imati u vidu da 0^0 nije definisano, da je $x^0 = 1$ za $x \neq 0$ i da za negativne n imamo $x^n = \dfrac{1}{x^{|n|}}$.

3.39. Za prirodan broj n, broj $n!!$ se definiše ovako: ako je n paran, onda je $n!!$ proizvod svih parnih brojeva od 2 do n; ako je n neparan, onda je $n!!$ proizvod svih neparnih brojeva od 3 do n. Na primer, $8!! = 2\cdot 4\cdot 6\cdot 8$, a $11!! = 3\cdot 5\cdot 7\cdot 9\cdot 11$.

Napisati Paskal program koji učitava ceo broj n i ako je $n > 0$ štampa $n!!$. Ako je $n \leqslant 0$ program ispisuje poruku o greški.

3.40. Verižni razlomak je razlomak koji ima oblik koji je naznačen pored, gde su a_k neki realni brojevi. Napisati Paskal program koji od korisnika učitava n, i onda računa verižni razlomak za

$$a_n + \cfrac{1}{a_{n-1} + \cfrac{1}{a_{n-2} + \cfrac{1}{\ddots \\ a_1 + \cfrac{1}{a_0}}}}$$

niz realnih brojeva $a_0, \ldots, a_n$ koji se tim redom učitavaju od korisnika. Pri tome korisnik vodi računa da svi razlomci budu definisani.

3.41. U kombinatorici se veoma često javlja sledeći broj:

$$\binom{n}{k} = \frac{n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-k+1)}{k!}$$

koji predstavlja broj načina da se iz skupa od n objekata odabere k-elementni podskup. Napisati Paskal program koji od korisnika učitava cele brojeve n i k, i ako je $n \geqslant k \geqslant 1$ računa $\binom{n}{k}$, dok u ostalim slučajevima prijavljuje grešku.

3.6 Sume i proizvodi malo inteligentnije

Da bismo napisali Paskal program koji računa vrednost izraza

$$1! + 2! + 3! + \ldots + n!$$

potrebno nam je jedno malo lukavstvo koje se sastoji u tome da sledeći sabirak računamo na osnovu prethodnog. U ovom slučaju je to moguće zato što je

$$\begin{aligned} k! &= k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 1 \\ &= k \cdot (k-1)!. \end{aligned}$$

Program `LukavaSuma` računa sumu faktorijela na lukav način.

```
program LukavaSuma;
var
   i, k, n : integer;
   suma : longint;
begin
   readln(n);
   suma := 1;
   k := 1;
   for i := 2 to n do
      begin
         k := k * i;
         suma := suma + k
      end;
   writeln('suma = ', suma)
end.
```

Evo kako lukavstvo radi:

Pre ulaska u petlju:	$\mathtt{k} = 1!$, $\mathtt{sum} = 1!$
1. prolaz koz petlju za $\mathtt{i} = 2$:	$\mathtt{k} = 2!$, $\mathtt{sum} = 1 + 2!$
2. prolaz koz petlju za $\mathtt{i} = 3$:	$\mathtt{k} = 3!$, $\mathtt{sum} = 1 + 2! + 3!$
3. prolaz koz petlju za $\mathtt{i} = 4$:	$\mathtt{k} = 4!$, $\mathtt{sum} = 1 + 2! + 3! + 4!$
$\vdots$	
poslednji prolaz koz petlju za $\mathtt{i} = n$:	$\mathtt{k} = n!$, $\mathtt{sum} = 1 + 2! + 3! + \ldots + n!$

☞ ***Opšta preporuka:*** *Kada treba izračunati sumu $a_1 + a_2 + \ldots + a_n$ pogledati prvo da li se a_k može jeftino dobiti od a_{k-1}. U tu svrhu proveriti da li je neki od ova dva izraza $a_k - a_{k-1}$ ili $\dfrac{a_k}{a_{k-1}}$ lep i jednostavan. Ako jeste, onda se može koristiti metod lukavog sumiranja. Ako nije, onda nam nema spasa ...*

Zadaci.

3.42. Napisati Paskal program koji računa proizvod:

$$\frac{1}{\sqrt{2}}\cdot\frac{1}{\sqrt{2+\sqrt{2}}}\cdot\frac{1}{\sqrt{2+\sqrt{2+\sqrt{2}}}}\cdot\ldots\cdot\frac{1}{\sqrt{2+\ldots\sqrt{2+\sqrt{2}}}}$$

gde se u poslednjem razlomku znak za koren javlja n puta.

3.43. Izračunati:

(a) $n^1-n^2+n^3-\ldots+(-1)^{k+1}n^k$, ($n$ i k su neki prirodni brojevi)

(b) $a_1+a_2+\ldots+a_n$, gde je $a_k=(k+3)^2-k$

(c) $\dfrac{1!}{n^1}+\dfrac{2!}{n^2}+\dfrac{3!}{n^3}+\ldots+\dfrac{m!}{n^m}$, gde su n i m neki prirodni brojevi

(d) $a_1-a_2+\ldots+(-1)^{n-1}a_n$, gde je $a_k=1^2+3^2+\ldots+(2k-1)^2$

(e) $a_1+a_2+\ldots+a_n$, gde je $a_k=\dfrac{1}{k}+\dfrac{1}{k+1}+\ldots+\dfrac{1}{2k}$

3.44. Napisati Paskal program koji od korisnika učitava nenegativan ceo broj $n\geqslant 0$, potom učitava realan broj x i realne brojeve $a_0,a_1,\ldots,a_n$ i nakon toga računa i štampa vrednost izraza

$$a_0+a_1x+a_2x^2+\ldots+a_nx^n.$$

3.7 Rekurzije

Za niz brojeva $a_1, a_2, a_3, \ldots, a_n, \ldots$ kažemo da je zadat *rekurzivno* ako je, osim prvih nekoliko članova, svaki član niza definisan preko nekih prethodnih. Tako, na primer, za niz brojeva

$$a_1=1,\quad a_n=3a_{n-1}-1$$

imamo: $a_1=1$, $a_2=3a_1-1=2$, $a_3=3a_2-1=5$, $a_4=3a_3-1=14$, $a_5=3a_4-1=41$, itd.

Program koji računa n-ti element niza

$$a_1 = 1, \quad a_n = 3a_{n-1} - 1$$

dat je pored. Ovu tehniku možemo da koristimo kadgod imamo niz kod koga je opšti član definisan samo preko njegovog prethodnika. Međutim, ako imamo niz kod koga je opšti član definisan preko više prethodnih članova, odgovarajući program se malo komplikuje.

```
program Niz;
var
   i, n : integer;
   a    : longint;
begin
   writeln('n = ');
   readln(n);
   if n <= 0 then
      writeln('Nije definisano')
   else if n = 1 then
      writeln('a_n = ', 1)
   else
      begin
         a := 1;
         for i := 2 to n do
            a := 3 * a - 1;
         writeln('a_n = ', a)
      end
end.
```

Razmotrimo sledeći primer. *Fibonačijevi (Fibonacci)* brojevi se definišu na sledeći način:

$$F_1 = 1, \quad F_2 = 1,$$
$$F_n = F_{n-1} + F_{n-2}.$$

Na primer,

$$F_3 = F_2 + F_1 = 2,$$
$$F_4 = F_3 + F_2 = 3,$$
$$F_5 = F_4 + F_3 = 5,$$
$$F_6 = F_5 + F_4 = 8,$$
$$F_7 = F_6 + F_5 = 13,$$

itd. Program koji računa n-ti Fibonačijev broj dat je pored.

```
program Fib;
var
   i, n       : integer;
   F1, F2, F3 : longint;
begin
   writeln('n = ');
   readln(n);
   if n <= 0 then
      writeln('Nije definisano')
   else if (n = 1) or (n = 2) then
      writeln('F_n = ', 1)
   else
      begin
         F2  := 1;
         F3  := 1;
         for i := 3 to n do
            begin
               F1 := F2;
               F2 := F3;
               F3 := F2 + F1
            end;
         writeln('F_n = ', F3)
      end
end.
```

Zadaci.

3.45. Napisati Paskal program koji računa n-ti član sledećeg niza:

(a) $a_1 = -1,\ a_2 = 5,\ a_n = a_{n-1}^2 - 4a_{n-2}$
(b) $b_1 = 1,\ b_2 = 2,\ b_3 = 3,\ b_n = 2b_{n-1}b_{n-2} - b_{n-3}^2$
(c) $c_1 = 1,\ c_2 = -2,\ c_3 = 3,\ c_4 = -2,\ c_n = c_{n-2} - c_{n-4}$
(d) $d_1 = 1,\ d_2 = -1,\ d_n = d_{n-1}^2 - (n-2)! \cdot d_{n-2}$
(Pažnja: faktorijel računati lukavo.)

3.46. Niz brojeva a_n je definisan ovako:

$$a_1 = 1, a_2 = 1,\ a_n = a_{n-2} + a_{n-3} + \ldots + a_1.$$

Napisati Paskal program koji od krisnika učitava prirodan broj n i računa n-ti element ovog niza.

3.47. *Paskalov trougao* je trougaona šema brojeva koja izgleda ovako:

```
1
1  1
1  2   1
1  3   3   1
1  4   6   4   1
1  5  10  10   5   1
```

Primetimo da n-ti red ovog trougla ima oblik $q_0\ q_1\ q_2 \ldots q_n$ gde je $q_{k+1} = (n-k)q_k$ div $(k+1)$. Napisati Paskal program koji ispisuje elemente n-tog reda Paskalovog trougla.

3.48. Niz brojeva a_n je definisan ovako:

$$a_1 = 1, a_2 = 1,\ a_n = \begin{cases} a_{n-1} + a_{n-2}, & n \text{ parno} \\ 3a_{n-1} - 2a_{n-2}, & n \text{ neparno}. \end{cases}$$

Napisati Paskal program koji od krisnika učitava prirodan broj n i računa n-ti element ovog niza.

3.49. Niz brojeva a_n je definisan ovako:

$$a_1 = 3, a_2 = -1,\ a_n = \begin{cases} na_{n-1} + a_{n-2}, & n \text{ parno} \\ a_{n-1} - na_{n-2}, & n \text{ neparno}. \end{cases}$$

Napisati Paskal program koji od krisnika učitava prirodan broj n i računa i štampa sumu prvih n elemenata ovog niza.

3.8 Ugnježdeni ciklusi

Telo "for" ciklusa može biti blok ili bilo koja komanda. Između ostalog, u telu jednog "for" ciklusa može se nalaziti neki drugi "for" ciklus, kako je to pokazano u primeru pored.

Za ovako raspoređene cikluse kažemo da su *ugnježđeni*.

```
for i := 1 to n do
  begin
    ...
    for j := -m to m do
      begin
        ...
      end;
    ...
  end
```

Primer. Napisati Paskal program koji od korisnika učitava ceo broj n i potom ispisuje tabelu brojeva koja je data pored za $n = 4$.

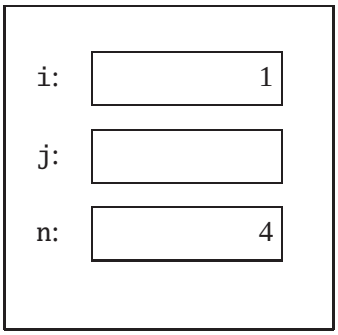
```
1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
3 1
3 2
3 3
3 4
4 1
4 2
4 3
4 4
```

Rešenje.

```
program Tabelica;
var
  i, j, n : integer;
begin
  readln(n);
  for i := 1 to n do
    for j := 1 to n do
      writeln(i, ' ', j)
end.
```

Pogledaćemo sada detaljno kako program radi za $n = 4$. U prvom prolazu kroz spoljašnji "for" ciklus promenljiva `i` dobije vrednost 1:

MEMORIJA:




```
program Tabelica;
var
  i, j, n : integer;
begin
  readln(n);
  for i := 1 to n do
    for j := 1 to n do
      writeln(i, ' ', j)
end.
```

nakon čega se izvrši unutrašnji “for” ciklus koji ispiše prva četiri reda tabele:

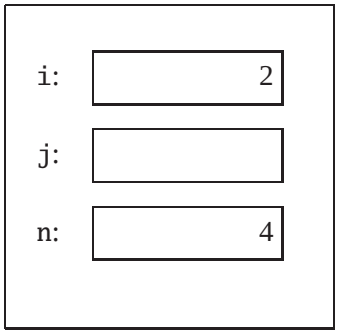
MONITOR:

```
1 1
1 2
1 3
1 4
```

```
program Tabelica;
var
  i, j, n : integer;
begin
  readln(n);
  for i := 1 to n do
    for j := 1 to n do
      writeln(i, ' ', j)
end.
```

U drugom prolazu kroz spoljašnji “for” ciklus promenljiva `i` dobije vrednost 2:

MEMORIJA:




```
program Tabelica;
var
  i, j, n : integer;
begin
  readln(n);
  for i := 1 to n do
    for j := 1 to n do
      writeln(i, ' ', j)
end.
```

nakon čega se izvrši unutrašnji “for” ciklus koji ispiše naredna četiri reda tabele:

MONITOR:

```
1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
```

```
program Tabelica;
var
  i, j, n : integer;
begin
  readln(n);
  for i := 1 to n do
    for j := 1 to n do
      writeln(i, ' ', j)
end.
```

U sledećem prolazu kroz spoljašnji "for" ciklus promenljiva `i` dobije vrednost 3 nakon čega se izvrši unutrašnji "for" ciklus koji ispiše još četiri reda tabele:

MONITOR:

```
2 1
2 2
2 3
2 4
3 1
3 2
3 3
3 4
```

```
program Tabelica;
var
  i, j, n : integer;
begin
  readln(n);
  for i := 1 to n do
    for j := 1 to n do
      writeln(i, ' ', j)
end.
```

dok se u poslednjem prolazu kroz spoljašnji "for" ciklus, za `i = 4`, ispišu poslednja četiri reda tabele:

MONITOR:

```
3 1
3 2
3 3
3 4
4 1
4 2
4 3
4 4
```

```
program Tabelica;
var
  i, j, n : integer;
begin
  readln(n);
  for i := 1 to n do
    for j := 1 to n do
      writeln(i, ' ', j)
end.
```

Primer. Napisati Paskal program koji od korisnika učitava pozitivne cele brojeve n i k i potom računa i štampa vrednost sledećeg izraza:

$$1^k + 2^k + 3^k + \ldots + n^k.$$

```
program SumaStepena;
var
  n, k, s, p, i, j : integer;
begin
  readln(n, k);
  s := 0;
  for i := 1 to n do
    begin
      p := 1;
      for j := 1 to k do
        p := p * i;
      s := s + p
    end;
  writeln(s)
end.
```

Da bismo shvatili kako ovaj program radi, primetimo prvo da uokvireni fragment računa i^k i odgovarajuću vrednost upisuje u promenljivu `p`.

Dalje je lako: za `i = 1` uokvireni fragment izračuna 1^k i to doda na `s`; potom za `i = 2` uokvireni fragment izračuna 2^k i to doda na `s`; i tako dalje. Na kraju, za `i = n`, uokvireni fragment izračuna n^k i to doda na `s`.

```
program SumaStepena;
var
  n, k, s, p, i, j : integer;
begin
  readln(n, k);
  s := 0;
  for i := 1 to n do
    begin
      p := 1;
      for j := 1 to k do
        p := p * i;
      s := s + p
    end;
  writeln(s)
end.
```

Zadaci.

3.50. Šta radi program pored? Šta se dešava ako je $n \leqslant 0$?

```
program X;
var
   i, j, n : integer;
begin
   readln(n);
   for i := n downto 1 do
      for j := 1 to i do
         writeln(i, ' ', j)
end.
```

3.51. Prirodan broj n je *savršen* ako je $n = \sigma(n)$ (videti Zadatak 3.36). Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 3$ i ispisuje sve savršene brojeve iz skupa $\{3,4,\ldots,n\}$.

3.52. Prirodni brojevi m i n su *prijateljski brojevi* ako je $n = \sigma(m)$ i $m = \sigma(n)$ (videti Zadatak 3.37). Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 3$ i ispisuje sve parove prijateljskih brojeva (a,b) takve da je $3 \leqslant a \leqslant b \leqslant n$.

3.53. Za uređenu trojku (k,l,m) prirodnih brojeva kažemo da je *Pitagorina trojka* ako je $k^2 + l^2 = m^2$. Napisati Paskal program koji od korisnika učitava ceo broj n i ako je $n > 0$ štampa sve Pitagorine trojke (k,l,m) za koje je $k,l,m \in \{1,2,\ldots,n\}$. Ako je $n \leqslant 0$ ispisati neku poruku o greški.

3.54. Papir je paralelnim linijama izdeljen na jedinične kvadratiće. Nacrtan je krug sa centrom u preseku jednog para normalnih linija i poluprečnikom r. Napisati Paskal program koji od korisnika učitava pozitivan realan broj r i ispituje koliko jediničnih kvadratića potpuno leži u unutrašnjosti kruga.

3.55. Papir je paralelnim linijama izdeljen na jedinične kvadratiće. Nacrtan je kružni prsten sa centrom u preseku jednog para normalnih linija i poluprečnicima r i R. Napisati Paskal program koji od korisnika učitava pozitivne realne brojeve r i R, i ispituje koliko jediničnih kvadratića potpuno leži u unutrašnjosti prstena.

3.56. Napisati Paskal program koji traži sve moguće načine da se dešifruje jednakost

$$*****-****=***,$$

gde je svaki od tri broja koji učestvuju u jednakosti palindrom, a zvezdice označavaju proizvoljne cifre. Pri tome, prva cifra u broju nije nula. (Broj je *palindrom* ako se isto čita i sleva i zdesna. Na primer, 12 521 jeste palindrom, a 1 234 nije.)

†**3.57.** Napisati Paskal program koji ispisuje sve moguće načine da se dešifruje sabiranje

$$\begin{array}{cccccc} & & B & O & R \\ & & & \vdots & \\ + & & B & O & R \\ \hline & Š & U & M & A \end{array}$$

gde različitim slovima odgovaraju različite cifre, i pri tome $Š \neq 0$ i $B \neq 0$. U sumi može da učestvuje i više od dva sabirka, a program treba da pronađe sve mogućnosti.

3.9 Ispis

Osim procedure `writeln`, programski jezik Paskal poznaje i proceduru `write` koja ispisuje podatke na monitor i koja je veoma slična proceduri `writeln`. Ove dve procedure se razlikuju samo po tome što `writeln` ispiše to što treba da ispiše i pređe u novi red (dakle, ispis sledeće poruke korisniku će početi u novom redu), dok `write` ispiše to što ima i ostane u istom redu, tako da će sledeća naredba za ispis nastaviti od mesta gde je prethodna stala. U tabeli ispod data su četiri primera. Crticom je označeno mesto gde će naredna naredba nastaviti ispis.

Niz naredbi	MONITOR
`write('Dobar ');`	`Dobar _`
`write('dan')`	`Dobar dan_`
`write('Dobar ');`	`Dobar _`
`writeln('dan')`	`Dobar dan` `_`
`writeln('Dobar ');`	`Dobar` `_`
`write('dan')`	`Dobar` `dan_`
`writeln('Dobar ');`	`Dobar` `_`

`writeln('dan')`	`Dobar`
	`dan`
	`_`

Argumenti procedura `write` i `writeln` mogu da budu poruke korisniku, numeričke promenljive i algebarski izrazi, kao i promenljive drugih tipova koje ćemo kasnije upoznati. Ukoliko se kao argument navede algebraski izraz prvo izračuna vrednost izraza, pa se ona ispiše. Na primer, ako je `n` celobrojna promenljiva čija vrednost je 25, tada

Naredba	*Ispis*
`write('Rezultat je ', n div 2)`	`Rezultat je 12`

Iza svakog argumenta procedure `write` ili `writeln` može da se navede modifikator. Modifikator za realne brojeve smo već koristili. Podsetimo se samo da on izgleda ovako

$$\langle izraz \rangle \ : \ w \ : \ d$$

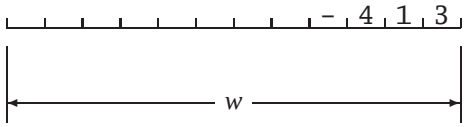
gde je $\langle izraz \rangle$ neki izraz koji kao rezultat daje realan broj, w je širina polja za ispis, a d je broj decimala. Za sve ostale argumente (uključujući i poruku korisniku) modifikator izgleda ovako:

$$\langle arg \rangle \ : \ w$$

gde je $\langle arg \rangle$ neki argument proizvoljnog tipa *koji nije* `real`, a w je širina polja za ispis. Ako je širina polja veća od potrebne, ispis će biti poravnat po desnoj ivici. Na primer, ako želimo da ispišemo broj -413 uz modifikator $w = 12$ to možemo učiniti komandom

```
writeln(-413 : 12)
```

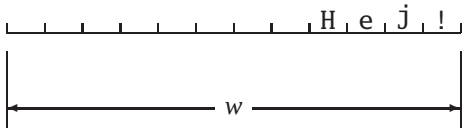
Na monitoru ćemo dobiti:



(Naravno, sve prateće strelice i tarabe se neće videti!) Slično, ako želimo da ispišemo poruku `'Hej!'` uz modifikator $w = 12$ to možemo učiniti komandom

```
writeln('Hej!' : 12)
```

a na monitoru ćemo dobiti:



Primer. Sledeća tabela pokazuje kako izgleda ispis vrednosti celobrojne promenljive `x` uz razne modifikatore, nakon komande `x := -11`:

```
writeln(x)      -11
writeln(x:10)   ________-11
writeln(x:5)    ___-11
writeln(x:2)    -11
writeln(x:0)    -11
```

Primer. Ako je `r = 3.14159` neka realna promenljiva, a `n = -22` neka celobrojna promenljiva, onda naredba

```
write('Brojevi', r:7:3, n:6)
```

daje

```
Brojevi  3.141   -22
        |← 7 →|← 6 →|
```

Primer. Napisati Paskal program koji od korisnika učitava ceo broj n i potom ispisuje prvih n redova sledećeg trougla:

```
1
1  2
1  2  3
1  2  3  4
1  2  3  4  5
.....................
```

```
program Trougao;
var
  i, j, n : integer;
begin
  readln(n);
  for i := 1 to n do
    begin
      for j := 1 to i do
        write(j : 3);
      writeln
    end
end.
```

☞ *Primetimo da kada se napiše bez argumenata, naredba* `writeln` *samo pređe u novi red!*

Zadaci.

3.58. Napisati Paskal program koji od korisnika učitava ceo broj n i potom iscrtava sledeći kvadrat $n \times n$:

```
1  1  1  ...  1
1  1  1  ...  1
1  1  1  ...  1
.....................
1  1  1  ...  1
```

3.59. Napisati Paskal program koji od korisnika učitava ceo broj n i potom ispisuje prvih n redova sledećeg trougla:

```
1
1 1
1 1 1
1 1 1 1
1 1 1 1 1
.....................
```

3.60. Napisati Paskal program koji od korisnika učitava ceo broj n i potom ispisuje prvih n redova sledećeg trougla:

```
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
.....................
```

3.61. Napisati Paskal program koji od korisnika učitava paran broj n i potom iscrtava sledeći kvadrat $n \times n$ (prvih $n/2$ vrsta je popunjeno nulama, a poslednjih $n/2$ vrsta jedinicama):

```
0 0 0 ... 0
0 0 0 ... 0
.....................
0 0 0 ... 0
1 1 1 ... 1
1 1 1 ... 1
.....................
1 1 1 ... 1
```

3.62. Napisati Paskal program koji od korisnika učitava paran broj n i potom iscrtava sledeći kvadrat $n \times n$ (prvih $n/2$ kolona je popunjeno nulama, a poslednjih $n/2$ kolona jedinicama):

```
0 0 ... 0 1 1 ... 1
0 0 ... 0 1 1 ... 1
0 0 ... 0 1 1 ... 1
.....................................
0 0 ... 0 1 1 ... 1
```

3.63. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 1$ i potom ispisuje kvadrat brojeva formata $2n \times 2n$ koji se sastoji od četiri manja kvadrata, kako je to pokazano u primerima:

$n = 1$

```
0 1
1 0
```

$n = 2$

```
0 0 1 1
0 0 1 1
1 1 0 0
1 1 0 0
```

$n = 3$

```
0 0 0 1 1 1
0 0 0 1 1 1
0 0 0 1 1 1
1 1 1 0 0 0
1 1 1 0 0 0
1 1 1 0 0 0
```

3.64. Napisati Paskal program koji ispisuje prvih n redova sledećeg trougla:

```
 1
 2  3
 4  5  6
 7  8  9 10
11 12 13 14 15
.....................
```

3.65. Napisati Paskal program koji ispisuje prvih n redova sledećeg trougla:

```
1
2 3 2
3 4 5 4 3
4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
.................................
```

3.66. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj n i potom ispisuje kvadrat brojeva kako sledi:

Opšti slučaj

1	2	3	…	$n-1$	n
2	3	4	…	n	1
3	4	5	…	1	2
⋮	⋮	⋮		⋮	⋮
n	1	2	…	$n-2$	$n-1$

Posebno za $n = 5$

1	2	3	4	5
2	3	4	5	1
3	4	5	1	2
4	5	1	2	3
5	1	2	3	4

3.67. Napisati Pascal program koji od korisnika učitava pozitivan ceo broj n i potom brojeve $1, 2, \ldots, n^2$ ispisuje u obliku kvadrata dimenzija $n \times n$ kako je to pokazano u primeru pored za $n = 5$.

$n = 5$

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

3.68. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 2$ i potom ispisuje kvadrat formata $n \times n$ u čija polja su upisani brojevi 1, 2, ..., n^2 kako je to pokazano u primerima:

$n = 2$

1	2
4	3

$n = 3$

1	2	3
6	5	4
7	8	9

$n = 4$

1	2	3	4
8	7	6	5
9	10	11	12
16	15	14	13

3.10 Rezime

⟨*Tip*⟩ ≡ `real` | `integer` | `longint` ...(ima ih još mnogo!)

⟨*Naredba*⟩ ≡ ⟨*Naredba dodele*⟩ | ⟨*If naredba*⟩ | ⟨*For naredba*⟩ | ...
(ima ih još mnogo!)

⟨*For naredba*⟩ ≡ ⟨*Rastući for*⟩ | ⟨*Opadajući for*⟩

⟨*Rastući for*⟩ ≡ `for` ⟨*ime*⟩ `:=` ⟨*izraz*$_1$⟩ `to` ⟨*izraz*$_2$⟩ `do`
⟨*Blok ili tačno jedna naredba*⟩

⟨*Opadajući for*⟩ ≡ `for` ⟨*ime*⟩ `:=` ⟨*izraz*$_1$⟩ `downto` ⟨*izraz*$_2$⟩ `do`
⟨*Blok ili tačno jedna naredba*⟩

Glava 4

Nizovi

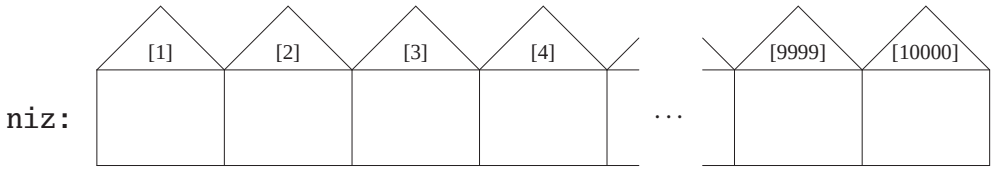
Nizovi u programskom jeziku Paskal predstavljaju način da se efektivno radi sa velikim brojem promenljivih istog tipa. Na primer, problemi koji zahtevaju da se istovremeno obrađuje 10000 brojeva bi se veoma teško mogli realizovati tako što bi se deklarisalo 10000 različitih promenljivih i onda radilo sa njima. U ovoj glavi upoznajemo se sa nizovima kao strukturom podataka programskog jezika Paskal.

4.1 Deklaracija niza

Evo primera promenljive nizovnog tipa:

```
var
  niz : array [1 .. 10000] of integer;
```

(*array* = niz, engl.). Tada ime `niz` označava složenu promenljivu koja se sastoji od 10000 kućica ("malih" promenljivih), a svaka od tih kućica može da primi jedan ceo broj:

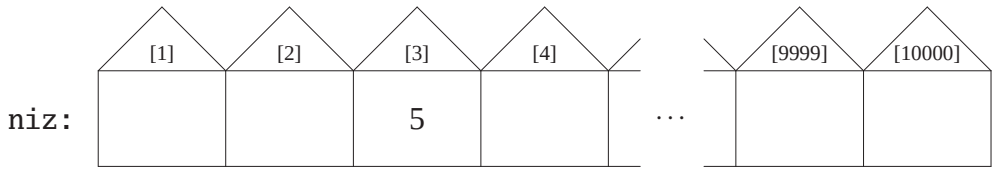


Promenljiva nizovnog tipa liči na ulicu: `niz` je ime ulice, a [1], [2], itd. su kućni brojevi. Pojedinačni elementi niza (tj. pojedinačne kućice) se ponašaju kao obične promeniljive tipa `integer`, a zovu se ovako: `niz[1]`, `niz[2]`, ..., `niz[10000]`. Dakle, ime pojedinačne promenljive se formira tako što se na ime ulice doda kućni

broj kućice. Na primer, ako želimo da u treću kućicu smestimo broj 5, to radimo komandom

```
niz[3] := 5;
```

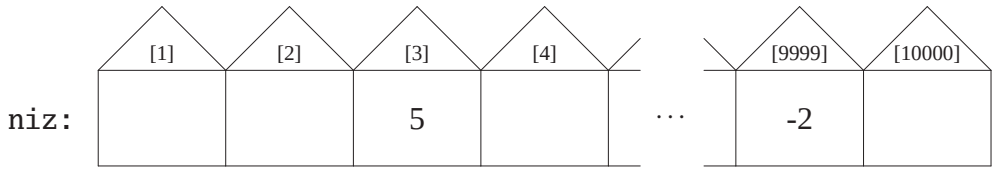
i efekt je sledeći:



a nakon naredbe

```
niz[9999] := 2 * niz[3] - 12;
```

imamo da je `niz[9999] = -2`, tj.



Tip koji se navodi iza ključne reči `of` može biti *bilo koji tip programskog jezika Paskal!* Primeri:

```
var
    s : array [1 .. 341]  of integer;
    r : array [1 .. 17]   of real;
    l : array [1 .. 1826] of longint;
```

kao i svaki drugi tip koga ćemo učiti kasnije tokom kursa.

Primer. Napisati Paskal program koji od korisnika učitava niz od n realnih brojeva ($1 \leqslant n \leqslant 1000$) i određuje i štampa zbir elemenata tog niza.

Rešenje.

```
program ZbirElNiza;
const
  MaxEl = 1000;
var
```

```
  n, i : integer;
  a : array [1 .. MaxEl] of real;
  sum : real;
begin
  write('Unesite n'); readln(n);

  writeln('Sada unesite brojeve:');
  for i := 1 to n do readln(a[i]);

  sum := 0.0;
  for i := 1 to n do sum := sum + a[i];

  writeln('Zbir je ', sum)
end.
```

Primer. Napisati Paskal program koji od korisnika učitava niz od n realnih brojeva ($1 \leqslant n \leqslant 1000$) i određuje i štampa najveći element tog niza, kao gde se od javlja u nizu.

Rešenje.

```
program MaxNiza;
const
  MaxEl = 1000;
var
  n, i, poz : integer;
  a : array [1 .. MaxEl] of real;
  max : real;
begin
  write('Unesite n'); readln(n);

  writeln('Sada unesite brojeve:');
  for i := 1 to n do readln(a[i]);

  max := a[1]; poz := 1;
  for i := 1 to n do
    if a[i] > max then
      begin
        max := a[i];
        poz := i
      end;

  writeln('Najve\'ci je ', max);
  writeln('Javlja se na mestu ', poz)
end.
```

Primer. Napisati Paskal program koji od korisnika učitava niz od n celih brojeva ($1 \leqslant n \leqslant 1000$) i određuje i štampa koliko u tom nizu ima neparnih brojeva.

Rešenje.

```
program BrojNeparnih;
const
  MaxEl = 1000;
var
  n, i, br : integer;
  a : array [1 .. MaxEl] of integer;
begin
  write('Unesite n'); readln(n);

  writeln('Sada unesite brojeve:');
  for i := 1 to n do readln(a[i]);

  br := 0;
  for i := 1 to n do
    if odd(a[i]) then
      inc(br);

  writeln('Ima ', br, ' neparnih')
end.
```

Primer. Nizovi se često koriste kada je potrebno računati sa brojevima koji imaju mnogo cifara ili mnogo decimala. Navodimo primer programa koji računa decimale broja π koristeći Spigot algoritam[1]. Nije lako objasniti zašto ovaj algoritam radi bez dubljeg poznavanja tajni matematike, tako da ga nećemo detaljno komentarisati. Primetite samo da decimale broja π smeštamo u niz a.

```
program PiSpigot;
const
  maxn = 1000;
  maxlen = 3333; {10 * maxn div 3}
var
  i, j, k, q, x, n, len, nines, predigit : longint;
  a : array[1..maxlen] of longint;
begin
  write('Broj decimala (bar 2, najvise ', maxn - 1, ') ->');
  readln(n);
  if (n < 2) or (n >= maxn) then
```

[1] S. Rabinowitz and S. Wagon, A spigot algorithm for the digits of π, Amer. Math. Monthly 102 (1995) 195–203.

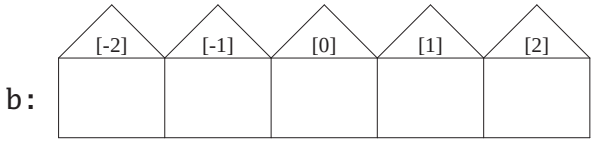
```
      begin
        writeln('Uneti broj nije u navedenom opsegu');
        HALT
      end;
    n := n + 1; len := 10 * n div 3;
    for j := 1 to len do a[j] := 2;
    nines := 0; predigit := 0;
    for j := 1 to n do begin
      q := 0;
      for i := len downto 1 do begin
        x := 10*a[i] + q*i;
        a[i] := x mod (2*i-1);
        q := x div (2*i-1);
      end;
      a[1] := q mod 10; q := q div 10;
      if q = 9 then nines := nines + 1
      else if q = 10 then begin
        write(predigit+1);
        for k := 1 to nines do write(0);
        predigit := 0; nines := 0
      end
      else begin
        if j = 2 then write(predigit, '.')
        else if j >= 3 then write(predigit);
        predigit := q;
        if nines <> 0 then begin
          for k := 1 to nines do write(9);
          nines := 0
        end
      end
    end;
    writeln(predigit)
  end.
```

4.2 Indeksi niza

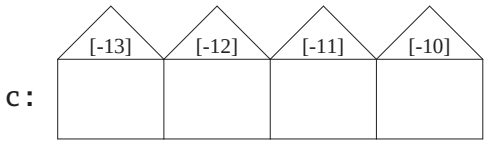
Jedna od lepih osobina nizova u Paskalu je to što prvi indeks niza ne mora biti 1, već može biti bilo koji broj. Na primer, sledeće deklaracije su ispravne:

```
var
    a : array [0 .. 49] of real;
    b : array [-2 .. 2] of integer;
    c : array [-13 .. -10] of longint;
```

Niz b izgleda ovako:



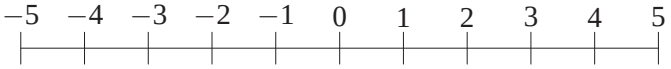
a niz c ovako:



Međutim, *sledeće nije dozvoljeno:*

```
const
  Pi = 3.1415;
  DvaPi = 6.283;
var
  ISNOGOOD : array [Pi .. DvaPi] of integer;
  (* NE SME OVAKO! *)
```

Primer 1. Učenici Gimnazije "Jovan Jovanović-Zmaj" anketirani su kako bi se utvrdilo koliko su zadovoljni ukupnim stanjem u školi. Svaki učenik je dobio anketni listić sa skalom



na kojoj je trebalo da zaokruži jedan od navedenih brojeva. Napisati Paskal program koji od korisnika učitava ceo broj *n* (broj anketiranih učenika), potom *n* brojeva iz skupa $\{-5,\ldots,-1,0,1,\ldots,5\}$ i ispisuje koliko se puta koji broj pojavio kao ocena.

```
program Anketa;
var
  BrUcenika, i, k : integer;
  frekv           : array [-5 .. 5] of integer;
begin
  for i := -5 to 5 do frekv[i] := 0;
  write('Unesite broj anketiranih ucenika -> ');
  readln(BrUcenika);
```

```
  writeln('Unesite rezultate ankete:');
  for i := 1 to BrUcenika do
    begin
      write('Ucenik ', i, ' -> ');
      readln(k);
      if (-5 <= k) and (k <= 5) then
        frekv[k] := frekv[k] + 1
      else
        writeln('Nekorektan podatak -- odbacuje se!')
    end;

  writeln('Frekvencije:');
  for i := -5 to 5 do
    writeln(i:3, frekv[i]:5)
end.
```

Zadaci.

4.1. Napisati Paskal program koji od korisnika učitava niz od n realnih brojeva ($1 \leqslant n \leqslant 1000$) i određuje i štampa najmanji od učitanih brojeva, kao i njegovu poziciju u nizu.

4.2. Napisati Paskal program koji od korisnika učitava niz od n realnih brojeva ($1 \leqslant n \leqslant 1000$) i određuje i štampa prosečnu vrednost elemenata u nizu.

4.3. Napisati Paskal program koji od korisnika učitava niz od n realnih brojeva ($1 \leqslant n \leqslant 1000$) i utvrđuje koliko njih je strogo iznad proseka svih učitanih brojeva.

4.4. Napisati program koji od korisnika učitava n celih brojeva ($1 \leqslant n \leqslant 1000$) i utvrđuje indeks i vrednost prvog člana u nizu koji je najbliži proseku elemenata niza.

4.5. Knjižara na zalihama ima n naslova, $1 \leqslant n \leqslant 5000$. Pri tome, cena i-tog naslova je d_i dinara, a u magacinu ima k_i primeraka te knjige. Napisati program koji od korisnika učitava ceo broj n, $1 \leqslant n \leqslant 5000$, potom n parova brojeva $(d_1,k_1),(d_2,k_2),\ldots,(d_n,k_n)$ koji predstavljaju cene i količine pojedinačnih naslova u magacinu, i potom računa i štampa ukupnu vrednost robe u magacinu.

4.6. Dato je n tačaka u ravni $A_1(x_1,y_1)$, $\ldots$, $A_n(x_n,y_n)$ svojim koordinatama, $1 \leqslant n \leqslant 1000$. Napisati Paskal program koji određuje dužinu najduže duži čiji krajevi su neke od ovih tačaka, kao i redne brojeve tačaka koje postižu

ovu dužinu. Rastojanje tačaka $A(x_1,y_1)$ i $B(x_2,y_2)$ dato je formulom

$$d(A,B) = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}.$$

4.7. Dato je n tačaka u ravni $A_1(x_1,y_1)$, ..., $A_n(x_n,y_n)$ svojim koordinatama, $1 \leqslant n \leqslant 1000$. Napisati Paskal program koji određuje površinu najvećeg trougla sa temenima u ovim tačkama i štampa površinu, kao i redne brojeve temena koji čine taj trougao. Ako su $A(x_1,y_1)$, $B(x_2,y_2)$, $C(x_3,y_3)$ temena trougla onda je

$$P(\triangle ABC) = \frac{1}{2}|(x_2y_3 - x_3y_2) - (x_1y_3 - x_3y_1) + (x_1y_2 - x_2y_1)|.$$

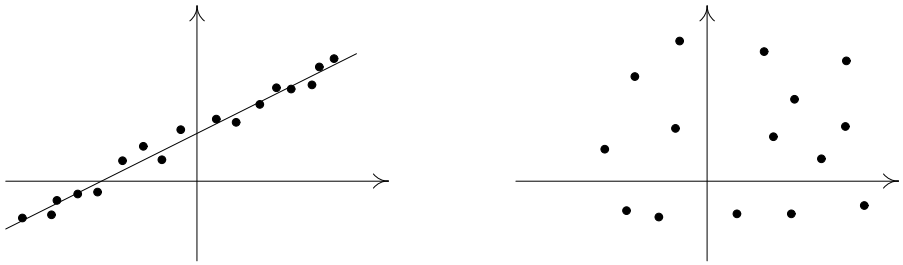
4.8. Napisati program koji od korisnika učitava n celih brojeva ($1 \leqslant n \leqslant 1000$) i utvrđuje koliko ima brojeva iza poslednjeg negativnog broja u nizu. Ako su svi brojevi u nizu nenegativni, program ispisuje n.

4.9. Napisati Paskal program koji učitava niz realnih brojeva i potom nalazi najduži strogo rastući segment u tom nizu, i ispisuje dužinu segmenta i redni broj elementa od koga segment počinje. Na primer, za niz brojeva

```
1 1 2 1 2 3 1 2 3 4 1 2 3 4 5 1 2 3 4 5 6 0 1 2
                3 4 5 6 7 0 1
```

program ispisuje `8 22` zato što je `0 1 2 3 4 5 6 7` najduži strogo rastući segment u tom nizu, njegova dužina je 8 i počinje na 22. mestu. (Napomena: segment čine uzastopni elementi nekog niza.)

4.10. *Linearna regresija* je jedan metod obrađivanja eksperimentalnih rezultata za koje pretpostavljamo da se ponašaju po linearnom modelu $y = ax + b$. Pretpostavimo da smo nizom merenja neke veličine u tačkama x_1, x_2, ..., x_n dobili izmerene vrednosti y_1, y_2, ..., y_n. Nas interesuje da li se izmerene vrednosti grupišu oko neke prave (u kom slučaju fenomen možemo opisati linearnim modelom) ili ne. Na primer, na slici ispod, podaci levo se grupišu oko neke prave, dok za podatke desno to nije slučaj:



Broj koji pokazuje da li se dati podaci ponašaju po linearnom modelu ili ne zove se *koeficijent korelacije* i računa se po formuli:

$$r = \frac{n \cdot \sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{i=1}^{n} y_i\right)}{\sqrt{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \cdot \sqrt{n \cdot \sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}}$$

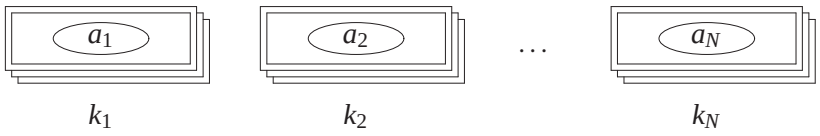
To je realan broj iz intervala $[-1,1]$. Ukoliko je r blizu 1 ili -1, radi se o fenomenu koji se može dobro opisati linearnim modelom (i tada za $r \approx 1$ imamo da će odgovarajuća prava biti rastuća, dok za $r \approx -1$ prava opada), a u ostalim slučajevima se linearni model ne preporučuje.

Nezavisno od vrednosti broja r, za svaki niz eksperimentalnih podataka $(x_1,y_1),\ldots,(x_n,y_n)$ možemo odrediti jednačinu prave koja najbolje aproksimira dati niz brojeva. Ona ima oblik $y = ax+b$ gde je

$$a = \frac{n \cdot \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \quad \text{i} \quad b = \frac{\sum_{i=1}^{n} y_i - a \cdot \sum_{i=1}^{n} x_i}{n}$$

Napisati Paskal program koji od korisnika učitava ceo broj n ($1 \leqslant n \leqslant 1000$), potom n parova realnih brojeva $(x_1,y_1),\ldots,(x_n,y_n)$ i potom računa i štampa veličine r, a i b.

†**4.11.** Na raspolaganju imamo k_1 novčanica u vrednosti od a_1 dinara, k_2 novčanica u vrednosti od a_2 dinara, …, k_N novčanica u vrednosti od a_N dinara, i podaci su tako organizovani da je $a_1 > a_2 > \ldots > a_N$.



Napisati Paskal program koji od korisnika učitava N, $1 \leqslant N \leqslant 100$, potom parove $(k_1,a_1),\ldots,(k_N,a_N)$ i na kraju pozitivan ceo broj m koji predstavlja količinu novca, a onda "isplaćuje" korisniku iznos od m dinara koristeći najmanji mogući broj novčanica.

Ukoliko to nije moguće, isplati korisniku najbolje što može, i obavesti ga o iznosu koji preostaje.

†**4.12.** Napisati Paskal program koji traži sve moguće načine da se dešifruje jednakost

$$***+***=***,$$

gde zvezdice označavaju proizvoljne cifre, i pri tome se svaka od cifara 1, 2, 3, 4, 5, 6, 7, 8, 9 javlja tačno jednom.

4.13. Napisati program koji formira i štampa niz `b[1]`, `b[2]`, ..., `b[m]` koji se dobija od niza `a[1]`, `a[2]`, ..., `a[n]` celih brojeva izbacivanjem svih neparnih brojeva u tom nizu.

4.14. Napisati program koji formira i štampa niz `b[1]`, `b[2]`, ..., `b[m]` koji se dobija od niza `a[1]`, `a[2]`, ..., `a[n]` izbacivanjem najvećeg elementa tog niza i svih njemu jednakih elemenata.

4.15. Napisati program koji formira i štampa niz `b[1]`, `b[2]`, ..., `b[m]` koji se dobija od niza `a[1]`, `a[2]`, ..., `a[n]` izbacivanjem najvećeg i najmanjeg elementa tog niza i svih njima jednakih elemenata.

4.16. Neka je dat niz brojeva $a_1, a_2, \ldots, a_n$. Niz brojeva definisan sa

$$s_k = a_1 + \ldots + a_k$$

zove se *niz parcijalnih suma* polaznog niza. Napisati program koji od korisnika učitava ceo broj n ($1 \leqslant n \leqslant 2000$), potom n brojeva $a_1, a_2, \ldots, a_n$, i računa i ispisuje niz parcijalnih suma ovog niza.

4.17. U redu u samoposluzi se nalazi n kupaca. Kasirka potroši t_k vremenskih jedinica da bi opslužila kupca k, $k \in \{1, \ldots, n\}$. Kupac k na raspolaganju ima f_k vremenskih jedinica za kupovinu. Ako završi kupovinu za više od f_k jedinica, smatraćemo da je zakasnio. Napisati Paskal program koji od korisnika učitava ceo broj n ($1 \leqslant n \leqslant 100$), potom n parova brojeva (t_1, f_1), $\ldots$, (t_n, f_n) i za svakog kupca računa da li će zakasniti ili ne.

4.18. *Konvolucija nizova* $A = (a_1, \ldots, a_n)$ i $B = (b_1, \ldots, b_n)$ je broj koji se računa ovako:

$$A \star B = a_1 b_n + a_2 b_{n-1} + a_3 b_{n-2} + \ldots + a_n b_1.$$

Napisati Paskal program koji od korisnika učitava ceo broj n, potom nizove $A = (a_1, \ldots, a_n)$ i $B = (b_1, \ldots, b_n)$, i računa njihovu konvoluciju.

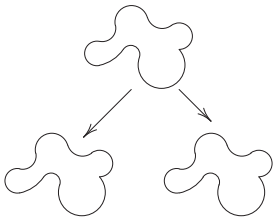
†**4.19.** *Katalanovi brojevi* (*Catalan numbers*) su brojevi definisani na sledeći način:

$$C_0 = 1$$

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}.$$

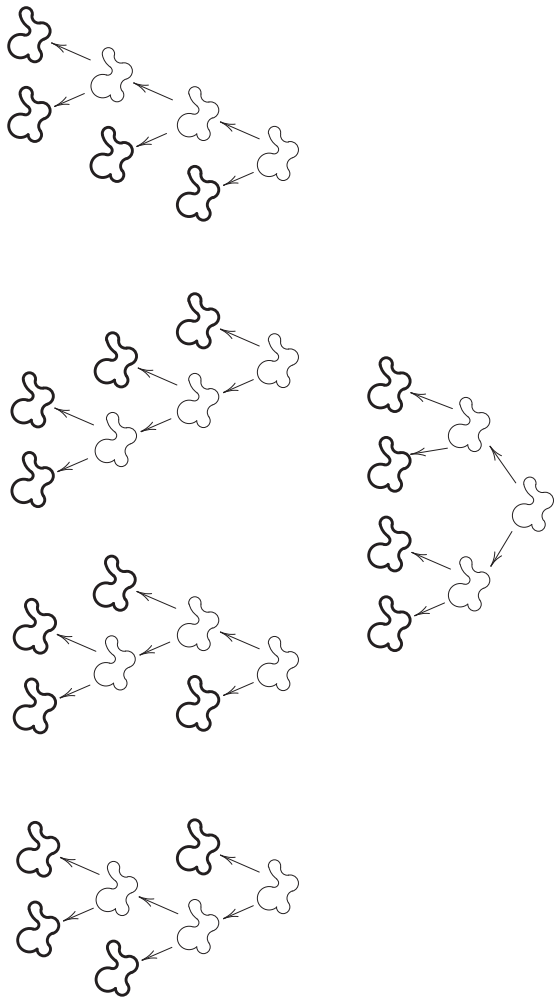
Napisati Paskal program koji od korisnika učitava ceo broj n ($1 \leqslant n \leqslant 100$) i računa n-ti Catalanov broj.

‡**4.20.** Amebe se razmnožavaju prostom deobom, što znači da se jedna ameba prosto podeli na dve:



U jednoj populaciji ameba se nalazi n ameba i znamo da su sve nastale od jedne jedine amebe. Napisati Paskal program koji od korisnika učitava n i određuje na koliko različitih načina se može realizovati niz deoba kojim od jedne amebe nastane n ameba. Na primer za $n = 4$ postoji pet različitih načina da deobama od jedne amebe nastane četiri amebe, kako je to pokazano na Sl. 4.1.

(Uputstvo: Dokazati da se radi o Katalanovim brojevima, pa se tako zadatak svodi na prethodni.)



Slika 4.1: Amebe iz Zadatka 4.20

Glava 5

Logički tip, “while” i “repeat” ciklus

U ovoj glavi ćemo upoznate dalje mogućnosti programskog jezika Paskal. Pre svega uvodimo novi simbolički tip podataka – logički tip. Potom uvodimo dve nove kontrolne strukture – “while” i “repeat” ciklus – i to sve koristimo da naučimo nekoliko izuzetno važnih algoritama: algoritam koji utvrđuje da li je dati broj prost, Euklidov algoritam za određivanje najvećeg zajedničkog delioca dva cela broja, algoritam za efikasno stepenovanje, RSA algoritam za kriptovanje i prošireni Euklidov algoritam kojim rešavamo linearne Diofantove jednačine.

5.1 Logički tip

Logički tip podataka se zove `boolean`, a promenljiva logičkog tipa može da ima jednu od sledeće dve vrednosti: `true` (tačno), ili `false` (netačno). Tip `boolean` programskog jezika Paskal je dobio ime po engleskom matematičaru Džordžu Bulu (George Boole), koji je živeo u drugoj polovini 19. veka.

Primer pored pokazuje kako se deklariše konstanta i promenljiva logičkog tipa, kao i da se promenljivoj logičkog tipa vrednost dodeljuje kao i svakoj drugoj promenljivoj.

```
program MaliPrimer;
const
  NeedsMoreMoney = true;
var
  p, q : boolean;
begin
  p := NeedsMoreMoney;
  q := false
end.
```

Promenljive logičkog tipa koristimo kada treba da registrujemo da li se pojavio neki fenomen ili ne. One se ponekad zovu i *zastavice* (engl. *flag*) zato što možemo da "podignemo zastavicu" kada se desi neki događaj, ili da je "spustimo" ako se događaj nije desio.

☞ *Naravno, moramo voditi računa da se logičkoj promenljivoj ne može dodeliti broj, niti se celobrojnoj ili realnoj promenljivoj sme dodeliti logička vrednost!* ***Dakle, ovo ne sme:***

```
program NeSme1;
var
  p : boolean;
begin
  ...
 p := 13;
  ...
end.
```

```
program NeSme2;
var
  n : integer;
begin
  ...
 n := false;
  ...
end.
```

```
program NeSme3;
var
  x : real;
begin
  ...
 x := true;
  ...
end.
```

```
program NeSme4;
var
  p : boolean;
begin
  ...
 p := 3.14;
  ...
end.
```

Ukoliko pokušate da logičkoj promenljivoj dodelite neki broj, ili bilo šta što nije logička vrednost, Paskal prevodilac će prijaviti grešku. Takođe, ukoliko pokušate promenljivoj koja nije deklarisana kao logička promenljiva da dodelite logičku vrednost, Paskal prevodilac će prijaviti grešku!

5.2 Logički izrazi

Vreme je da se ponovo podsetimo matematičke logike. Sećate li se priče o iskaznim formulama i tautologijama? Imali smo iskazna slova, recimo p, q, r, ..., i logičke operacije $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$, pa smo od svega toga pravili iskazne formule (ili izraze), kao što je:

$$(\neg p \vee q) \wedge r \Rightarrow (q \wedge r).$$

Vrednosti logičkih promenljivih mogu da budu $\top$ ili $\bot$, pa kada nam neko zada konkretne vrednosti za p, q i r, možemo lako da izračunamo vrednost bilo kog logičkog izraza.

Sličnu situaciju imamo i u programskom jeziku Paskal. Iskazna slova su logičke promenljive, dok smo logičke operacije `and`, `or` i `not` upoznali ranije, u nešto drugačijem kontekstu. Logičke izraze sada možemo da pravimo koristeći logičke promenljive, proste uslove i logičke veznike.

Matematika	$\mapsto$	Paskal
p	$\mapsto$	`p`
$\top$	$\mapsto$	`true`
$\bot$	$\mapsto$	`false`
$\wedge$	$\mapsto$	`and`
$\vee$	$\mapsto$	`or`
$\neg$	$\mapsto$	`not`

Matematika	$\mapsto$	Paskal
$(\neg p \vee q) \wedge r$	$\mapsto$	`(not p or q) and r`
$\neg(p \wedge q) \vee \neg(q \vee \neg p)$	$\mapsto$	`not(p and q) or not (q or not p)`

Operatori poređenja <, >, =, <>, >= i <= programskog jezika Paskal se ponašaju kao logičke operacije. Svaki od njih uporedi date izraze i kao rezultat vrati `true` ako dati izrazi stoje u datom odnosu, odnosno, `false` ako ne stoje. Tako, sve što je navedeno u sledećoj tabeli ima smisla, tačno je:

```
(3 < 5) = true
(2 >= 9) = false
(2 + 2 = 5) = false
```

Da li je već postalo suviše apstraktno? Ne bojte se, biće još uzbudljivije! Između `true` i `false` je takođe uvedeno uređenje. Uzima se da je `false` $<$ `true` (sa sledećim opravdanjem: laž nosi manje informacija od istine).

Kao što znamo još od priče sa operacijama na brojevima, nemaju sve operacije isti prioritet (na primer, množenje brojeva ima viši prioritet od sabiranja, tako da se u izrazu $x+yz$ prvo računa yz, pa se na to doda x). Pregled prioriteta operacija u programskom jeziku Paskal je dat u sledećoj tabeli:

Operacija		Prioritet
`(`, `)`	zagrade	4
`not`, `-` (promena znaka)	unarni operatori	3
`*`, `/`, `div`, `mod`, `and`	multiplikativni operatori	2
`+`, `-` (oduzimanje), `or`	aditivni operatori	1
<, >, <=, >=, <>, =	relacioni operatori	0

Operacije višeg prioriteta se izvršavaju pre operacija nižeg prioriteta. Zagrade baš i nisu neke operacije, ali njihovo pojavljivanje u tabeli treba da vas podseti

da se prvo računaju izrazi u zagradama nezavisno od prioriteta operatora koji je u zagradama. Tako se postavljanjem zagrada može izmeniti “prirodan” redosled izvršavanja operacija.

☞ *Sada je jasno zašto se prilikom pravljenja složenih uslova prosti uslovi moraju staviti u zagrade: relacioni operatori su najnižeg prioriteta!*

Na primer:

Ispravno	Pogrešno...	... jer bi bilo shvaćeno kao
`(x = 3) and (y = 4)`	`x = 3 and y = 4`	`(x = (3 and y)) = 4`
`not(x = 3)`	`not x = 3`	`(not x) = 3`
`not(x = 3)`	`x not = 3`	besmislica

5.3 Logički tip i “if” kontrolna struktura

Kada smo pričali o “if” kontrolnoj strukturi, govorili smo da uslov koji se navodi iza “if” može biti “prost” ili “složen”. Na primer,

```
if x < 5 then ...  ili  if (x < 5) or (y >= 12) then ...
```

Sada kada smo naučili više o logičkim izrazima možemo dati opšte pravilo:

☞ *Uslov u “if” strukturi može biti* proizvoljan logički izraz!

Pravi oblik “if” kontrolne strukture izgleda ovako (i analogno za ostale varijante):

⟨*Jednostruki “if”*⟩ ≡ `if` ⟨*logički izraz*⟩ `then`
⟨*Blok ili jedna naredba*⟩
[`else`
⟨*Blok ili jedna naredba*⟩]

Drugim rečima, “if” kontrolnom strukturom možemo proveriti kakvu vrednost ima proizvoljan logički izraz, koga može činiti i samo jedna usamljena logička promenljiva, kao u primeru pored.

```
var
  OK : boolean;
begin
  ...
  if OK then ...
  ...
end.
```

☞ *Evo i prave istine o testu* `odd`*: radi se o ugrađenoj Paskalovoj funkciji koja kao rezultat vraća* `true` *ili* `false`.

Primer. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj *n*, potom *n* celih brojeva i proverava da li su svi neparni.

Problem smo mogli da rešimo i tako što bismo uveli brojač koji bi pamtio broj neparnih brojeva. Na kraju bismo samo proverili vrednost brojača. Međutim, od nas se nije tražilo da utvrdimo koliko je bilo neparnih brojeva, već samo da proverimo da li su svi bili neparni. Zato prikazano rešenje smatramo "elegantnijim" od onog sa brojačem.

```
program PrimerUpotrebeLogProm;
var
  i, k, n : integer;
  SviNep : boolean;
begin
  readln(n);
  SviNep := true;
  for i := 1 to n do
    begin
      writeln('Unesi broj');
      readln(k);
      SviNep := SviNep and odd(k)
    end;

  if SviNep then
    writeln('Svi su neparni')
  else
    writeln('Bilo je i parnih')
end.
```

("Elegancija" u programiranju je dosta neuhvatljiv pojam; pisanje "elegantnih" programa se uči tako što vam neko godinama pokazuje "elegantne" i "neelegantne" programe, i onda vam se odjednom sjasni. Nema boljeg načina.)

Kviz.

1. Da li vam progam `PrimerUpotrebeLogProm` liči na programe sa sumama? Objasnite!

2. Izračunati vrednost izraza (upisati `true` ili `false`):

(a) `(true < false) =` ☐

(b) `(true >= false) =` ☐

(c) `(false <> true) =` ☐

(d) `(false = false) =` ☐

(e) `(true <= true) =` ☐

(f) `(not true <= true) =` ☐

3. Izračunati vrednost izraza (upisati `true` ili `false`):

(a) `(true < false) and (false = false) =` ☐

(b) `not(true >= false) or (false < false) =` ☐

(c) `(false and true) > (false or true) =` ☐

(d) `((false = false) = (true = false)) =` ☐

(e) `(true >= true) and (false or true) =` ☐

4. U kućicu upisati `true` ili `false` tako da ceo izraz bude tačan:

(a) `(true and false) < (false = false) =` ☐

(b) `(true and (` ☐ `< true)) = true`

(c) `((` ☐ `or false) or (` ☐ `and true)) = false`

(d) `(` ☐ `>=` ☐ `) = false`

5. Date su sledeće deklaracije:

```
const
    NeedsMoreMoney = true;
    Epsilon        = 0.001;
    Pi             = 3.14;
    DaysInMarch    = 31;
```

Za svaki od sledećih izraza uvrditi da li ima smisla i ako ima odrediti njegovu vrednost:

```
not(Epsilon < Pi)
not Epsilon < Pi
Epsilon not < Pi
not(NeedsMoreMoney)
not NeedsMoreMoney
(DaysInMarch - 3 = 28) and (Epsilon > 0)
(DaysInMarch = 31) and NeedsMoreMoney
(DaysInMarch * 31) and NeedsMoreMoney
(DaysInMarch = 31) and (NeedsMoreMoney)
DaysInMarch = 31 and NeedsMoreMoney
```

```
(Epsilon = 0) or NeedsMoreMoney
(Epsilon = 0) or not NeedsMoreMoney
(Epsilon = 0) or (not NeedsMoreMoney)
Epsilon = 0 or not NeedsMoreMoney
not(Epsilon = 0) or not NeedsMoreMoney
not Epsilon = 0 or NeedsMoreMoney
```

6. Dva operatora poređenja su posebno interesantni: = i <=. Njihove tabelice su:

=	false	true
false	true	false
true	false	true

i

<=	false	true
false	true	true
true	false	true.

Koje su to poznate logičke operacije?

Zadaci.

5.1. Operacije < i >= su neke logičke operacije. Za svaku od njih napraviti odgovarajuću Bulovu funkciju i predstaviti je pomoću osnovnih logičkih operacija.

5.2. Napisati "pravi oblik" za višestruki "if".

5.3. Zapisati sledeće naredbe bez upotrebe operatora poređenja:

(a) `if OK = true then ...`
(b) `if OK = false then ...`

5.4. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj *n*, a potom još *n* celih brojeva za koje proveri da li su svi pozitivni. (Ne koristiti brojače, već raditi "elegantno"!)

5.5. Napisati Paskal program koji od korisnika učitava ceo broj *n*, a potom još *n* celih brojeva za koje proveri da li je bar jedan veći ili jednak sa 5. (Ne koristiti brojače, već raditi "elegantno"!)

5.4 “While” i “repeat” ciklus

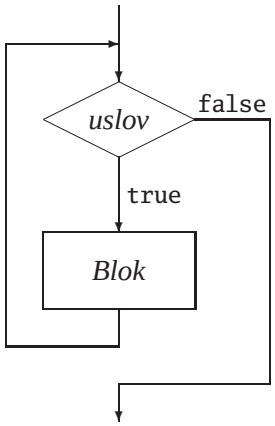
Pogledajmo ponovo program koji ispituje da li je dati broj prost upotrebom “for” ciklusa i pretpostavimo da smo za vrednost promenljive `n` uneli 2 000 000 000. Već u četvrtom prolazu kroz “for” ciklus kada `d` dobije vrednost 4, promenljiva `br` će imati vrednost 3 (zato što je broj deljiv sa 1, 2 i 4). To znači da smo već u četvrtom prolazu kroz ciklus detektovali da broj nije prost, ali će program morati da se vrti još 1 999 999 996 puta kroz petlju pre nego što uspe da to i prijavi.

```
program Prost;
var
  d, n, br : longint;
begin
  writeln('Unesi n');
  readln(n);
  br := 0;
  for d := 1 to n do
    if n mod d = 0 then
      br := br + 1;
  if br = 2 then
    writeln('Prost')
  else
    writeln('Nije prost')
end.
```

Odatle zaključujemo da “for” ciklus nije najpogodnije oruđe za rešavanje ovakvih problema. Voleli bismo da imamo neku vrstu ciklusa koji može da se prekine *čim ustanovi da se nešto desilo*, a ne da se neumitno vrti svojih 2 000 000 000 puta. Programski jezik Paskal poseduje dve takve konstrukcije: “while” ciklus, i “repeat” ciklus.

“While” ciklus ima sledeći oblik:

`while` ⟨*uslov*⟩ `do`
⟨*Blok ili tačno jedna naredba*⟩



gde je ⟨*uslov*⟩ proizvoljan logički izraz. Blok ili naredbu u “while” ciklusu zovemo *telo ciklusa*. “While” ciklus radi na sledeći način:

sve dok je ispunjen navedeni uslov
izvršavaj naredbe iz bloka

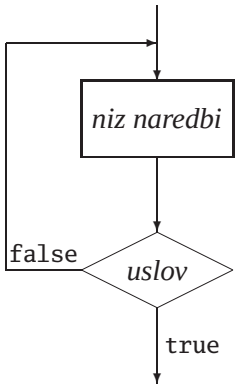
Primer. Napisati Paskal program koji utvrđuje koliko cifara ima dati prirodan broj.

```
program BrojCifara;
var
  n, BrCif : longint;
begin
  writeln('n = ');
  readln(n);
  BrCif := 0;
  while n > 0 do
    begin
      n := n div 10;
      BrCif := BrCif + 1
    end;
  writeln(BrCif)
end.
```

Program koristi sledeću ideju: podeliti broj celobrojno sa 10 je isto što i skratiti ga za poslednju cifru. Zato, sve dok je broj pozitivan (tj. sve dok "ima cifara") radimo sledeće: delimo ga sa 10 i uvećavamo brojač cifara za jedan. Kada `n` padne na nulu to znači da smo mu "otkinuli" sve cifre jednu po jednu, i zato brojač `BrCif` sadrži broj cifara broja `n`. Program radi koraktno samo za pozitivne cele brojeve `n`. (Šta će program ispisati ako korisnik unese 0 ili negativan broj?)

"Repeat" ciklus ima sledeći oblik:

```
repeat
    ⟨naredba_1⟩;
    ⟨naredba_2⟩;
      ⋮
    ⟨naredba_{k-1}⟩;
    ⟨naredba_k⟩
until ⟨uslov⟩
```



gde je ⟨*uslov*⟩ proizvoljan logički izraz, a ⟨*naredba*$_1$⟩, ..., ⟨*naredba*$_k$⟩ su proizvoljne naredbe. "Repeat" ciklus radi na sledeći način:

izvršava se navedeni spisak naredbi sve dok navedeni uslov ne postane tačan.

☞ *Kod "repeat" ciklusa nam ne trebaju "begin" i "end", zato što "repeat" i "until" služe kao "zagrade". Ovo je jedino odstupanje od opšteg pravila programskog jezika Paskal da svuda gde treba da se izvrši niz naredbi, taj niz naredbi mora biti ograđen sa "begin" i "end".*

Primer. Koren pozitivnog realnog broja a se može približno izračunati Njutnovim iterativnim postupkom ovako:

$$x_1 = \frac{a}{2},\ x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right).$$

Napisati Paskal program koji ovim postupkom računa koren pozitivnog realnog broja na 4 tačne decimale. To znači da treba da se vrtimo u petlji sve dok ne dobijemo aproksimaciju x_n za koju je $|x_n - x_{n-1}| < 10^{-4}$. Tada x_n uzimamo za približnu vrednost korena.

```
program KorenNa4Dec;
var
  x1, x2, a : real;
begin
  writeln('a = '); readln(a);
  if a < 0 then
    writeln('Greska')
  else
    begin
      x2 := a/2;
      repeat
        x1 := x2;
        x2 := 0.5*(x1 + a/x1)
      until abs(x2 - x1) < 0.0001;
      writeln('koren = ', x2:10:4)
    end
end.
```

Osnovna razlika između "while" i "repeat" ciklusa je u sledećem:

- kod "while" ciklusa uslov se proverava pre nego što izvršimo blok; zato se može desiti da se blok ne izvrši nijednom u slučaju da je uslov odmah na početku netačan;
- kod "repeat" ciklusa se prvo izvrše sve navedene naredbe, pa se tek onda proverava uslov; dakle, navedene naredbe se izvrše bar jednom.

Paskal poznaje obe vrste ciklusa zato što je nekad zgodnije koristiti jedan, a nekad drugi. Ne postoji spisak pravila koji reguliše kada se koja vrsta ciklusa koristi. Štaviše, postoje problemi koji se mogu rešiti i na jedan i na drugi način. Ukus i navike programera odlučuju kada će upotrebiti koju vrstu ciklusa.

☞ *Za razliku od "for" ciklusa koji se sigurno završava, ove dve vrste ciklusa ne moraju da se završe! Na primer, sledeća dva ciklusa*

```
k := 1;
while k > 0 do
    k := k + 1;
```

```
k := 1;
repeat
    k := k + 1
until k = 0;
```

će se završiti tek kada nestane stuje.

Zato uvek moramo imati nešto što raste (ili opada) i tako se približava uslovu za izlazak iz ciklusa. To je najčešće neka promenljiva čija vrednost raste (ili se

smanjuje) u svakom prolazu kroz ciklus. Kadgod napišete "while" ili "repeat" ciklus, proverite obavezno da li će se završiti. Na primer, u ciklusu

```
k := 0;
repeat
    k := k + 1;
    writeln(k, 'Hello!')
until k = 10;
```

vrednost promenljive k raste u svakom prolazu kroz ciklus. Kako smo krenuli od nule, ciklus će se *sigurno* završiti.

Primer. Otplata duga primenom konformne kamatne stope se realizuje tako što svakog meseca dužnik uplaćuje neki iznos (rata), a kamata se obračunava na ostatak duga. Kada ostatak duga postanje manji od rate, dug se likvidira odjednom. Pri tome se iznos rate i kamatna stopa regulišu ugovorom između banke i dužnika. Na primer, ako otplaćujemo dug u visini od 150.000,00 din primenom konformne kamatne stope od 1,83% na mesečnom nivou i sa visinom rate od 15.000,00 din, plan otplate kredita izgleda ovako:

Mesec	*Staro stanje*	*Kamata*	*Rata*	*Novo Stanje*
1.	150.000,00	2.745,00	15.000,00	137.745,00
2.	137.745,00	2.520,73	15.000,00	125.265,73
3.	125.265,73	2.292,36	15.000,00	112.558,10
4.	112.558,10	2.059,81	15.000,00	99.617,91
5.	99.617,91	1.823,01	15.000,00	86.440,92
6.	86.440,92	1.581,87	15.000,00	73.022,79
7.	73.022,79	1.336,32	15.000,00	59.359,10
8.	59.359,10	1.086,27	15.000,00	45.445,37
9.	45.445,37	831,65	15.000,00	31.277,03
10.	31.277,03	572,37	15.000,00	16.849,39
11.	16.849,39	308,34	15.000,00	2.157,74
12.	2.157,74	39,49	2.197,23	0,00

Vidimo da će dug biti otplaćen za 12 meseci i da će banka zaraditi 17.197,23 din (zarada banke se dobija kao zbir brojeva u koloni *Kamata*).

Napisati Paskal program koji od korisnika učitava visinu duga, kamatnu stopu na mesečnom noviou i visinu rate, a onda štampa plan otplate duga i računa za koliko meseci će dug biti otplaćen, kao i kolika je zarada banke.

```
program KonformnaOtplataDuga;
var
  stanje : real;
  mesKamStopa : real;
  kamata : real;
  rata : real;
  zaradaBanke : real;
  n : integer;
begin
  write('Visina duga -> '); readln(stanje);
  write('Kamatna stopa na mesecnom nivou (%) -> ');
  readln(mesKamStopa);
  write('Rata -> '); readln(rata);
  zaradaBanke := 0;
  n := 0;
  writeln('Mesec Staro stanje    Kamata      Rata Novo Stanje');
  while stanje > rata do begin
    n := n + 1;
    kamata := stanje * mesKamStopa / 100;
    zaradaBanke := zaradaBanke + kamata;
    write(n : 4, '. ');
    write(stanje : 12 : 2);
    write(kamata : 10 : 2);
    write(rata : 10 : 2);
    stanje := stanje + kamata - rata;
    writeln(stanje : 12 : 2)
  end;
  if stanje >= 0.01 then begin
    n := n + 1;
    kamata := stanje * mesKamStopa / 100;
    zaradaBanke := zaradaBanke + kamata;
    rata := kamata + stanje;
    write(n : 4, '. ');
    write(stanje : 12 : 2);
    write(kamata : 10 : 2);
    write(rata : 10 : 2);
    writeln(0.0 : 12 : 2)
  end;
  writeln('Dug ce biti otplacen za ', n, ' meseci');
  writeln('Zarada banke iznosi: ', zaradaBanke : 12 : 2)
end.
```

Zadaci.

5.6. (while) Napisati Paskal program koji za pozitivan ceo broj n određuje najveći broj k takav da je n deljiv sa 2^k.

5.7. (while) Napisati Paskal program koji za pozitivne cele brojeve n i m određuje najveći broj k takav da je n deljiv sa m^k.

5.8. (*Sirakuza algoritam*) Za proizvoljan prirodan broj n uradimo sledeće:

- ako je paran, podelimo ga sa 2;
- ako je neparan, pomnožimo ga sa 3 i dodamo 1.

Hipoteza teorije brojeva poznata pod imenom *Sirakuza algoritam* tvrdi da ćemo nakon konačno mnogo koraka doći do jedinice, nezavisno od prirodnog broja od koga smo krenuli. Na primer:

$$\begin{aligned} 7 &\to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to \\ &\to 5 \to 16 \to 8 \to 4 \to 2 \to 1; \\ 113 &\to 340 \to 170 \to 85 \to 256 \to 128 \to 64 \to 32 \to 16 \to 8 \to \\ &\to 4 \to 2 \to 1. \end{aligned}$$

(a) Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 2$ i potom utvrđuje u koliko koraka se Sirakuza algoritmom dolazi do broja 1 počev od učitanog broja.

(b) Naći Sirakuza rekordera do 100 000.

5.9. (repeat) Napisati Paskal program koji učitava prirodne brojeve n i $b \geqslant 2$ i ispisuje tabelu koja objašnjava postupak ručne konverzije broja n u zapis sa osnovom b. Na primer, za $n = 117$ i $b = 5$ program ispisuje:

117	2
23	3
4	4
0	

5.10. (repeat) Neka je $a_1 = 5$, $a_2 = 7$, a svaki sledeći element niza se dobija kao poslednja cifra zbira prethodna dva elementa niza. Napisati Paskal program koji nalazi broj $n > 2$ sa osobinom $a_n = 5$ i $a_{n+1} = 7$.

5.11. (repeat) Da se podsetimo: Fibonačijev niz je niz brojeva definisan na sledeći način: $F_1 = 1$, $F_2 = 1$ i $F_{n+2} = F_{n+1} + F_n$. Napisati Paskal program koji određuje kada će se u nizu poslednjih cifara Fibonačijevog niza brojeva ponoviti 1, 1.

5.12. (while & repeat) Napisati Paskal program koji za prirodan broj n i prost broj p određuje najveći broj k takav da je $n!$ deljiv sa p^k. Uputstvo: koristiti formulu

$$k = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \left\lfloor \frac{n}{p^4} \right\rfloor + \dots$$

5.13. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj n i određuje sa koliko nula se završava broj $n!$. Napomena: *nipošto ne računati* $n!$

5.14. (repeat) Napisati Paskal program koji nalazi najmanji prirodan broj veći od 1 koji je jednak zbiru kubova svojih cifara.

5.15. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj n, potom učitava n pozitivnih celih brojeva i računa koliko se puta pojavljuje svaka od cifara $0, \dots, 9$ u decimalnim zapisima tih brojeva.

5.16. Funkcija $\sin x$ se može približno izračunati ovako:

$$\sin x \approx \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}.$$

Pri tome, što je n veće, to je i vrednost izraza na desnoj strani tačnija (bliža stvarnoj vrednosti). Napisati Paskal program koji od korisnika učitava realan broj x i na opisani način računa približnu vrednost broja $\sin x$. Sumu računati dok ne dođemo do sabirka čija apsolutna vrednost je manja od 10^{-5}.

"While" ciklus je izuzetno moćan alat, što ćemo pokazati u nekoliko "teorijskih" zadataka.

5.17. Ciklusi "while" i "repeat" nisu nezavisni u sledećem smislu: ako imamo jedan od njih, uvek pomoću njega i "if" možemo simulirati onaj drugi.

(a) Opisati kako se pomoću "while" i "if" može simulirati "repeat".

(b) Opisati kako se pomoću "repeat" i "if" može simulirati "while".

5.18. Simulirati rad

(a) rastućeg "for"ciklusa

(b) opadajućeg "for"ciklusa

pomoću "while"petlje.

5.19. Simulirati rad "if" kontrolne strukture pomoću "while" ciklusa. (Napomena: radimo pod pretpostavkom da uslov "if" kontrolne strukture nema bočnih efekata, štagod to značilo.)

5.20. Napisati Paskal program koji od korisnika učitava brojeve iz skupa $\{-10, \ldots, -1, 0, 1, \ldots, 10\}$ i broji koliko se puta svaki od njih pojavio. Kraj unosa označen je brojem 99 koji se, naravno, ne računa.

5.21. Napisati Paskal program koji od korisnika učitava prirodan broj n i utvrđuje koliko se puta koja cifra javlja u njegovom zapisu, tj. koliko ima nula, koliko jedinica, koliko dvojki, itd, koliko devetki.

5.22. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj n, potom učitava n pozitivnih celih brojeva i računa koliko se puta pojavljuje svaka od cifara $0, \ldots, 9$ u decimalnim zapisima tih brojeva. Na primer, za brojeve 12, 354, 121, 10000, 1, 990 dobijamo:

```
Cifra 0 -> 5 puta
Cifra 1 -> 5 puta
Cifra 2 -> 2 puta
Cifra 3 -> 1 puta
Cifra 4 -> 1 puta
Cifra 5 -> 1 puta
Cifra 9 -> 2 puta
```

5.23. Za niz $(a_1, \ldots, a_n)$ kažemo da je periodičan ako postoji broj p (period niza) takav da je $a_{i+p} = a_i$ za sve $i \in \{1, \ldots, n-p\}$. Napisati Paskal program koji od korisnika učitava niz realnih brojeva, potom ceo broj p i proverava da li je učitani niz periodičan sa periodom p.

5.24. Za niz $(a_1, \ldots, a_n)$ kažemo da je periodičan ako postoji broj p (period niza) takav da je $a_{i+p} = a_i$ za sve $i \in \{1, \ldots, n-p\}$. Napisati Paskal program koji od korisnika učitava niz realnih brojeva i proverava da li je on periodičan. Ako je niz periodičan program ispisuje dužinu perioda, a ako nije ispisuje poruku `NIJE`.

5.5 Prosti brojevi

Od ovog odeljka pa do kraja glave ćemo znanje o novim kontrolnimi strukturama upotrebiti kako bismo demonstrirali neke od najvažnijih algoritama u istoriji civilizacije. Počinjemo sa algoritmom koji proverava da li je dati broj prost.

Za prirodan broj kažemo da je prost ako ima tačno dva delioca. Jedan ne baš inteligentan algoritam kojim se proverava da li je broj prost smo videli na početku

glave. Sada ćemo dati jedan znatno bolji algoritam.

Primetimo, prvo, da ako je $n \geqslant 2$ onda n ima bar dva delioca: 1 i n. Njih zovemo *trivijalni delioci* broja n. Zato se provera da li je n prost svodi na to da se utvrdi da li n ima *netrivijalnog delioca*. Lako se pokazuje sledeća teorema:

Teorema. *Ako broj n ima netrivijalni delilac koji je* $\geqslant \lfloor\sqrt{n}\rfloor$ *onda ima i netrivijalni delilac koji je* $\leqslant \lfloor\sqrt{n}\rfloor$.

Dokaz. Pretpostavimo da n ima netrivijalni delilac d takav da je $d \geqslant \lfloor\sqrt{n}\rfloor$. Neka je $k = n/d$; to je ceo broj zato što je d delilac broja n, i $1 < k < n$ zato što je d netrivijalni delilac broja n. Dakle, i k je netrivijalni delilac broja n. Sada se lako vidi da iz $d \geqslant \lfloor\sqrt{n}\rfloor$ sledi da je $k = n/d \leqslant \lfloor\sqrt{n}\rfloor$. □

Na osnovu toga zaključujemo da je prilikom provere da li je $n \geqslant 2$ prost broj dovoljno proveriti da li on ima netrivijalnog delioca iz skupa $\{2, 3, \ldots, \lfloor\sqrt{n}\rfloor\}$. Pri tome je dovoljno proveriti samo neparne brojeve iz tog skupa, zato što parni brojevi veći od 2 nisu prosti, a parnost broja lako proveravamo.

Pored je dat algoritam koji utvrđuje da li je dati broj prost. Vrednost promenljive `d` raste u svakom prolazu kroz ciklus, tako da će se program *sigurno* završiti. Ciklus se završava čim naiđemo na prvi netrivijalni delilac broja `n`. Deljivost sa 2 proveravamo pre ulaska u ciklus, dok u ciklusu proveravamo samo deljivost neparnim brojevima (početna vrednost za `d` je 3, a u ciklusu se uvećava za dva). Na kraju, podsetimo se da je `n mod d <> 0` logički izraz čija vrednot (`true` ili `false`) se u telu ciklusa dodeljuje logičkoj promenljivoj `prost`.

```
program ProstBroj;
var
  d, n, L : integer;
  prost : boolean;
begin
  readln(n);
  if n < 2 then
    writeln('nije prost')
  else
    begin
      prost := odd(n) or (n = 2);
      d := 3;
      L := trunc(sqrt(n));
      while prost and (d <= L) do
        begin
          prost := n mod d <> 0;
          d := d + 2
        end;
      if prost then
        writeln('prost')
      else
        writeln('slozen')
    end
end.
```

Zadaci.

5.25. Za prost broj p kažemo da je lep ako postoji prost broj q takav da je $p = 2q - 1$. (Na primer, broj 13 je lep prost broj zato što je $13 = 2 \cdot 7 - 1$. Broj 17 je prost ali nije lep zato što je $17 = 2 \cdot 9 - 1$, a 9 nije prost broj.) Napisati Paskal program koji učitava pozitivan broj n i ispituje da li je lep prost broj.

5.26. Napisati Paskal program koji ispisuje sve *proste* delioce datog prirodnog broja n.

5.27. Napisati Paskal program koji od korisnika učitava ceo broj n takav da je $n > 5$ i potom određuje i ispisuje sve proste brojeve p takve da p deli n, ali p^2 ne deli n. Ako takvih brojeva nema, program ispisuje poruku `NEMA`. Na primer, za $n = 36$ program ispisuje `NEMA`, dok za $n = 60$ program ispisuje brojeve 3 i 5.

5.28. Za prirodan broj n sa $\sigma_p(n)$ označavamo sumu svih prostih brojeva iz skupa $\{1, 2, \ldots, n\}$ koji su delioci broja n. Napisati Paskal program koji od korisnika učitava ceo broj n i računa $\sigma_p(n)$ ukoliko je $n > 1$.

5.29. Napisati Paskal program koji od korisnika učitava prirodan broj n, a potom još n prirodnih brojeva za koje proveri da li su svi prosti.

5.30. Napisati Paskal program koji učitava ceo broj n i ukoliko je $n \geqslant 2$ računa

$$p_1^2 + p_2^2 + \ldots + p_k^2,$$

gde su $p_1, p_2, \ldots, p_k$ svi prosti brojevi koji pripadaju skupu $\{2, 3, \ldots, n\}$. Za $n < 2$ program prijavljuje grešku.

5.31. Vreme zapisano u obliku *HH:MM:SS* možemo da shvatimo kao šestocifreni broj $\overline{HHMMSS}$. Na primer, 23:15:46 možemo da shvatimo kao broj 231 546. Napisati Paskal program koji određuje kada ima više prostih brojeva ovog oblika: pre podne (tj. od 00:00:01 do 11:59:59) ili posle podne (tj. od 12:00:01 do 23:59:59).

5.32. Napisati Paskal program koji za dati prirodan broj n određuje njegov najmanji i najveći prost faktor. Na primer, za $n = 14245$ program štampa brojeve 5 i 37 zato što je $14245 = 5 \cdot 7 \cdot 11 \cdot 37$.

5.33. Prosta baza prirodnog broja n je prirodan broj s koji ima iste proste faktore kao i n, s tim da se svaki prost faktor broja n javlja kao faktor broja s tačno jednom. Na primer, za $n = 56 = 2^3 \cdot 7$ je $s = 2 \cdot 7 = 14$, a za $n = 42471 = 3^3 \cdot 11^2 \cdot 13$ je $s = 3 \cdot 11 \cdot 13 = 429$.

Napisati Paskal program koji od korisnika učitava prirodan broj $n \geqslant 2$ i računa i ispisuje njegovu prostu bazu. Za brojeve manje od 2 se prosta

baza ne definiše.

5.34. Napisati Paskal program koji od korisnika učitava prirodan broj n i potom ispisuje prvih n prostih brojeva od najvećeg ka najmanjem. Na primer, za $n = 10$ program ispisuje

29 23 19 17 13 11 7 5 3 2

5.35. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 2$ i utvrđuje da li je on proizvod uzastopnih prostih brojeva. Na primer, sledeći brojevi jesu proizvod uzastopnih prostih brojeva: $6 = 2 \cdot 3$, $2341 = 11 \cdot 13 \cdot 17$, $23 = 23$, a sledeći nisu: $12 = 2^2 \cdot 3$, $21 = 3 \cdot 7$, $286 = 2 \cdot 11 \cdot 13$.

5.6 Najveći zajednički delilac dva broja

Sada ćemo pokazati čuveni Euklidov algoritam za nalaženje *NZD* dva prirodna broja. Iako jednostavan, Euklidov algoritam je izuzetno važan i predstavlja osnovu za mnoge druge algoritme teorije brojeva. On je značajan i istorijski, čak toliko da se ponekad može sresti i ovakva izreka:

Euklidov algoritam je pradeda svih algoritama!

Najveći zajednički delilac (*NZD*) brojeva a i b je prirodan broj d takav da

$(NZD1)$ $d|a$, $d|b$; i

$(NZD2)$ ako je s prirodan broj takav da $s|a$ i $s|b$, onda $s|d$.

Uslov $(NZD1)$ znači da se d sadrži i u a i u b, dok uslov $(NZD2)$ znači da je d najveći takav broj. Brojevi a i b mogu biti i negativni, *ali je NZD dva broja uvek pozitivan broj!*

Teorema. *Neka su a i b prirodni brojevi i neka su brojevi q_i i r_i takvi da je*

$$\begin{array}{lll}
a & = q_1 b + r_1, & 0 < r_1 < b \\
b & = q_2 r_1 + r_2, & 0 < r_2 < r_1 \\
r_1 & = q_3 r_2 + r_3, & 0 < r_3 < r_2 \\
r_2 & = q_4 r_3 + r_4, & 0 < r_4 < r_3 \\
 & \cdots & \\
r_{k-3} & = q_{k-1} r_{k-2} + r_{k-1}, & 0 < r_{k-1} < r_{k-2} \\
r_{k-2} & = q_k r_{k-1} + r_k, & 0 < r_k < r_{k-1} \\
r_{k-1} & = q_{k+1} r_k, & (r_{k+1} = 0).
\end{array}$$

Tada je $r_k = \text{NZD}(a,b)$.

Dokaz. Dokažimo da važe uslovi $(NZD1)$ i $(NZD2)$.

$(NZD1)$: Iz poslednjeg reda vidimo da $r_k|r_{k-1}$. Iz pretposlednjeg onda dobijamo da $r_k|r_{k-2}$, iz reda pre tog imamo da $r_k|r_{k-3}$, itd. Dakle, $r_k|r_2$, $r_k|r_1$, $r_k|b$ i $r_k|a$. Time smo pokazali da se r_k sadrži i u a i u b.

$(NZD2)$: Pretpostavimo sada da je s prirodan broj takav da je $s|a$ i $s|b$. Iz prvog reda dobijamo da tada $s|r_1$. Iz drugog dobijamo da $s|r_2$ itd. Spuštajući se naniže dobijamo, redom, da $s|r_3$, $s|r_4$, $\ldots$, $s|r_{k-2}$, $s|r_{k-1}$ i, napokon, $s|r_k$.

Dakle, r_k je zaista *NZD* brojeva a i b, čime je dokaz završen. □

Pored je dat Paskal program koji računa *NZD* dva pozitivna cela broja. Program pretpostavlja da će korisnik uneti pozitivne brojeve.

```
program Euklid;
var
  a, b, r : integer;
begin
  readln(a, b);
  repeat
    r := a mod b;
    a := b;
    b := r
  until b = 0;
  writeln(a)
end.
```

Zadaci.

5.36. Izračunati "peške" *NZD* sledećih brojeva:

(a)	89 i 55	(c)	437 i 529
(b)	125 i 75	(d)	330 i 121

5.37. Napisati Paskal program koji računa *NZD* za dva *proizvoljna* cela broja. Voditi računa o tome da $NZD(0,0)$ nije definisano, da je $NZD(x,y) = NZD(|x|,|y|)$ ako x i y nisu oba pozitivni, kao i da je $NZD(0,x) = NZD(x,0) = x$ za $x > 0$.

5.38. Za prirodne brojeve a i b kažemo da su uzajamno prosti ako je $NZD(a,b) = 1$. Napisati Paskal program koji učitava prirodne brojeve a, p i q, i štampa sve prirodne brojeve iz intervala $[p,q]$ koji su uzajamno prosti sa a.

5.39. Napisati Paskal program koji računa *NZS* prirodnih brojeva a i b. Koristiti činjenicu da je

$$NZS(a,b) = \frac{ab}{NZD(a,b)}.$$

5.40. (*MRAK* – Mali RAcionalni Kalkulator) Napisati Paskal program koji učitava cele brojeve p_1, q_1, p_2 i q_2 i računa zbir, razliku, proizvod i količnik razlomaka $\frac{p_1}{q_1}$ i $\frac{p_2}{q_2}$. Razlomak $\frac{u}{v}$ koji se dobija kao rezultat treba da bude skraćen i ispisan u obliku u/v. (Na primer, ako je rezultat neke operacije $\frac{4}{6}$, računar treba da ispiše $2/3$.)

5.41. Za prirodan broj n, Ojlerova funkcija $\varphi(n)$ se definiše ovako: $\varphi(n)$ je jednako broju elemenata skupa $\{1,2,3,\ldots,n\}$ koji su uzajamno prosti sa n. Napisati Paskal program koji učitava prirodan broj n i računa i štampa $\varphi(n)$.

5.7 Efikasno stepenovanje

Program koji određuje stepen nekog realnog broja smo već videli, i može se rezimirati kao što je to pokazano pored (i koji radi korektno samo za $n \geqslant 0$). On je veoma jednostavan i lako se pamti, ali nije baš efikasan: ako neki realan broj x dižemo na stepen n, tada se "for" ciklus ovog programa izvršava n puta.

```
program NaivnoStepenovanje;
var
  x, rez : real;
  n, i : integer;
begin
  readln(x, n);
  rez := 1.0;
  for i := 1 to n do
    rez := rez * x;
  writeln(rez)
end.
```

U ovom pododeljku ćemo pokazati *veoma* efikasan način da realan broj x dignemo na stepen n. Odgovarajuća petlja novog programa će se izvršavati onoliko puta koliko cifara ima binarni zapis broja n. Na primer, ako pokušamo da odredimo x^{32000}, prvi, naivan algoritam će se vrteti u petlji 32000 puta, dok će se novi, efikasniji algoritam vrteti u petlji samo 15 puta!

Recimo da želimo da izračunamo x^{154}. Osnovna ideja novog algoritma se sastoji u tome da se izračunaju svi stepeni broja x oblika x^{2^k}:

$$x,\quad x^2,\quad x^4,\quad x^8,\quad x^{16},\quad x^{32},\quad x^{64},\quad x^{128},\ldots$$

i da se potom pomnože samo oni stepeni koji su nam potrebni. Kako je

$$154 = 10011010_{(2)} = 128+16+8+2$$

dobijamo da je $x^{154} = x^{128}\cdot x^{16}\cdot x^8\cdot x^2$. Zato algoritam uzastopnim kvadriranjem računa stepene broja x oblika x^{2^k} i u isto vreme konvertuje broj n iz osnove 10 u osnovu 2. Pri tome, kada naiđe na cifru 1 u binarnom zapisu broja n, neće je

ispisati, nego će odgovarajući stepen broja x domnožiti[1] na promenljivu u kojoj se polako akumulira rezultat. Tabela pored programa pokazuje kako algoritam radi za $n = 154$ i proizvoljan realan broj x.

i	rez	x	n	b_i
0	1	x	154	0
1	$1 \leftarrow x^2$	x^2	77	1
2	x^2	x^4	38	0
3	$x^2 \leftarrow x^8$	x^8	19	1
4	$x^{10} \leftarrow x^{16}$	x^{16}	9	1
5	x^{26}	x^{32}	4	0
6	x^{26}	x^{64}	2	0
7	$x^{26} \leftarrow x^{128}$	x^{128}	1	1
8	x^{154}	x^{256}	0	

```
program EfikasnoStepenovanje;
var
  x, rez: real;
  n : integer;
begin
  readln(x, n);
  if n < 0 then
    writeln('Greska: n < 0')
  else
    begin
      rez := 1.0;
      while n > 0 do
        begin
          if odd(n) then
            rez := rez * x;
          x := sqr(x);
          n := n div 2
        end;
      writeln(rez)
    end
end.
```

5.8 Eratostenovo sito i nizovi logičkih vrednosti

Posebno interesantan način za pronalaženje svih prostih brojeva iz skupa $\{2,3,\ldots,n\}$ pronašao je starogrčki naučnik Eratosten. Svi brojevi od 2 do n se ispišu u niz. Potom neke od njih zaokružujemo, a neke precrtavamo prema jednom veoma jednostavnom pravilu. Na kraju brojevi koji ostanu neprecrtani su prosti. Precrtavanje se radi ovako. Uočimo prvi broj koji nije zaokružen. Neka je to k. Zaokružimo k, a svaki k-ti broj počev od njega precrtamo. Evo primera za $n = 25$. Na početku ispišemo brojeve od 2 do 25:

[1] ovo *nije* reč književnog srpskog jezika, ali nam je u ovom trenutku jako zgodna!

2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21
22	23	24	25						

Prvi neprecrtani broj, a to je 2, zaokružimo i precrtamo svaki drugi broj počev od njega:

(2)	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11
~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~	21
~~22~~	23	~~24~~	25						

Prvi sledeći neprecrtani broj, a to je 3, zaokružimo i precrtamo svaki treći broj počev od njega:

(2)	(3)	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11
~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~
~~22~~	23	~~24~~	25						

Sledeći broj je 5, itd. Kao i ranije, dovoljno je raditi samo sa brojevima koji su manji ili jednaki sa $\lfloor\sqrt{n}\rfloor$. Na kraju dobijamo:

(2)	(3)	~~4~~	(5)	~~6~~	7	~~8~~	~~9~~	~~10~~	11
~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~	~~21~~
~~22~~	23	~~24~~	~~25~~						

Brojevi koji nisu precrtani su prosti, nezavisno od toga da li su zaokruženi ili ne. Kako ceo postupak liči na prosejavanje, ovaj algoritam se zove *Eratostenovo sito*.

Elementi niza mogu da budu i logičke vrednosti, na primer ovako:

```
const
    MaxEl = 1000;
var
    Precrtan : array[2 .. MaxEl] of boolean;
```

To je ideja koja se koristi za implementaciju Eratostenovog sita.

```
program EratostenovoSito;
const
  MaxEl = 1000;
var
  N, i, k  : integer;
  Precrtan : array [2 .. MaxEl] of boolean;
begin
  write('Unesite broj izmedju 5 i ', MaxEl, ' -> ');
  readln(N);
  for k := 2 to N do Precrtan[k] := false;

  for k := 2 to trunc(sqrt(N)) do
    if not Precrtan[k] then
      begin
        (* precrtavanje umnozaka *)
        i := 2 * k;
        while i <= N do
          begin
            Precrtan[i] := true;
            i := i + k
          end;
      end;

  writeln('Prosti brojevi su:');
  for k := 2 to N do
    if not Precrtan[k] then
      writeln(k)
end.
```

5.9 RSA kriptosistem

RSA kriptosistem (Ron Rivest, Adi Shamir i Leonard Adleman) se pojavio 1978. godine i to je prvi kriptosistem sa *javnim ključem*[2]. To znači da se ključ za šifrovanje poruka može bez brige objaviti u novinama, jer se na osnovu njega *ne može u razumnom vremenu* odrediti ključ za dešifrovanje poruke. On je i danas osnova velikog broja bezbednosnih sistema. Na primer, PGP (Pretty Good Privacy) je jedna popularna svima dostupna (open-source) implementacija RSA kriptosistema.

Matematika koja radi iza RSA algoritma je prilično komplikovana. Zato ćemo pokazati samo osnovnu ideju algoritma i procedure za kodiranje i dekodiranje poruke, dok se procedurom za generisanje ključa nećemo baviti.

Generisanje ključa predstavlja prvi korak u radu algoritma. Na slučajan način se odaberu dva *velika* (bar osamdeset cifara) prosta broja p i q, $p \neq q$, i stavimo $n = pq$ i $s = (p-1)(q-1)$. Potom odaberemo broj e takav da je $1 < e < s$ i $NZD(e,s) = 1$. Teorija brojeva nam tada garantuje da postoji tačno jedan broj d takav da je $1 < d < s$ i $(e \cdot d) \bmod s = 1$, pa izračunamo i njega.

Sada je par brojeva n i e *javni ključ* i on se koristi za kodiranje poruka, a d je *privatni ključ* koga čuvamo u tajnosti i on se koristi za dekodiranje poruka. Lepota ideje RSA algoritma je u tome što je jako teško izračunati d ako znamo n i e. Zato n i e ne moramo da krijemo!

Kodiranje poruke se potom obavlja tako što se poruka, koja se najčešće sastoji iz niza slova, prvo na neki unapred dogovoreni način prevede u niz brojeva i onda se kodira broj po broj. Šifra c broja m se sada računa veoma jednostavno:

$$c = m^e \bmod n.$$

Dekodiranje poruke je takođe jednostavno. Ako nam je stigla kodirana poruka c, onda se dekodirana poruka m dobija ovako:

$$m = c^d \bmod n.$$

Primetimo da se poruka kodira javnim ključem, a dekodira tajnim!

Primer. Neka je $p = 11$ i $q = 23$. Tada je $n = pq = 253$ i $s = (p-1)(q-1) = 220$. Broj e biramo tako da bude $1 < e < s$ i $NZD(e,s) = 1$. Najmanji takav broj je 3, pa uzmimo $e = 3$. Sada treba naći d tako da je $1 < d < s$ i $(e \cdot d) \bmod s = 1$. Takav broj d je jednoznačno određen i u našem slučaju je $d = 147$.

Ako je poruka koju treba preneti $m = 19$, onda se šifrovanjem dobija $c = 19^3 \bmod 253 = 28$. Obrnuto, dekodiranjem dobijamo $m = 28^{147} \bmod 253 = 19$.

[2] Rivest R., Shamir A., Adleman L., *A method for obtaining digital signature and public-key cryptosystems*, Communications of the ACM, **21**, 2(1978), 120–126

Implementacija. Dajemo sada implementaciju procedure za šifrovanje/dešifrovanje RSA kriptosistemom. Pošto se i šifrovanje i dešifrovanje obavljaju na isti način, samo sa drugim parametrima, jedan program radi oba posla. Naravno, mi ćemo sada raditi samo sa `longint` brojevima, a rad sa *veoma* velikim brojevima ćemo naučiti kasnije.

Kako d i e mogu biti *veoma* veliki, stepeni m^e i c^d se računaju koristeći efikasno stepenovanje. Na primer, ako e ima 80 cifara, naivna procedura za stepenovanje bi radila vekovima, dok efikasno stepenovanje izvrši oko 250 prolaza kroz petlju. Osim toga, da bi se uštedelo na vremenu, nećemo prvo izračunati m^e pa tek onda odrediti ostatak pri deljenju sa n, nego ćemo taj ostatak računati odmah. Zašto ovo smemo da radimo? Eeeee, to je zbog opšte lepote matematike!

Da bi se stekao bolji utisak o tome koliki su brojevi koji mogu da se pojave prilikom šifrovanja i dešifrovanja, podsetićemo se prethodnog primera gde je bilo potrebno izračunati 28^{147}:

```
program RSA;
var
  n, e, e1, m, c : longint;
  i, duzina : integer;
begin
  writeln('Kljuc n, e ->');
  readln(n); readln(e);
  writeln('Duzina poruke ->');
  readln(duzina);
  for i := 1 to duzina do
    begin
      readln(m);
      c := 1; e1 := e;
      while e1 > 0 do
        begin
          if odd(e1) then
            c := (c * m) mod n;
          m := sqr(m) mod n;
          e1 := e1 div 2
        end;
      writeln('= ', c)
    end
end.
```

$$28^{147} = \begin{array}{l} 53979717492170694886345235374820524114093354687035208569455655176615209810695948338449569071812818689826980000673301917952874672341953607248709057584552613569438656142695889398039998825476269447734228973436220211 2 \end{array}$$

5.10 Prošireni Euklidov algoritam i Diofantove jednačine

Najveći zajednički delilac dva broja ima jednu važnu osobinu: on se može izraziti preko ta dva broja. Na primer,

$$NZD(33,21) = 3 \quad \text{i onda je} \quad 3 = 2 \cdot 33 - 3 \cdot 21,$$

ili

$$NZD(143,121) = 11 \quad \text{i onda je} \quad 11 = -5 \cdot 143 + 6 \cdot 121.$$

Ovo važi za svaka dva pozitivna broja: ako je $d = NZD(a,b)$, onda postoje celi brojevi α i β takvi da je $d = \alpha a + \beta b$. Brojevi α i β se mogu naći Euklidovim algoritmom, kako to pokazuje sledeća teorema.

Teorema. *Neka su a i b prirodni brojevi i neka je $d = \mathrm{NZD}(a,b)$. Tada postoje celi brojevi α i β takvi da je $d = \alpha a + \beta b$.*

Dokaz. Neka su q_i i r_i brojevi takvi da je

$$\begin{array}{lll}
a & = q_1 b + r_1, & 0 < r_1 < b \\
b & = q_2 r_1 + r_2, & 0 < r_2 < r_1 \\
r_1 & = q_3 r_2 + r_3, & 0 < r_3 < r_2 \\
r_2 & = q_4 r_3 + r_4, & 0 < r_4 < r_3 \\
 & \cdots & \\
r_{k-3} & = q_{k-1} r_{k-2} + r_{k-1}, & 0 < r_{k-1} < r_{k-2} \\
r_{k-2} & = q_k r_{k-1} + r_k, & 0 < r_k < r_{k-1} \\
r_{k-1} & = q_{k+1} r_k. &
\end{array}$$

Tada je, kao što znamo, $r_k = d = NZD(a,b)$. Matematičkom indukcijom pokazujemo da za svako $i \in \{1,\ldots,k\}$ postoje celi brojevi α_i i β_i takvi da je $r_i = \alpha_i a + \beta_i b$. Za $i = 1$ uzmimo da je $\alpha_1 = 1$ i $\beta_1 = -q_1$. Tada iz $a_1 = q_1 b + r_1$ direktno sledi da je $r_1 = \alpha_1 a + \beta_1 b$.

Pretpostavimo sada da je tvrđenje tačno za sve indekse manje od i i posmatrajmo broj r_i. Broj r_i je odabran tako da zadovoljava

$$r_{i-2} = q_i r_{i-1} + r_i.$$

Prema induktivnoj hipotezi, postoje brojevi α_{i-1} i β_{i-1} takvi da je

$$r_{i-1} = \alpha_{i-1} a + \beta_{i-1} b.$$

Množenjem ove jednakosti sa q_i dobijamo

$$q_i r_{i-1} = q_i \alpha_{i-1} a + q_i \beta_{i-1} b,$$

pa kako je $r_{i-2} = q_i r_{i-1} + r_i$, to je

$$r_{i-2} - r_i = q_i \alpha_{i-1} a + q_i \beta_{i-1} b.$$

Sada je

$$r_i = -q_i \alpha_{i-1} a - q_i \beta_{i-1} b + r_{i-2},$$

odakle na osnovu induktivne hipoteze za r_{i-2} dobijamo

$$r_i = -q_i \alpha_{i-1} a - q_i \beta_{i-1} b + \underbrace{\alpha_{i-2} a + \beta_{i-2} b}_{r_{i-2}}.$$

Nakon sređivanja izraz dobija oblik

$$r_i = (\alpha_{i-2} - q_i\alpha_{i-1})a + (\beta_{i-2} - q_i\beta_{i-1})b$$

odakle se lako vidi da je $r_i = \alpha_i a + \beta_i b$ za $\alpha_i = \alpha_{i-2} - q_i\alpha_{i-1}$ i $\beta_i = \beta_{i-2} - q_i\beta_{i-1}$. Time je dokaz završen. □

Dokaz prethodne teoreme nam daje i rekurzivne formule na osnovu kojih možemo da odredimo nizove celih brojeva α_i i β_i koji će na kraju dovesti do traženih brojeva α_k i β_k:

$$\begin{aligned}&\alpha_0 = 0,\\ &\alpha_1 = 1,\\ &\alpha_i = \alpha_{i-2} - q_i\alpha_{i-1};\end{aligned}$$

$$\begin{aligned}&\beta_0 = 1,\\ &\beta_1 = -q_1,\\ &\beta_i = \beta_{i-2} - q_i\beta_{i-1}.\end{aligned}$$

Formule za α_i i β_i kao i vrednosti za α_1 i β_1 su preuzete iz dokaza teoreme, dok su vrednosti za α_0 i β_0 određene tako da formule za α_i i β_i za $i = 2$ daju željene vrednosti.

Euklidov algoritam koga smo videli u odeljku 5.6 računa $d = NZD(a,b)$ za date pozitivne cele brojeve a i b. Algoritam koji je naveden pored osim d određuje još i cele brojeve α i β takve da je $d = \alpha a + \beta b$. Zato se on zove *prošireni Euklidov algoritam.*

```
program ProsireniEuklid;
var
  a, b, q, r : integer;
  alfa0, alfa1, alfa2 : integer;
  beta0, beta1, beta2 : integer;
begin
  readln(a, b);
  if (a <= 0) or (b <= 0) then
    writeln('Greska')
  else
    begin
      alfa1 := 0;
      alfa2 := 1;
      beta1 := 1;
      beta2 := -(a div b);
      while a mod b > 0 do
        begin
          r := a mod b;
          a := b;
          b := r;
          q := a div b;
          alfa0 := alfa1;
          alfa1 := alfa2;
          alfa2 := alfa0 - q*alfa1;
          beta0 := beta1;
          beta1 := beta2;
          beta2 := beta0 - q*beta1
        end;
      writeln(alfa1:12, beta1:12)
    end
end.
```

Diofantove jednačine su jednačine sa celobrojnim koeficijentima kod kojih tražimo samo celobrojna rešenja. Linearna Diofantova jednačina sa dve nepoznate je jednačina oblika

$$ax + by = c$$

gde su a, b i c celi brojevi i kod koje tražimo samo celobrojna rešenja. U ovom odeljku ćemo primeniti prošireni Euklidov algoritam da bismo rešili linearne Diofantove jednačine sa dve nepoznate.

Teorema. *Neka su a, b, c celi brojevi različiti od nule i neka je $d = \mathrm{NZD}(a,b)$. Tada jednačina $ax+by=c$ ima rešenje ako i samo ako $d|c$.*

Ako jednačina $ax+by=c$ ima rešenje (x_0,y_0), onda ona ima beskonačno mnogo rešenja i sva rešenja te jednačine su data sa

$$x = x_0 + t \cdot B, \qquad y = y_0 - t \cdot A,$$

gde je $A = a/d$, $B = b/d$ i t je proizvoljan ceo broj.

Dokaz. Ako jednačina $ax+by=c$ ima rešenje (x_0,y_0), onda je $ax_0+by_0=c$, pa kako $d|(ax_0+by_0)$, dobijamo $d|c$. Obrnuto, neka $d|c$ i neka je $A=a/d$, $B=b/d$ i $C=c/d$. Deljenjem jednačine $ax+by=c$ sa d lako se vidi da mora biti $Ax+By=C$, kao i da ove dve jednačine imaju ista rešenja. Zato da bismo pokazali da jednačina $ax+by=c$ ima rešenje, dovoljno je da pokažemo da jednačina $Ax+By=C$ ima rešenje. Zbog $d=NZD(a,b)$ i $A=a/d$, $B=b/d$ lako se vidi da je $NZD(A,B)=1$, pa prema teoremi o proširenom Euklidovom algoritmu postoje celi brojevi α i β takvi da je $A\alpha+B\beta=1$, odakle množenjem sa C dobijamo $A(C\alpha)+B(C\beta)=C$. Dakle, $x_0=C\alpha$, $y_0=C\beta$ je rešenje jednačine $Ax+By=C$, pa i jednačine $ax+by=c$.

Pretpostavimo sada da je $ax_0+by_0=c$. Tada je i $Ax_0+By_0=C$, gde je $A=a/d$, $B=b/d$ i $C=c/d$. Nije teško proveriti da je $x=x_0+t\cdot B$, $y=y_0-t\cdot A$, rešenje jednačine $Ax+By=C$ za svaki ceo broj t, odakle sledi da jednačina $ax+by=c$ ima beskonačno mnogo rešenja. Ostalo je još samo da se dokaže da svako rešenje te jednačine ima oblik $x=x_0+t\cdot B$, $y=y_0-t\cdot A$ za neki ceo broj t. Neka je $Au+Bv=C$ za neke cele brojeve u i v. Tada je

$$(Au+Bv)-(Ax_0+By_0)=C-C=0$$

odakle je

$$B(v-y_0)=-A(u-x_0).$$

Odatle dobijamo da $B|A(u-x_0)$ pa kako je $NZD(A,B)=1$, sledi da $B|(u-x_0)$. To znači da postoji ceo broj t takav da je $u-x_0=Bt$, odnosno, $u=x_0+Bt$. Sada je

$$B(v-y_0)=-ABt,$$

odakle je nakon sređivanja $v=y_0-At$. Time je dokaz završen. □

Dakle, da bismo odredili sva celobrojna rešenja jednačine $ax+by=c$, potrebno je naći $d=NZD(a,b)$, i potom za $A=a/d$, $B=b/d$ i $C=c/d$ proširenim Euklidovim algoritmom naći cele brojeve α i β takve da je $A\alpha+B\beta=1$. Tada je $x_0=C\alpha$, $y_0=C\beta$ jedno rešenje jednačine $ax+by=c$, pa sva rešenja ove jednačine imaju oblik $x=x_0+Bt$, $y=y_0-At$.

Zadaci.

5.42. Napisati Paskal program koji od korisnika učitava cele brojeve a, b i c od kojih nijedan nije nula, i utvrđuje da li Diofantova jednačina $ax+by=c$ ima rešenja. Ako ima, program treba da odredi i ispiše opšti oblih rešenja te jednačine.

5.43. Podsetimo se da je za generisanje ključa u RSA kriptosistemu (odeljak 5.9) pre svega potrebno na slučajan način odabrati dva prosta broja p i q, $p\neq q$, onda za $n=pq$ i $s=(p-1)(q-1)$ odabrati broj e takav da je $1<e<s$ i $NZD(e,s)=1$, i na kraju odrediti jednoznačno određen broj d takav da je $1<d<s$ i $(e\cdot d)\bmod s=1$. Ako smo odabrali p, q i e na opisani način, broj d se može dobiti rešavanjem Diofantove jednačine $sx+ey=1$ (koja ima rešenje zato što je $NZD(e,s)=1$), i nalaženjem onog rešenja y koje zadovoljava uslov $1<y<s$. Ta vrednost za y je traženi broj d.

Napisati Paskal program koji od korisnika učitava cele brojeve e i s, i potom određuje i štampa broj d koji predstavlja privatni ključ RSA kriptosistema za odabrane parametre.

5.11 Rezime

⟨*Tip*⟩ ≡ `real` | `integer` | `longint` | `boolean` ... (ima ih još mnogo!)

⟨*Naredba*⟩ ≡ ⟨*Naredba dodele*⟩ | ⟨*If-naredba*⟩ | ⟨*Case-naredba*⟩
⟨*For-naredba*⟩ | ⟨*While-naredba*⟩ | ⟨*Repeat-naredba*⟩
| ... (ima ih još mnogo!)

⟨*If-naredba*⟩ ≡ `if` ⟨*uslov*⟩ `then`
⟨*Blok ili jedna naredba*⟩
[`else`
⟨*Blok ili jedna naredba*⟩]

⟨*Case-naredba*⟩ ≡ `case` ⟨*izraz*⟩ `of`
⟨*spisak*$_1$⟩ `:` ⟨*Blok ili tačno jedna naredba*⟩
⟨*spisak*$_2$⟩ `:` ⟨*Blok ili tačno jedna naredba*⟩
⋮
⟨*spisak*$_k$⟩ `:` ⟨*Blok ili tačno jedna naredba*⟩
`end`

⟨*While-naredba*⟩ ≡ `while` ⟨*uslov*⟩ `do`
⟨*Blok ili tačno jedna naredba*⟩

⟨*Repeat-naredba*⟩ ≡ `repeat`
⟨*Naredba*⟩`;`
⋮
⟨*Naredba*⟩
`until` ⟨*uslov*⟩

⟨*uslov*⟩ ≡ ⟨*logički izraz*⟩

⟨*logički izraz*⟩ ≡ ⟨*Izraz*⟩

⟨*Izraz*⟩ ≡ ⟨*Prost izraz*⟩ | ⟨*Prost izraz*⟩ ⟨*Relacioni operator*⟩ ⟨*Prost izraz*⟩

⟨*Prost izraz*⟩ ≡ ⟨*Sabirak sa predznakom*⟩ |
⟨*Sabirak sa predznakom*⟩ ⟨*Aditivni operator*⟩ ⟨*Sabirak*⟩
⟨*Aditivni operator*⟩ … ⟨*Aditivni operator*⟩ ⟨*Sabirak*⟩

⟨*Sabirak sa predznakom*⟩ ≡ ⟨*Sabirak*⟩ | `-` ⟨*Sabirak*⟩

⟨*Sabirak*⟩ ≡ ⟨*Činilac*⟩ |
⟨*Činilac*⟩ ⟨*Multiplikativni operator*⟩ ⟨*Činilac*⟩
⟨*Multiplikativni operator*⟩ … ⟨*Multiplikativni operator*⟩ ⟨*Činilac*⟩

⟨*Činilac*⟩ ≡ ⟨*Neoznačeni broj*⟩ | ⟨*Ime konstante*⟩ | ⟨*Promenljiva*⟩ |
`not` ⟨*Činilac*⟩ | `(` ⟨*Izraz*⟩ `)`

⟨*Relacioni operator*⟩ ≡ `=` | `<>` | `<` | `<=` | `>` | `>=`

⟨*Aditivni operator*⟩ ≡ `+` | `-` | `or`

⟨*Multiplikativni operator*⟩ ≡ `*` | `/` | `div` | `mod` | `and`

Glava 6

Tip “char”, “case” kontrolna struktura

Iako su nastali kao mašine za obradu numeričkih podataka, računari su doživeli vrhunac svog uspeha kada se shvatilo da oni mogu da obrađuju i *simboličke* podatke. Obrada simboličkih podataka danas predstavlja dominantan način upotrebe računara. U ovoj glavi ćemo upoznati prvi simbolički tip podataka koji se javlja u kursu pred vama – tipom “char” koji omogućuje rad sa pojedinačnim simbolima. Osim toga, upoznaćemo i “case” naredba koja se koristi kada treba proveriti mnogo jednostavnih uslova oblika ⟨*izraz*⟩=⟨*konstantna vrednost*⟩ jedan za drugim. U tom slučaju je “case” naredba znatno efikasnija od višestruke “if” naredbe.

6.1 Tip “char”

Tip podataka `char` opisuje promenljive u koje može da se smesti tačno jedno slovo. Ime potiče od engleske reči *character* koja znači *karakter* (skup mentalnih i moralnih osobina neke osobe), ali i *simbol* (npr. *Chinese characters*, što znači *kineski simboli*, a ne *karakteri Kineza*).

Na primer, kao vrednost promenljive tipa `char` može da se pojavi bilo koji od simbola `'a' 'b' 'c'`... `'A' 'B' 'C'`... `'0' '1' '2'`... `'#' '$' '%' '+' '-' '*' '(' ']' '{' '!' '?' '_' ';' ':' ',' '"' '‘' ''''`.

Primetimo da odgovarajući simbol mora biti “ograđen” apostrofima, kao i to da se simbol `'` (apostrof) piše kao `''''`.

☞ *Simbol ' (apostrof) se zapisuje kao niz od dva apostrofa, tako da naredba* `writeln('Don''t panic!');` *ispisuje* `Don't panic!`

Konstante i promenljive tipa `char` se deklarišu na uobičajeni način, kako se vidi u primeru pored. Prilikom dodele vrednosti promenljivoj takođe nema iznenađenja, tako da program `Primer1` ispisuje `3$` zato što promenljiva `d` nakon dodele `d := c` ima vrednost `'3'`, a promenljiva `money` ima vrednost `'$'`.

```
program Primer1;
const
  DollarSign = '$';
var
  c, d : char;
  money : char;
begin
  c := '3';
  d := '@';
  money := DollarSign;
  d := c;
  writeln(d, money)
end.
```

Apostrofi oko simbola koji predstavljaju vrednosti tipa `char` se pišu samo u programskom kodu, a prilikom ispisa se ne pojavljuju.

☞ *Prilikom ispisa se apostrofi ne pojavljuju. Ispisuju se samo simboli!*

Smisao apostrofa je u tome da naprave razliku između imena promenljive i vrednosti tipa `char` kako to pokazuje `Primer2`. Tu se promenljivoj `d` dodeljuje *simbol* `c`, a promenljivoj `e` vrednost *promenljive* `c`.

U tabeli pored prikazan je rad programa `Primer2` naredbu po naredbu. Na početku vrednosti promenljivih nisu određene. Posle naredbi `c := '@'` i `d := 'c'` promenljiva `c` ima vrednost `'@'`, a promenljiva `d` vrednost `'c'`. Naredba `e := c` kopira vrednost *promenljive* `c`, tako da posle ove naredbe promenljiva `e` ima vrednost `'@'`. Dakle, program `Primer2` na kraju ispisuje `c@` (bez apostrofa!).

```
program Primer2;
var
  c, d, e : char;
begin
  c := '@';
  d := 'c';
  e := c;
  writeln(d, e)
end.
```

Naredba	Promenljiva		
	c	d	e
	?	?	?
`c := '@';`	`'@'`	?	?
`d := 'c';`	`'@'`	`'c'`	?
`e := c;`	`'@'`	`'c'`	`'@'`

Naravno, moramo voditi računa da se promenljivoj tipa `char` ne može dodeliti ni broj ni logička vrednost! *Ovako nešto ne sme:*

```
program NeMozeOvako;
var
  c : char;
begin
  c := 13;        ← compilation error!
  c := true;      ← compilation error!
end.
```

Ukoliko pokušate da promenljivoj tipa `char` dodelite nešto što nije slovo ili slovna promenljiva prevodilac će prijaviti grešku. Slično, ukoliko pokušamo numeričkoj ili logičkoj promenljivoj da dodelimo vrednost koja je tipa `char` prevodilac će prijaviti grešku.

6.2 ASCII kod

Svakom simbolu je dodeljen neki broj, njegov *kod*. Računar u suštini ne pamti simbole kao simbole, već pamti i radi sa odgovarajućim brojevima – kodovima simbola. Kodiranje simbola je standardizovano još sedamdesetih godina ovog veka sistemom propisa koji se zove ASCII (*American Standard Code for Information Interchange*) koji je još uvek u upotrebi. Sledeća tabela sadrži ASCII kodove onih simbola koji se mogu odštampati (*printable symbols*):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Kod odgovarajućeg simbola se dobija tako što se saberu brojevi vrste i kolone kojoj pripada. Na primer, kod simbola `A` je 65, a simbola `a` je 97. Kodovi 0–31, kao i kod 127 su rezervisani za specijalne namene (markiranje kraja reda, kraja datoteke itd).

Za pamćenje promenljivih tipa `char` se obično koristi jedan bajt. Stoga nam na raspolaganju stoje i kodovi 128–255. ASCII standardom nije predviđen sadržaj tog dela tabele, tako da razni proizvođači računara tim kodovima dodeljuju razne simbole. Često su to neki grafički simboli.

Kako su računari postali rasprostranjeni, primećeno je da je ASCII kod suviše anglocentričan (prilagođen je samo engleskom govornom području). Danas se radi na uvođenju novog standarda, UNICODE, koji bi podržavao simbole svih evropskih jezika, kao i simbole nekih azijskih (bar kineski i japanski set simbola). Međutim, implementacije programskog jezika Paskal sa kojima ćemo mi raditi još uvek podržavaju samo ASCII kod.

6.3 Operacije sa promenljivim tipa "char"

Postoje dve standardne funkcije koje omogućuju konverziju simbola u njihove kodove i obrnuto. Funkcija `ord` uzima jedno slovo i vraća njegov kod, dok funkcija `chr` uzima broj i vraća simbol čiji je to kod. Na primer, `ord('b') = 98`, dok je `chr(67) = 'C'`. Funkcije `chr` i `ord` su jedna drugoj inverzne. To znači da je `ord(chr(`n`)) =` n i `chr(ord(`c`)) =` c za svaki broj $n \in \{0,1,\ldots,255\}$ i svaki simbol c.

Na skupu svih simbola je uveden poredak ovako:

$$c_1 < c_2 \text{ ako i samo ako je } \texttt{ord}(c_1) < \texttt{ord}(c_2).$$

Tako je `'a' < 'b' < 'z'` i `'A' < 'Z' < 'a'`.

Postoje još dve funkcije za manipulaciju simbolima: `succ` (*successor* = naslednik) i `pred` (*predecessor* = prethodnik). Na primer: `succ('b') = 'c'`, dok je, `pred('Z') = 'Y'`. Primetimo da je

`succ(`c`) = chr(ord(`c`) + 1)`, i `pred(`c`) = chr(ord(`c`) - 1)`.

Pri tome vrednosti `succ(chr(255))` i `pred(chr(0))` nisu definisane.

Zadaci.

6.1. Prokomentarisati sledeće Paskal programe:

```
program CpostajeC1;
var
 c : char;
begin
 c := 'c';
end.
```

```
program CpostajeC2;
var
 c : char;
begin
 c := c;
end.
```

6.2. Dekodirati:

slovo:				
ASCII kod:	90	109	97	106

6.3. Dekodirati: 83 117 112 101 114 32 70 111 114 99 101

6.4. Šta ispisuje sledeća naredba:

```
writeln(succ('H'), succ('A'), succ('L'));
```

6.5. Izračunati:

(*a*) `ord(succ('A')) - 10 =` ☐

(*b*) `chr(ord(pred('0')) + 15) =` ☐

(*c*) `succ(succ('A')) >= pred(chr(66)) =` ☐

6.6. Sa deklaracijom `var c : char;` na snazi označiti korektne naredbe:

☐ `c := 'a';`	☐ `c := "';`	☐ `c := true;`
☐ `c := 'c';`	☐ `c := "";`	☐ `c := pred('a');`
☐ `c := "a";`	☐ `c := 65;`	☐ `c := succ(c);`
☐ `c := ";`	☐ `c := ord(65);`	☐ `c := pred(c);`
☐ `c := '"';`	☐ `c := chr(65);`	☐ `c := 12.1;`

6.7. Napisati Paskal program koji od korisnika učitava brojeve p i q takve da je $32 \leqslant p \leqslant q \leqslant 126$ i štampa simbole čiji kod pripada intervalu $[p,q]$, kao i odgovarajuće kodove. Na primer, za brojeve 65 i 68 izlaz je dat pored.

```
65  A
66  B
67  C
68  D
```

6.8. (Hi-Lo Game) Hi-Lo Game je igra u kojoj računar pogađa broj koga je zamislio korisnik. Korisnik zamisli broj između 1 i 10000, a računar pokušava da pogodi o kom se boju radi postavljanjem pitanja korisniku. Pitanja su oblika: "Da li je zamišljeni broj veći od . . . ?" Na svako pitanje korisnik odgovara sa "da" ili "ne". Napisati Paskal program koji igra Hi-Lo igru, i pogađa broj koga je zamislio korisnik sa najviše 14 pitanja. (Uputstvo: koristiti binarno pretraživanje!) Na primer,

```
Zamislite neki broj izmedju 1 i 10000.
Da li je broj veci od 5000 (d/n) -> n
Da li je broj veci od 2500 (d/n) -> n
Da li je broj veci od 1250 (d/n) -> d
Da li je broj veci od 1875 (d/n) -> d
Da li je broj veci od 2187 (d/n) -> n
```

```
Da li je broj veci od 2031 (d/n) -> n
Da li je broj veci od 1953 (d/n) -> d
Da li je broj veci od 1992 (d/n) -> d
Da li je broj veci od 2011 (d/n) -> n
Da li je broj veci od 2001 (d/n) -> n
Da li je broj veci od 1996 (d/n) -> d
Da li je broj veci od 1998 (d/n) -> d
Da li je broj veci od 1999 (d/n) -> d
Da li je broj veci od 2000 (d/n) -> n
Sada znam! Zamislili ste broj 2000.
```

6.4 “Case” kontrolna struktura

Pogledajmo program koji od korisnika učitava dva cela broja koji predstavljaju mesec i godinu, i potom određuje i štampa broj dana u tom mesecu:

```
program BrojDanaUMesecu;
var
  m, g, p : integer;
begin
  writeln('Mesec i godina -> '); readln(m, g);

  { ispitamo da li je g prestupna godina }
  if (g mod 400 = 0) or (g mod 100 <> 0) and (g mod 4 = 0) then
    p := 1
  else
    p := 0;
  { ispisujemo broj dana u mesecu m }
  if      m =  1 then writeln(31)
  else if m =  2 then writeln(28 + p)
  else if m =  3 then writeln(31)
  else if m =  4 then writeln(30)
  else if m =  5 then writeln(31)
  else if m =  6 then writeln(30)
  else if m =  7 then writeln(31)
  else if m =  8 then writeln(31)
  else if m =  9 then writeln(30)
  else if m = 10 then writeln(31)
  else if m = 11 then writeln(30)
  else if m = 12 then writeln(31)
  else writeln('Pogresan mesec')
end.
```

Kada treba proveriti mnogo jednostavnih uslova oblika

```
if ⟨izraz1⟩=⟨konstantna vrednost1⟩ then ...
else if ⟨izraz2⟩=⟨konstantna vrednost2⟩ then ...
  ⋮
else if ⟨izrazk⟩=⟨konstantna vrednostk⟩ then ...
```

jedan za drugim kao što je ovde slučaj, višestruki "if" može biti neefikasno i nepregledno rešenje. Zato Paskal nudi jedan drugi vid grananja: "case" kontrolnu strukturu.

"Case" kontrolna struktura ima oblik koji je naveden pored, gde je svaki $\boxed{P_i}$ ili tačno jedna naredba, ili blok, a $\langle spisak_i \rangle$ je niz pojedinačnih vrednosti ili intervala oblika `p .. q`.

"Case" kontrolna struktura radi tako što računar prvo izračuna vrednost izraza $\langle izraz \rangle$ i traži spisak u kome se ta vrednost nalazi. Ako se vrednost nalazi na spisku i, tada se izvrši $\boxed{P_i}$.

```
case ⟨izraz⟩ of
    ⟨spisak1⟩ : [ P1 ];
    ⟨spisak2⟩ : [ P2 ];
    ⋮
    ⟨spisakk−1⟩ : [ Pk−1 ];
    ⟨spisakk⟩ : [ Pk ]
end
```

Koristeći "case" kontrolnu strukturu, program koji ispisuje broj dana u mesecu može mnogo preglednije da se napiše ovako:

```
program BrojDanaUMesecu;
var
  m, g, p : integer;
begin
  writeln('Mesec i godina -> '); readln(m, g);
  { ispitamo da li je g prestupna godina }
  if (g mod 400 = 0) or (g mod 100 <> 0) and (g mod 4 = 0) then
    p := 1
  else
    p := 0;
```

```
  { ispisujemo broj dana u mesecu m }
  if (1 <= m) and (m <= 12) then
    case m of
       1: writeln(31);
       2: writeln(28 + p);
       3: writeln(31);
       4: writeln(30);
       5: writeln(31);
       6: writeln(30);
       7: writeln(31);
       8: writeln(31);
       9: writeln(30);
      10: writeln(31);
      11: writeln(30);
      12: writeln(31)
    end
  else writeln('Pogresan mesec')
end.
```

ili još kraće, ovako:

```
program BrojDanaUMesecu;
var
  m, g, p : integer;
begin
  writeln('Mesec i godina -> '); readln(m, g);

  { ispitamo da li je g prestupna godina }
  if (g mod 400 = 0) or (g mod 100 <> 0) and (g mod 4 = 0) then
    p := 1
  else
    p := 0;

  { ispisujemo broj dana u mesecu m }
  if (1 <= m) and (m <= 12) then
    case m of
       1, 3, 5, 7, 8, 10, 12: writeln(31);
       2: writeln(28 + p);
       4, 6, 9, 11: writeln(30)
    end
  else writeln('Pogresan mesec')
end.
```

Postoji nekoliko pravila o kojima moramo voditi računa kada koristimo "case":

- spiskovi moraju da se sastoje isključivo od *konstantnih* vrednosti; promenljive i izrazi *ne smeju* da se pojave u spisku;
- ukoliko se vrednost izraza ne nalazi ni na jednom od ponuđenih spiskova, računar će prijaviti grešku;
- jedna ista konstanta ne sme da se pojavi u dva različita spiska, jer u tom slučaju računar ne bi znao koju naredbu/blok da izvrši.

Primer. Napisati Paskal program koji učitava redni broj meseca (broj između 1 i 12) i štampa godišnje doba kojem pripada taj mesec. Jednostavnosti radi, uzimamo da su decembar, januar i februar zimski meseci, mart, april i maj prolećni, jun, jul i avgust letnji, a septembar, oktobar i novembar jesenji meseci.

```
program GodDoba;
var
  m : integer;
begin
  writeln('Mesec (1..12) -> ');
  readln(m);
  if (1 <= m) and (m <= 12) then
    case m of
      12, 1, 2 : writeln('zima');
      3..5     : writeln('prolece');
      6..8     : writeln('leto');
      9..11    : writeln('jesen')
    end
  else writeln('Pogresan mesec')
end.
```

☞ *Oznaka* `3..5` *znači "od 3 do 5" i ekvivalentna je sa* `3, 4, 5`.

Primer. *Encyclopaedia Hysterica* ima 6 tomova: prvi tom obuhvata slova od A do F i slovo X, drugi slovo G, treći slova od H do N i slovo Y, četvrti slova od O do S, peti slova T i U, a šesti slova V, W i Z (pri čemu se iz nekih nikada razjašnjenih razloga koristi engleska abeceda). Napisati program koji od korisnika učitava slovo i ispisuje redni broj toma koji sadrži to slovo.

```
program EncyclopaediaHysterica;
var
  c : char;
begin
  writeln('Slovo -> '); readln(c);
  if ('A' <= c) and (c <= 'Z') then
    case c of
      'A'..'F', 'X' : writeln('I');
      'G'           : writeln('II');
      'H'..'N', 'Y' : writeln('III');
      'O'..'S'      : writeln('IV');
      'T', 'U'      : writeln('V');
      'V', 'W', 'Z' : writeln('VI')
    end
  else writeln('Pogresno slovo')
end.
```

Videli smo da se “case” kontrolna struktura koristi kada treba proveriti mnogo jednostavnih uslova oblika ⟨*izraz*⟩=⟨*konstantna vrednost*⟩ jedan za drugim. Međutim, ako su uslovi “komplikovani”, ili ako su vrednosti koje treba proveriti “jako razmaknute”, onda ipak treba koristiti “if”naredbu. Pored je dat primer kada *ne treba* koristiti “case” zato što su vrednosti “jako razmaknute”.

```
readln(n);
case n of
      1 : writeln('jedan');
     10 : writeln('deset');
    100 : writeln('sto');
   1000 : writeln('hiljadu');
  10000 : writeln('deset hiljada')
end
```

Takođe, ako jedan spisak ima “jako mnogo vrednosti”, onda postoji mogućnost da se tokom kompilacije dobije neefikasan kod. Tako, ni u primeru pored se *ne preporučuje* upotreba “case” kontrolne strukture.

```
readln(n);
case n of
 1 .. 32767   : writeln('pozitivan');
 0            : writeln('nula');
 -32768 .. -1 : writeln('negativan')
end
```

U oba ova fragmenta prirodnije je koristiti “if”.

Zadaci.

6.9. (*a*) Šta radi program `Zad1`?

(*b*) Šta radi program `Zad2` kada korisnik unese slovo ’M’?

(*c*) Napisati program `Zad2` bez upotrebe “case” kontrolne strukture.

```
program Zad1;
var
  n : integer;
begin
  readln(n);
  case n mod 3 of
    0    : writeln('jeste');
    1, 2 : writeln('nije')
  end
end.
```

```
program Zad2;
var
  c : char;
begin
  readln(c);
  case c of
    'A', 'a' : writeln('1');
    'E', 'e' : writeln('5');
    'I', 'i' : writeln('9');
    'O', 'o' : writeln('15');
    'U', 'u' : writeln('21')
  end
end.
```

6.10. Šta nije dobro u sledećim programima?

```
program Zad3;
var
  n : integer;
begin
  readln(n);
  case n + 2 of
    2*n   : writeln('n = 2');
    n     : writeln('nemoguce');
    1..99 : writeln('nije trocifren')
  end
end.
```

```
program Zad4;
var
  c : char;
begin
  readln(c);
  case c of
    'A' .. 'Z' : writeln('veliko slovo');
    'a' .. 'z' : writeln('malo slovo slovo');
    'A', 'a', 'E', 'e', 'I', 'i', 'O', 'o', 'U', 'u' :
                 writeln('samoglasnik')
  end
end.
```

6.11. Sledeći programski fragment napisati koristeći "case" naredbu:

```
if k = 0 then
  begin
    r := r + 1;
    writeln(r)
  end
else if k = 1 then
  begin
    s := s + 1;
    writeln(s)
  end
else if (k = 2) or (k = 3) or (k = 4) then
  begin
    t := t + 1;
    writeln(t)
  end
```

6.12. Utvrditi vrednosti promenljivih `p` i `d` nakon izvršenja sledećeg programskog fragmenta ako je korisnik za vrednost promenljive `k` uneo

(a) 6; (b) 235; (c) 71; (d) 100.

```
readln(k);
p := 1; d := 1;
case k mod 10 of
  2, 3, 5, 7 : d := k;
  1          : ;
  4, 8       : begin p := 0; d := 2 end;
  9, 6       : begin p := 0; d := 3 end
end
```

6.13. Napisati program koji učitava tri cela broja koji predstavljaju neki datum, a potom ispisuje taj datum tako da mesec bude ispisan rečima. Na primer za 3. 11. 2000. program treba da ispiše `3. novembar 2000`.

6.14. Napisati program koji učitava tri cela broja koji predstavljaju neki datum, a potom ispisuje taj datum prema engleskom standardu, ali tako da mesec bude ispisan rečima. Na primer za 3. 11. 2000. program treba da ispiše `November 3rd, 2000`.

6.15. Napisati program koji konvertuje dati ceo broj iz skupa $\{1,2,\ldots,3999\}$ u rimski zapis. Na primer:

```
Unesite broj : 1937
Rimski zapis broja: MCMXXXVII
```

6.16. (Večiti kalendar) Napisati Paskal program koji za dati datum predstavljen pomoću tri celobrojne promenljive d, m, g utvrđuje koji je to dan u nedelji. Pretpostavlja se da je datum korektan i da je $g \geqslant 1582$. Pri tome koristiti sledeću formulu:

$$p = \big(d + k + g + g \texttt{ div } 4 - 2 \cdot (v \texttt{ mod } 4) - f\big) \texttt{ mod } 7,$$

gde je

- $v = g$ `div` 100 broj vekova u godini;
- $f = 1$ ako je g prestupna godina i $m \in \{1,2\}$, u ostalim slučajevima je $f = 0$;
- k je broj koji se dobija od broja m prema sledećoj tabeli:

m	8	2, 3, 11	6	9, 12	4, 7	1, 10	5
k	0	1	2	3	4	5	6

Broj p tumačimo ovako: ako je $p = 0$ taj dan je nedelja, ako je $p = 1$ taj dan je ponedeljak, ..., ako je $p = 6$ taj dan je subota.

6.5 Rezime

Tip podataka `char` opisuje promenljive u koje može da se smesti tačno jedan ASCII simbol. Promenljiva tipa `char` se deklariše ovako:

```
var
  ⟨ime⟩ : char;
```

Dve standardne funkcije omogućuju konverziju simbola u njihove kodove i obrnuto. Funkcija `ord` uzima jedno slovo i vraća njegov kod, dok funkcija `chr` uzima broj i vraća simbol čiji je to kod. Pri tome je `ord(chr(`n`)) = `n i `chr(ord(`c`)) = `c za svaki broj $n \in \{0,1,\ldots,255\}$ i svaki simbol c. Na skupu svih simbola je uveden poredak ovako:

$$c_1 < c_2 \text{ ako i samo ako je } \texttt{ord(}c_1\texttt{)<ord(}c_2\texttt{)}.$$

Funkcija `succ` vraća naredni karakter u ASCII tabeli, dok funkcija `pred` vraća prethodni karakter. Drugim rečima,

`succ(`c`) = chr(ord(`c`) + 1)`, i `pred(`c`) = chr(ord(`c`) - 1)`.

Pri tome vrednosti `succ(chr(255))` i `pred(chr(0))` nisu definisane.

“Case” kontrolna struktura ima sledeći oblik:

⟨*Case-naredba*⟩≡ `case` ⟨*izraz*⟩ `of`

⟨*spisak*$_1$⟩ `:` $\boxed{P_1}$;

⟨*spisak*$_2$⟩ `:` $\boxed{P_2}$;

⋮

⟨*spisak*$_{k-1}$⟩ `:` $\boxed{P_{k-1}}$;

⟨*spisak*$_k$⟩ `:` $\boxed{P_k}$

`end`

gde je svaki $\boxed{P_i}$ ili tačno jedna naredba, ili blok, a ⟨*spisak*$_i$⟩ je niz pojedinačnih vrednosti ili intervala oblika `p .. q`. Pri tome,

- spiskovi moraju da se sastoje isključivo od *konstantnih* vrednosti; promenljive i izrazi *ne smeju* da se pojave u spisku;
- ukoliko se vrednost izraza ne nalazi ni na jednom od ponuđenih spiskova, računar će prijaviti grešku;
- jedna ista konstanta ne sme da se pojavi u dva različita spiska, jer u tom slučaju računar ne bi znao koju naredbu/blok da izvrši.

Glava 7

Nizovi karaktera i tekstualne datoteke

U ovom poglavlju ćemo se baviti tehnikama za manipulaciju teksta. Prvo ćemo naučiti da koristimo stringove, koji predstavljaju specijalne nizove karaktera. Prema standardu, programski jezik Paskal ne podržava stringove direktno. Većina implementacija Pascala, međutim, podržava stringove kao osnovni tip podataka. Zato stringovi imaju poseban status u programskom jeziku Paskal.

Kako stringovi imaju relativno ograničenu veličinu (najviše 255 karaktera), videćemo i dve strategije za manipulisanje većim tekstovima. Nakon stringova ćemo videti kako se u programskom jeziku Paskal može upravljati tekstualnim datotekama, što su tekstovi pohranjeni na disku računara čina čija veličina je ograničena samo veličinom diska, a na kraju ćemo pokazati kako možemo da koristimo nizove karaktera u radnoj memoriji.

7.1 Stringovi

☞ *Sve što je navedeno u ovom i narednom odeljku* **nije deo standarda** *i može se drastično razlikovati od implementacije do implementacije! Program koji koristi ove osobine nije prenosiv i može se desiti da ispravno radi na jednoj platformi, a da se ne može prevesti ili da nekorektno radi na drugoj!*

String je reč književnog engleskog porekla čije osnovno značenje je “objekti iste vrste koji su nanizani na nit”. U programskom jeziku Paskal string je niz karaktera. Na primer, ovo su stringovi:

```
'Zdravo! Kako si?'
'Biti il'' ne biti'
':-)'
''
```

Vidimo da se stringovi pišu tako što se između znakova apostrofa navede proizvoljan niz printabilnih karaktera. Poslednji primer nam pokazuje kako izgleda *prazan string*, tj. string koji nema nijedno slovo, dok nam drugi primer pokazuje kako da zapišemo string u kome se javlja apostrof kao jedan od karaktera. Pošto se apostrofi koriste kao graničnici za sadržaj stringa, apostrof koji je karakter u stringu zapisujemo koristeći dva uzastopna apostrofa.

Broj karaktera u stringu se zove *dužina stringa*. Dužine stringova iz prethodnog primera su, redom, 16, 16, 3 i 0.

Promenljiva tipa string se može deklarisati kako je to pokazano pored.

```
var
    s : string;
```

Skoro sve implementacije programskog jezika Paskal podržavaju stringove kao "standardni" tip podataka zato što je tada rad sa njima udobniji. String `s` se interno interpretira kao pakovani niz karaktera čija maksimalna dužina zavisi od implementacije, a najčešće je to 255.

Informaciju o dužini stringa `s` daje nam ugrađena funkcija `length` koja vraća rezultat tipa `integer`. Pri tome, funkcija `length` vraća *realnu* dužinu stringa, a ne veličinu prostora koji je za string rezervisan.

Program `MaliPrimer1` učitava string od korisnika i ispisuje njegovu dužinu. Na primer, ako kao vrednost promenljive `s` korisnik unese `Programiranje`, program će ispisati 13.

```
program MaliPrimer1;
var
  s : string;
begin
  readln(s);
  writeln(length(s))
end.
```

Stringovi se učitavaju standardnom naredbom `read`/`readln` i ispisuju standardnom naredbom `write`/`writeln`, kao i sve ostale promenljive osnovnog tipa. Stringu se može dodeliti neka vrednost naredbom dodele kao u `MalomPrimeru2`.

```
program MaliPrimer2;
var
  s, t : string;
begin
  readln(s);
  t := 'Hello, ';
  writeln(t, s)
end.
```

Literali tipa `string` se navode između apostrofa i *moraju počinjati i završavati u istom redu.*

Pošto je string zapravo (pakovani) niz karaktera, moguće je pristupiti jednom elementu stringa i promeniti ga, kako je to pokazano u `MalomPrimeru3`.

```
program MaliPrimer3;
var
  s : string;
begin
  readln(s);
  if length(s) > 3 then
    s[3] := 'x';
  writeln(s)
end.
```

Tip `string` poseduje sledeće operatore koji su "ugrađeni" u jezik, gde su `s` i `t` promenljive tipa `string`, a `c` promenljiva tipa `char`:

Operacija	Značenje (rezultat je uvek tipa string)
`s + t`	nadovezivanje (konkatenacija)
`s + c`	dodavanje karaktera `c` na kraj stringa `s`
`c + s`	dodavanje karaktera `c` na početak stringa `s`

Stringovi mogu i da se upoređuju. Za poređenje stringova u programskom jeziku Paskal koristi se sistem koji se zove *leksikografski poredak*. To je poredak koji se može videti u rečnicima (odatle ime), a definisan je na sledeći način. Neka su `s` i `t` različiti stringovi.

- Ako je string `s` prefiks stringa `t` onda je $\texttt{s} < \texttt{t}$.
- Ako string `s` nije prefiks stringa `t`, onda krenemo sleva na desno i nađemo prvo mesto na kome se stringovi razlikuju. Neka je to pozicija i. Sada, ako je $\texttt{s}[i] < \texttt{t}[i]$, onda je $\texttt{s} < \texttt{t}$. U suprotnom je $\texttt{s} > \texttt{t}$.

Na primer,

`'bar'` < `'bara'` zato što je string `'bar'` prefiks stringa `'bara'`
`'bara'` < `'bas'` zato što se stringovi razlikuju na 3. mestu i `'r'` < `'s'`
`'Avala'` < `'ala'` zato što se stringovi razlikuju na 1. mestu i `'A'` < `'a'`

Operator = proverava da li su dva stringa jednaki, dok <> proverava da li su stringovi različiti. Operatori <, <=, > i >= proveravaju da li je prvi string manji, manji ili jednak, veći, veći ili jednak od drugog stringa u leksikografskom poretku.

Operacija	Značenje (rezultat je uvek tipa boolean)
`s = t`	stringovi su jednaki
`s <> t`	stringovi su različiti
`s < t`	`s` je ispred `t` u leksikografskom poretku
`s <= t`	`s` je jednako sa `t` ili ispred `t` u leksikografskom poretku
`s > t`	`s` je iza `t` u leksikografskom poretku
`s >= t`	`s` je jednako sa `t` ili je iza `t` u leksikografskom poretku

Na primer, neka su `s` i `t` stringovi čija vrednost je `s = 'bar'` i `t = 'bara'`, a `c` promenljiva tipa `char` čija vrednost je `c = 'i'`. Tada:

Izraz	Vrednost	Tip
`s + t`	`'barbara'`	`string`
`t + s`	`'barabar'`	`string`
`s + c`	`'bari'`	`string`
`c + s`	`'ibar'`	`string`
`s = t`	`false`	`boolean`
`s < t`	`true`	`boolean`
`t[3]`	`'r'`	`char`

Osim ovih, implementacije Paskala nude još obilje operacija koje nisu "ugrađene" u jezik, već se mogu naći u pratećim bibliotekama.

Primer. Napisati Paskal program koji od korisnika učitava neki string, i onda ga štampa u obrnutom redosledu. Na primer, ako korisnik unese

```
Napisati Paskal program
```

program će ispisati

```
margorp laksaP itasipaN
```

```
program Obrnuto;
var
  s : string;
  i : integer;
begin
  write('Tekst -> ');
  readln(s);
  write('Obrnuto -> ');
  for i := length(s) downto 1 do
    write(s[i]);
  writeln
end.
```

Primer. Napisati Paskal program koji utvrđuje koliko ima razmaka u učitanom stringu.

```
program Razmaci;
var
  n, i : integer;
  s : string;
begin
  readln(s);
  n := 0;
  for i := 1 to length(s) do
    if s[i] = ' ' then inc(n);
  writeln('Broj razmaka je ', n)
end.
```

Primer. Napisati Paskal program koji utvrđuje da li je u učitanom stringu broj otvorenih zagrada `'('` jednak broju zatvorenih zagrada `')'`.

```
program Zagrade;
var
  n, i : integer;
  s : string;
begin
  readln(s);
  n := 0;
  for i := 1 to length(s) do
    begin
      if s[i] = '(' then inc(n);
      if s[i] = ')' then dec(n)
    end;
  if n = 0 then writeln('jeste')
  else writeln('nije')
end.
```

Primer. Napisati Paskal program koji učitava string `s` i onda formira i ispisuje novi string u kome su sva mala slova stringa `s` konvertovana u velika. Na primer, string `'Govori tiho!'` će biti konvertovan u string `'GOVORI TIHO!'`

```
program VelikaSlova;
var
  i : integer;
  s : string;
begin
  readln(s);
  for i := 1 to length(s) do
    if ('a' <= s[i]) and (s[i] <= 'z') then
      s[i] := chr(ord(s[i]) - ord('a') + ord('A'));
  writeln(s)
end.
```

Primer. Napisati Paskal program koji učitava stringa `s`, iz njega izbacuje sva pojavljivanja karaktera `' '` (razmak) i potom ispisuje tako skraćeni string. Na primer, ako je je program učitao string `'Idi mi  dodji mi'` ispisaće `Idimidodjimi`.

```
program IzbaciRazmake;
var
  i : integer;
  s, t : string;
begin
  readln(s);
  t := ''; {prazan string}
  for i := 1 to length(s) do
    if s[i] <> ' ' {razmak} then
      t := t + s[i];
  writeln(t)
end.
```

☞ *Prethodni primer nam pokazuje kako se grade stringovi: krenemo od praznog stringa, pa na njega nadovezujemo karaktere ili nove stringove.*

7.2 Funkcije koje manipulišu stringovima

☞ *Napomenimo još jednom da sve što je navedeno u ovom odeljku* **nije deo standarda** *i može se drastično razlikovati od implementacije do implementacije! Funkcije za transformaciju stringova koje ćemo ovde opisati su karakteristične za Free Pascal i može desiti da u drugim implementacijama Paskala ne postoje, ili da imaju drugačiju upotrebu!*

Stringovi su veoma važni u programiranju i zato postoji veliki broj funkcija koje su namenjene raznim transformacijama stringova. Ovde ćemo prikazati nekoliko najznačajnijih.

Upcase i lowercase. Funkcija `upcase` konvertuje sva mala slova stringa u velika slova, dok funkcija `lowercase` konvertuje sva velika slova stringa u mala slova. Na primer, ako korisnik unese string `Pera`, program pored će ispisati `peraPERA`

```
program ULPrimer;
var
  s, t, u : string;
begin
  readln(s);
  t := lowercase(s);
  u := upcase(s);
  writeln(t + u)
end.
```

Str. Procedura `str` konvertuje dati numerički podatak (podatak tipa `integer`, `longint` ili `real`) u string. Iza numeričkog podatka može da se pojavi i informacija o formatu ispisa, kao kod formatiranog ispisa broja. Na primer, program pored ispisuje

```
2
      3.14
10
```

```
program StrPrimer;
const
  N = 10;
  pi = 3.14159;
var
  s : string;
begin
  Str(N, s);
  writeln(length(s));
  Str(pi : 10 : 2, s);
  writeln(s);
  writeln(length(s))
end.
```

zato što su 2, odnosno, 10 dužine odgovarajućih stringova nakon konverzije, i zato što je prilikom konvertovanja konstante `pi` u string primenjen format `: 10 : 2` koji za dati realan broj rezerviše 10 mesta, od čega su poslednja dva decimalna.

Val. Procedura `val` konvertuje dati string u odgovarajući numerički podatak. Ukoliko je konverzija uspešna agrument `ok` će nakon konverzije sadržati vrednost 0, a ako string ne sadrži zapis numeričkog podatka agrument `ok` će nakon konverzije sadržati celobrojnu vrednost koja nije 0. Na primer, program pored ispisuje

```
3.1400000000000001E+000
4
```

```
program ValPrimer;
var
  x : real;
  n, ok : integer;
begin
  val('3.14', x, ok);
  writeln(x);
  val('125$', n, ok);
  writeln(ok)
end.
```

Primetimo da je drugi podatak 4, što znači da druga konverzija nije bila uspešna, kao što smo i očekivali.

Delete. Procedura `delete(s, i, n)` briše `n` uzastopnih karaktera stringa `s` počev od `i`-tog karaktera. Na primer, ako je `s = 'This is not easy!'` nakon naredbe

```
delete(s, 9, 4);
```

string `s` će imati vrednost `'This is easy!'`

Pos. Funkcija `pos(t, s)` vraća mesto prvog pojavljivanja stringa `t` u stringu `s`. Ukoliko se string `t` ne pojavljuje u stringu `s` funkcija `pos` vraća vrednost 0. Na primer,

```
pos('bara', 'barbara') = 4
pos('bara', 'rhubarb') = 0
```

Kao još jedan primer navodimo program koji u datom stringu zamenjuje svako pojavljivanje simbola \ simbolom /. Ovo je standardna operacija za programe koji treba da rade pod raznim operativnim sistemima. Na primer, ako korisnik unese:

```
c:\users\dm\test.pas
```

program će ispisati:

```
c:/users/dm/test.pas
```

```
program KoseCrte;
var
  s : string;
  i : integer;
begin
  readln(s);
  i := pos('\', s);
  while i > 0 do
    begin
      s[i] := '/';
      i := pos('\', s)
    end;
  writeln(s)
end.
```

Insert. Procedura `insert(t, s, i)` umeće string `t` u string `s` od mesta `i`. Podrazumeva se da se karakteri iz stringa `s` koji se nalaze na mestu `i` i desno pri tome pomeraju udesno. Na primer, ako se nakon naredbe

```
s := 'Programiranje je tesko!';
```

izvrše naredbe

```
insert('ni', s, 15);
writeln(s);
```

računar će ispisati

```
Programiranje nije tesko!
```

Copy. Funkcija `copy(s, i, n)` izdvaja iz stringa `s` segment koji počinje na mestu `i` i ima dužinu `n` i to vraća kao svoju vrednost. Ako je $\texttt{i}+\texttt{n}-1 > \texttt{length(s)}$ funkcija će vratiti kraći segment koji ide od mesta `i` pa do kraja stringa. Na primer,

```
copy('barbara', 3, 4) = 'rbar'
copy('barbara', 5, 5) = 'ara'
```

Primer. Napisati program koji konvertuje temperaturu iskazanu u Farenhajtima, t_F, u Celzijuse, t_C, i obrnuto, imajući u vidu da je:

$$t_F = 1.8 \cdot t_C + 32.$$

Korisnik unosi niz simbola koji predstavlja decimalni zapis broja odmah iza koga se nalazi slovo `C` ili `F`. Ako se niz simbola završava sa `C` znači da je temperatura iskazana u Celzijusima i da je treba konvertovati u Farenhajte, a ako se niz simbola završava sa `F` znači da je temperatura iskazana u Farenhajtima i da je treba konvertovati u Celzijuse. Na primer:

```
Unesite temperaturu:        ili     Unesite temperaturu:
92F                                 12C
92.00F = 33.33C                     12.00C = 53.60F
```

Rešenje.

```
program CelzijusFarenhajt;
var
  s : string;
  t, x : real;
  k, ok : integer;
  jed : char;
begin
  writeln('Unesite temperaturu');
  readln(s);
  k := length(s);
  jed := s[k];
  delete(s, k, 1);
  val(s, t, ok);
  if ok <> 0 then
    writeln('Greska u zapisu broja')
  else if jed = 'C' then
    begin { konverzija iz C u F }
      x := 1.8 * t + 32;
      writeln(t : 6 : 2, 'C = ', x : 6 : 2, 'F')
    end
  else if jed = 'F' then
    begin { konverzija iz F u C }
      x := (t - 32) / 1.8;
      writeln(t : 6 : 2, 'F = ', x : 6 : 2, 'C')
    end
  else
    writeln('Nepoznata jedinica')
end.
```

Primer. Napisati Paskal program koji korisnika učitava algebarski izraz oblika

$$\langle broj \rangle + \langle broj \rangle$$

i računa njegovu vrednost. Ceo izraz je niz karaktera bez razmaka koji se nalazi u jednom redu. Brojevi koji su navedeni su decimalni brojevi.

Rešenje.

```
program Zbir;
var
  s : string;
  k, ok1, ok2 : integer;
  x, y : real;
begin
  write('Unesite izraz -> '); readln(s);
  k := pos('+', s);
  if k = 0 then
    writeln('A gde je znak +?')
  else
    begin
      val(copy(s, 1, k - 1), x, ok1);
      val(copy(s, k+1, length(s) - k), y, ok2);
      if (ok1 = 0) and (ok2 = 0) then
        writeln(x + y : 16 : 6)
      else
        writeln('Greska u zapisu brojeva')
    end
end.
```

Zadaci.

7.1. Napisati Paskal program koji od korisnika učitava string i potom formira i ispisuje novi string koji se dobija dodavanjem jednog razmaka iza svake tačke u stringu `s`.

7.2. Napisati Paskal program koji od korisnika učitava string i potom formira i ispisuje novi string koji se dobija dodavanjem jednog razmaka iza svakog znaka interpunkcije, ako je to potrebno. Na primer,

`'Halo?Da? Ne.Ko je to?'` $\mapsto$ `'Halo? Da? Ne. Ko je to?'`

7.3. Napisati Paskal program koji od korisnika učitava string i potom ispisuje broj reči u tom stringu. (Napomena: reč je neprekidan niz slova.)

7.4. Napisati program koji ispisuje reči unetog teksta u obrnutom poretku. Na primer, ukoliko se unese prethodna rečenica, program treba da ispiše:

`poretku obrnutom u teksta unetog reci ispisuje koji program Napisati`

7.5. Napisati Paskal program koji od korisnika učitava string i potom ispisuje poslednju reč u tom stringu. Na primer, ako je korisnik uneo string `'Biti il' ne biti...'`, program ispisuje reč `biti`

7.6. Napisati Paskal program koji proverava da li su otvorene `'('` i zatvorene `')'` zagrade u unetom stringu balansirane. Na primer

`a((bc)()+)n`	jesu
`[)`	nisu
`(())())`	nisu
`ab)()`	nisu

(Uputstvo: zagrade su balansirane ako i samo ako u svakom početnom segmentu ulazne linije imamo da je broj `'('` zagrada $\geqslant$ od broja `')'` zagrada.)

7.7. Napisati Paskal program koji proverava da li su u unetom stringu balansirane sledeće zagrade: `'('` i `')'`, `'['` i `']'`, `'<'` i `'>'`.

7.8. Napisati Paskal program koji utvrđuje da li je dati *veliki* broj deljiv sa 30. Veliki broj je broj sa proizvoljno mnogo cifara koje se unose u jednoj liniji kao niz karaktera.

Primer 1:
```
Unesite broj:  458273048572304587230924
Nije deljiv sa 30
```

Primer 2:
```
Unesite broj:  303030300000000030000000030
Deljiv je sa 30
```

7.9. Isto kao u prethodnom zadatku, samo što se proverava deljivost sa 4.

7.10. Isto kao u prethodnom zadatku, samo što se proverava deljivost sa 11.

Kriterijum deljivosti sa 11. Neka je s_p suma cifara broja na parnim mestima, a s_n suma cifara tog broja na neparnim mestima. Broj je deljiv sa 11 ako i samo ako je $s_n - s_p$ deljivo sa 11. Na primer, za broj 190949 je $s_n = 5$, $s_p = 27$, $s_n - s_p = -22$, što je deljivo sa 11, pa je i taj broj deljiv sa 11 (zapravo, $190949 = 11 \cdot 17359$).

7.11. Isto kao u prethodnom zadatku, samo što se proverava deljivost sa 33.

7.12. Napisati Paskal program koji utvrđuje da li je zbir dva data *velika* broja deljiv sa 3. Veliki broj je broj sa proizvoljno mnogo cifara koje se unose u jednoj liniji kao niz karaktera.

†**7.13.** U sledećoj tabeli data je atomska masa nekih hemijskih elemenata:

H	1.0079	O	15.9990	F	18.9984
He	4.0026	Na	22.9898	Au	196.9665
Be	9.0122	S	32.0600	Hg	200.5000
C	12.0110	Cl	35.4530	Ra	226.0254

Napisati program koji od korisnika učitava hemijsku formulu nekog molekula i određuje i štampa njegovu molekularnu masu. Formula molekula se unosi kao što smo na to i navikli, s tim da se brojevi ne pišu kao indeksi. Na primer: `C2H5OH`, `NaCl`, `H2O`, `H2SO4`.

7.14. Napisati program za igranje igre pogađanja reči. Prvi igrač unese reč koju drugi igrač treba da pogodi. Računar prikaže na ekranu niz od onoliko zvezdica koliko slova sadrži reč. Drugi igrač onda pogađa slova. Ako uneta reč ne sadrži slovo, računar uveća brojač negativnih poena za jedan. Ako uneta reč sadrži slovo, računar prikaže reč u kojoj su sva pogođena slova otkrivena.

Primer:

```
********** Negativnih poena: 0
 -> a
*****a*a*** Negativnih poena: 0
 -> e
*****a*a**e Negativnih poena: 0
 -> z
*****a*a**e Negativnih poena: 1
 -> k
k*k**aka**e Negativnih poena: 1
 -> m
k*k**aka**e Negativnih poena: 2
 -> o
koko*aka**e Negativnih poena: 2
 -> d
kokodaka**e Negativnih poena: 2
 -> n
kokodakan*e Negativnih poena: 2
 -> j
kokodakanje Negativnih poena: 2
```

7.15. Napisati Paskal program koji proverava da li dati niz slova predstavlja *palindrom* nakon izbacivanja razmaka, znakova interpunkcije i zanemarivanja velikih i malih slova. Palindrom je reč koja se isto čita i sleva i zdesna. Na primer, za sledeće rečenice program treba da prijavi da su palindromi:

```
Sir ima miris.
Sava zidar radi za vas!
Ana voli Milovana!
Maja sa Nedom ode na sajam.
```

7.16. Napisati Paskal program koji konvertuje iznos iskazan u evrima, u dolare i obrnuto, imajući u vidu da u trenutku pisanja ovog teksta 1 dolar vredi 1.05 evra. Korisnik unosi niz simbola koji predstavlja zapis valute tako što prvo unese oznaku valute, "$" za dolare ili "E" za evre, a odmah iza toga i iznos. Na primer:

```
Unesite iznos:      ili     Unesite iznos:
$210                        E136
$210 = E200.00              E136 = $142.80
```

7.17. Napisati program koji iz dva uneta stringa izdvaja njihov najduži zajednički podstring.

7.18. Napisati program koji iz unetog stringa uklanja sve podstringove koji se nalaze između zagrada `(` i `)`, kao i zagrade. Zagrade mogu biti ugnježdene i zna se da, ako ih ima, onda su izbalansirane.

7.19. Za potrebe ovog zadatka, broj decimala nekog decimalnog broja koje su navedene u njegovom decimalnom zapisu zvaćemo *preciznost* tog broja. Tako, broj 12.3430 ima preciznost 4, broj -175.00 ima preciznost 2, dok broj 75 ima preciznost 0. *Preciznost niza brojeva* je najmanja preciznost broja u tom nizu. Na primer:

Niz	*Preciznost*
12.5660 -1.726656 9.300	3
13.9483 55 43.00 1496.3432	0

Napisati Paskal program koji od korisnika učitava string koji sadrži niz decimalnih brojeva razdvojenih bar jednim razmakom i potom računa i štampa preciznost tog niza.

7.20. (MIKA - Mali Integralni KAlkulator) Napisati Paskal program koji korisnika učitava algebarski izraz oblika

$$\langle broj \rangle \langle operacija \rangle \langle broj \rangle$$

i računa njegovu vrednost. Ceo izraz je niz karaktera bez razmaka koji se

nalazi u jednom redu. Brojevi koji su navedeni su pozitivni celi brojevi koji mogu da stanu u `longint`, a ⟨*operacija*⟩ je jedan od ovih karaktera: `+` (sabiranje), `-` (oduzimanje), `*` (množenje), `/` (celobrojni količnik – div), `%` (ostatak pri celobrojnom deljenju – mod).

Primer 1:
```
Izraz: 123*45
5535
```

Primer 2:
```
Izraz: 1247%13
12
```

Primer 3:
```
Izraz: 1247/13
95
```

7.3 Tekstualne datoteke

Datoteka je proizvoljno dugačak niz podataka istog tipa koji se nalazi u spoljašnjoj memoriji, najčešće na disku. Postoje razne vrste datoteka, a mi ćemo sada pokazati kako Paskal radi sa jednom posebnom vrstom datoteka koje se zovu *tekstualne datoteke* ili *tekstovi*.

Tekstualna datoteka je niz karaktera koji je smešten na disk i ima svoje ime. Na primer, kada u nekom ASCII editoru kao što je Notepad otkucamo neki tekst i "snimimo" ga, operativni sistem ga smesti tekstualnu datoteku na disku. Programski jezik Paskal ima način da pristupa takvim datotekama, da ih kreira i da ih menja. Podacima iz neke tekstualne datoteke se iz Paskal programa pristupa preko promenljive tipa `text` čija deklaracija izgleda ovako:

```
var
    f : text;
```

U programskom jeziku Paskal se podaci iz datoteke mogu ili samo čitati, ili je moguć samo upis u datoteku. Nije moguće istovremeno i pisati i čitati iz iste datoteke. Pre početka rada sa datotekom potrebno je vezati promenljivu za neku fizičku datoteku (koja se najčešće nalazi na hard disku) i naglasiti programu da li će podaci biti samo čitani iz datoteke ili samo upisivani u datoteku. Dodela fizičke datoteke promenljivoj nije standardizovana, tako da svaka implementacija nudi svoje rešenje. Implementacija Paskala koju mi koristimo ima posebnu komandu `assign` koja vezuje datotečku promenljivu za neku konkretnu datoteku na disku.

Ukoliko želimo da pročitamo sadržaj datoteke `spisak.txt` koja se nalazi na disku, možemo je vezati za promenljivu `f` naredbom `assign` kako je to pokazano u primeru pored.

```
program TekstDat;
var
  f : text;
begin
  assign(f, 'spisak.txt');
  ...
end.
```

Naredba `reset` priprema datoteku za čitanje, a naredba `rewrite` priprema datoteku za upisivanje. Kada se rad sa datotekom završi, potrebno ju je zatvoriti naredbom `close`. Primeri tipične upotrebe ovih naredbi su:

```
program CitajIzTekstDat;
var
  f : text;
begin
  assign(f, 'spisak.txt');
  reset(f);
  ...
  close(f)
end.
```

```
program PisiUTekstDat;
var
  f : text;
begin
  assign(f, 'spisak.txt');
  rewrite(f);
  ...
  close(f)
end.
```

Upis u tekstualnu datoteku se obavlja naredbama `write` i `writeln`, kao ispis na ekran, s tim da se kao prvi argument ovih naredbi javlja datotečka promenljiva. Na primer:

```
writeln(f, 'Na kraju smo dobili N = ', 2*k : 10);
```

znači da će u tekstualnu datoteku `f` biti upisan odgovarajući tekst. Sva pravila ponašanja naredbi `write` i `writeln` koja smo do sada naučili važe i za pisanje u tekstualnu datoteku: *sve je isto, samo se umesto na ekran piše u datoteku!*

Slično, čitanje iz tekstualne datoteke se obavlja naredbom `readln`, s tim da se i ovde kao prvi argument javlja datotečka promenljiva. Na primer:

```
readln(f, n, a, b);
```

znači da će iz tekstualne datoteke `f` biti pročitane vrednosti promenljivih `n`, `a` i `b`. Sva pravila ponašanja koja smo do sada naučili važe i za čitanje iz tekstualne datoteke: *sve je isto, samo se umesto sa terminala čita iz datoteke!*

Standardni test `eof` (*end of file*) proverava da li su pročitani svi podaci iz datoteke koja je otvorena za čitanje. Tipični programi koji čitaju podatke iz neke datoteke ili upisuju podatke u neku datoteku izgledaju ovako:

```
program CitamIzDat;
var
  f : text;
  s : string;
begin
  assign(f, 'spisak.txt');
  reset(f);
  ...
  while not eof(f) do
    begin
      ...
      readln(f, s);
      ...
    end;
  ...
  close(f)
end.
```

```
program PisemUDat;
var
  f : text;
  a : integer;
  x : real;
begin
  assign(f, 'spisak.txt');
  rewrite(f);
  ...
  writeln(f, 'A = ', a);
  ...
  writeln(f, 'X = ', x : 12 : 3);
  ...
  close(f)
end.
```

Dakle, nad datotekama tipa `text` moguće je vršiti više operacija i one su navedene u Tabeli 7.1.

Primer. Kao primer pokazaćemo program koji određuje dužinu svakog reda tekstualne datoteke `ulaz.txt` i dobijene brojeve upisuje u datoteku `izlaz.txt`, red po red. Na primer,

ulaz.txt	izlaz.txt
Biti il' ne biti,	17
pitanje je sad!	15
Jel' lepse u dusi trpeti	24
pracke i strele sudbe obesne,	29
ili na oruzje protiv mora bede	30
podici se i borbom	18
uciniti im kraj?	16

Rešenje.

Operacija	Značenje	Primer
`assign`	dodela fizičke datoteke promenljivoj	`assign(f, 'p.txt');`
`reset`	priprema datoteke za čitanje	`reset(f);`
`rewrite`	priprema datoteke za pisanje (ako datoteka nije prazna, stari sadržaj se briše)	`rewrite(f);`
`write`	upis u datoteku; prvi argument je datotečka promenljiva, a ostali argumenti su izrazi proizvoljnog tipa	`write(f, ...);`
`writeln`	upis u datoteku i prelazak u novi red; prvi argument je datotečka promenljiva, a ostali argumenti su izrazi proizvoljnog tipa	`writeln(f, ...);`
`readln`	čitanje podataka iz datoteke; ukoliko u redu ima više podataka nego promenljivih, ostatak reda se ignoriše; prvi argument je datotečka promenljiva, a ostali argumenti su proizvoljnog tipa	`readln(f, ...);`
`close`	zatvaranje datoteke	`close(f);`
`eof`	proverava da li je su pročitani svi podaci iz datoteke	`if eof(f) then ...`

Tabela 7.1: Operacije sa tekstualnim datotekama

```
program DuzineRedova;
var
  f, g : text;
  s : string;
begin
  assign(f, 'ulaz.txt');  reset(f);
  assign(g, 'izlaz.txt'); rewrite(g);
  while not eof(f) do
    begin
      readln(f, s);
      writeln(g, length(s))
    end;
  close(f); close(g)
end.
```

Zadaci.

7.21. Napisati Paskal program koji u datoj tekstualnoj datoteci utvrđuje dužinu najdužeg reda.

7.22. Napisati Paskal program koji u datoj tekstualnoj datoteci utvrđuje broj slova koja nisu razmaci i specijalni simboli, broj reči i broj redova.

7.23. Napisati Paskal program koji formira novu tekstualnu datoteku nadovezivanjem dve tekstualne datoteke.

7.24. Napisati Paskal program koji upoređuje dve tekstualne datoteke i ispisuje prvi red u kome je detektovana razlika.

7.25. Napisati Paskal program koji od tekstualne datoteke pravi novu tako što na početak svakog reda dopiše redni broj tog reda. Na primer,

```
ulaz.txt
```

```
Biti il' ne biti,
pitanje je sad!
Jel' lepse u dusi trpeti
pracke i strele sudbe obesne,
ili na oruzje protiv mora bede
podici se i borbom
uciniti im kraj?
```

izlaz.txt

```
 1: Biti il' ne biti,
 2: pitanje je sad!
 3: Jel' lepse u dusi trpeti
 4: pracke i strele sudbe obesne,
 5: ili na oruzje protiv mora bede
 6: podici se i borbom
 7: uciniti im kraj?
```

7.26. Napisati Paskal program koji od tekstualne datoteke pravi novu tako što na početak svakog petog reda dopiše redni broj tog reda.

7.27. Napisati Paskal program koji od tekstualne datoteke pravi novu tako što na početak svakog reda dopiše redni broj tog reda kao i ukupan broj redova u datoteci. Na primer,

ulaz.txt

```
Biti il' ne biti,
pitanje je sad!
Jel' lepse u dusi trpeti
pracke i strele sudbe obesne,
ili na oruzje protiv mora bede
podici se i borbom
uciniti im kraj?
```

izlaz.txt

```
 1/ 7: Biti il' ne biti,
 2/ 7: pitanje je sad!
 3/ 7: Jel' lepse u dusi trpeti
 4/ 7: pracke i strele sudbe obesne,
 5/ 7: ili na oruzje protiv mora bede
 6/ 7: podici se i borbom
 7/ 7: uciniti im kraj?
```

7.4 Nizovi karaktera i stringova

Kao što smo videli, stringovi predstavljaju specijalne nizove karaktera. Dobra stvar u vezi sa stringovima je to što programski jezik Paskal tretira stringove kao osnovni tip i implementira niz operacija na stringovima. Cena koju za to moramo da platimo je činjenica da je maksimalna dužina stringa ograničena, najčešće na 255 simbola. Ukoliko imamo potrebu da radimo sa dužim nizovima karaktera, postoji mogućnost da ih deklarišemo i koristimo na isti način na koji smo deklarisali i koristili nizove celih brojeva. Dužina ovako deklarisanih nizova je ograničena

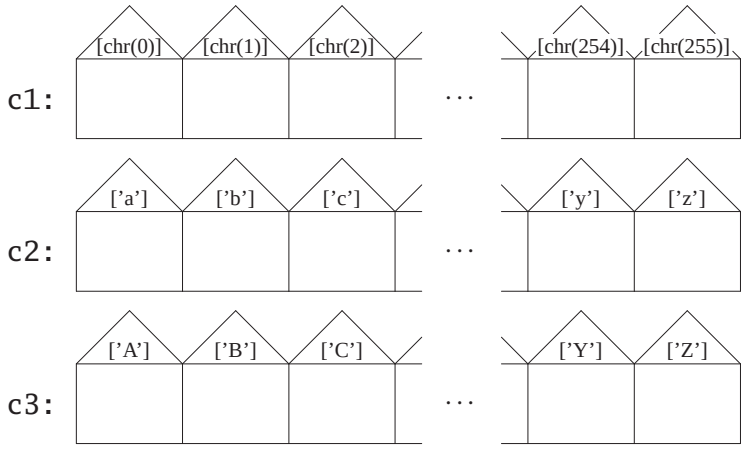
samo količinom raspoložive memorije računara, ali zato gubimo fleksibilnost u radu koju su nudili stringovi. Deklaracija niza karaktera, na primer, izgleda ovako:

```
var
  z : array [1 .. 100] of char;
```

Dodatni kvalitet u radu sa nizovima karaktera je i to što se karakteri se mogu pojaviti i kao indeksi niza. Na primer:

```
var
    c1 : array [char] of integer;
    c2 : array ['a' .. 'z'] of integer;
    c3 : array ['A' .. 'Z'] of char;
```

Promenljive `c1`, `c2` i `c3` izgledaju ovako:



Primer. Napisati program koji od korisnika učitava jedan red teksta, a onda utvrđuje koliko se puta koji karakter pojavio u tom redu.

```
program KarakteriURedu;
var
  a : array [char] of integer;
  c : char;
  s : string;
  i : integer;
begin
  for c := chr(0) to chr(255) do a[c] := 0;
  writeln('Unesite jedan red teksta:');
  readln(s);
  for i := 1 to length(s) do inc(a[s[i]]);
  writeln('Frekvencije:');
  for c := chr(0) to chr(255) do
    if a[c] > 0 then
      writeln(ord(c):3, a[c]:5)
end.
```

Programski jezik Paskal podržava i nizove stringova. Deklaracija niza stringova, na primer, izgleda ovako:

```
var
  ime : array [1 .. 100] of string;
```

Primer. Napisati program koji učitava broj iz skupa $\{1, 2, \ldots, 9999\}$ i ispisuje taj broj rečima na engleskom jeziku. Na primer, za 328 program treba da ispiše:

```
three hundred twenty eight
```

Rešenje.

```
program SayTheNumber;
var
  n, k : integer;
  jedinice : array [1 .. 19] of string;
  desetice : array [2 .. 9]  of string;
begin
  jedinice[1] := 'one';    jedinice[10] := 'ten';
  jedinice[2] := 'two';    jedinice[11] := 'eleven';
  jedinice[3] := 'three';  jedinice[12] := 'twelve';
  jedinice[4] := 'four';   jedinice[13] := 'thirteen';
  jedinice[5] := 'five';   jedinice[14] := 'fourteen';
  jedinice[6] := 'six';    jedinice[15] := 'fifteen';
  jedinice[7] := 'seven';  jedinice[16] := 'sixteen';
  jedinice[8] := 'eight';  jedinice[17] := 'seventeen';
  jedinice[9] := 'nine';   jedinice[18] := 'eighteen';
                           jedinice[19] := 'nineteen';

  desetice[2] := 'twenty'; desetice[6]  := 'sixty';
  desetice[3] := 'thirty'; desetice[7]  := 'seventy';
  desetice[4] := 'fourty'; desetice[8]  := 'eighty';
  desetice[5] := 'fifty';  desetice[9]  := 'ninety';

  write('Broj od 1 do 9999 -> '); readln(n);
  if (n <= 1) or (n >= 9999) then
    writeln('Izvan opsega')
  else
    begin
      k := n div 1000;
      if k > 0 then write(jedinice[k], ' thousand ');
      k := (n div 100) mod 10;
      if k > 0 then write(jedinice[k], ' hunderd ');
      k := n mod 100;
      if k >= 20 then
        begin
          write(desetice[k div 10], ' ');
          k := k mod 10;
        end;
      if k > 0 then write(jedinice[k]);
      writeln
    end
end.
```

Zadaci.

7.28. U sledećoj tabeli označiti deklaracije koje su korektne:

☐
```
var a : array [1 .. 100] of real;
```

☐
```
var a : array [3 .. 3] of real;
```

☐
```
var b : array [-1 .. -3] of real;
```

☐
```
var c : array [2.1 .. 3.3] of real;
```

☐
```
var e : array [char] of boolean;
```

☐
```
var f : array [integer] of char;
```

☐
```
var g : array [real] of integer;
```

☐
```
const N = 13;
var d : array [N .. 20] of char;
```

☐
```
var g : array [boolean] of integer;
```

☐
```
var g : array [false .. true] of char;
```

☐
```
var g : array ['a' .. 'z'] of char;
```

7.29. Napisati program koji od korisnika učitava pozitivan ceo broj *n*, potom *n* redova teksta, i utvrđuje koliko se puta u tim redovima javlja svako od slova *a*, ..., *z*, nezavisno od toga da li se u redu pojavljuje kao veliko ili kao malo slovo. Na primer, u redu *Ana voli Milovana* se slovo *a* javlja 4 puta, a slovo *m* samo jednom.

7.30. Jedan od najstarijih metoda šifrovanja poruka sastoji se u tome da se svako slovo poruke na sistematičan način zameni nekim drugim slovom. Na primer, ako nam je data tabela zamene:

Slovo	a b c d e f g h i j k l m n o p q r s t u v w x y z
Zamena	q w e r t y u i o p a s d f g h j k l z x c v b n m

onda poruka *Sir ima miris* postaje *Lok odq dokok*. Niz od 26 slova

qwertyuiopasdfghjklzxcvbnm

u tabeli zamene zove se *ključ šifre*.

(*a*) Tekstualna datoteka `kljuc.txt` sadrži jedan red teksta koji predstavlja ključ šifre. Napisati Paskal program koji tekst u datoteci `poruka.txt` šifruje tim ključem i šifrovanu poruku upisuje u datoteku `sifra.txt`.

(*b*) Tekstualna datoteka `kljuc.txt` sadrži jedan red teksta koji predstavlja ključ šifre. Napisati Paskal program koji dešifruje tekst u datoteci `sifra.txt` i dobijenu poruku upisuje u datoteku `original.txt`.

7.31. Morzeov kōd izgleda ovako:

A	·–	H	····	O	–––	U	··–
B	–···	I	··	P	·––·	V	···–
C	–·–·	J	·–––	Q	––·–	W	·––
D	–··	K	–·–	R	·–·	X	–··–
E	·	L	·–··	S	···	Y	–·––
F	··–·	M	––	T	–	Z	––··
G	––·	N	–·				

Napisati program koji od korisnika učitava string koji se sastoji samo od velikih slova i praznina i kodira ga Morzeovim kodom tako da se između kodova dva slova nalazi jedna praznina, a između reči dve praznine.

7.32. Napisati program koji učitava broj iz skupa $\{1,2,\ldots,2\,000\,000\,000\}$ i ispisuje taj broj rečima na srpskom jeziku. Na primer, za 328 program treba da ispiše:

```
tri stotine dvadeset osam
```

7.5 Sistematizacija

⟨*Tip*⟩ ≡ `real` | `integer` | `longint` | `boolean` | `char` | `text` ... (ima ih još)

⟨*Naredba*⟩ ≡
⟨*Naredba dodele*⟩ | ⟨*If-naredba*⟩ | ⟨*For-naredba*⟩ | ⟨*While-naredba*⟩ |
⟨*Repeat-naredba*⟩ | ⟨*Standardna procedura*⟩ ... (ima ih još)

⟨*Standardna procedura*⟩ ≡
⟨*Naredba čitanja*⟩ | ⟨*Naredba pisanja*⟩ | ... (ima ih još)

⟨*Naredba čitanja*⟩ ≡
⟨*Naredba čitanja sa terminala*⟩ | ⟨*Naredba čitanja iz datoteke*⟩

⟨*Naredba pisanja*⟩ ≡
⟨*Naredba pisanja na terminal*⟩ | ⟨*Naredba pisanja u datoteku*⟩

⟨*Naredba čitanja sa terminala*⟩ ≡
`readln(`⟨*promenljiva*⟩`, ..., `⟨*promenljiva*⟩`)`

⟨*Naredba čitanja iz datoteke*⟩ ≡
    `readln(`⟨*datotečka promenljiva*⟩`,` ⟨*promenljiva*⟩`, ..., ` ⟨*promenljiva*⟩`)`

⟨*Naredba pisanja na terminal*⟩ ≡
    `write(`⟨*izlazna stavka*⟩`, ..., ` ⟨*izlazna stavka*⟩`)` |
    `writeln(`⟨*izlazna stavka*⟩`, ..., ` ⟨*izlazna stavka*⟩`)`

⟨*Naredba pisanja u datoteku*⟩ ≡
    `write(`⟨*datotečka prom*⟩`,` ⟨*izlazna stavka*⟩`, ..., ` ⟨*izlazna stavka*⟩`)` |
    `writeln(`⟨*datotečka prom*⟩`,` ⟨*izlazna stavka*⟩`, ..., ` ⟨*izlazna stavka*⟩`)`

⟨*izlazna stavka*⟩ ≡
    ⟨*string*⟩ | ⟨*izraz*⟩ | ⟨*celobrojni izraz*⟩ `:` ⟨*ceolobrojna konstanta*⟩ |
    ⟨*realan izraz*⟩ `:` ⟨*ceolobrojna konstanta*⟩ `:` ⟨*ceolobrojna konstanta*⟩

⟨*string*⟩ ≡ `'`⟨*niz slova*⟩`'`

Glava 8

Funkcije i procedure

Programski jezik Paskal ima standardnu (“ugrađenu”) funkciju `sqr` koja računa kvadrat nekog broja, ali nema funkciju koja ume da računa *NZD* dva broja. S druge strane, mi *znamo* da napišemo Paskal program koji računa *NZD* dva broja. Tako se prirodno postavlja pitanje: Da li je moguće Paskal na neki način “proširiti” novom funkcijom `NZD`?

Dalje, programski jezik Paskal ima standardni test `odd` koji ume da utvrdi da li je dati ceo broj neparan, ali nema “ugrađen” test kojim se može proveriti da li je neki ceo broj prost. Ponovo, mi *znamo* da napišemo Paskal program koji utvrđuje da li je dati broj prost. Da li je moguće Paskal na neki način “proširiti” novim testom `Prost`?

Programski jezik Paskal ima standardnu proceduru `writeln` koja ume da ispiše neki ceo broj u decimalnom sistemu, ali nema proceduru kojom bi se dati broj mogao ispisati rečima. Da li je moguće Paskal na neki način “proširiti” novom komandom `SpellOutNumber` koja to radi?

Ova tri retorička pitanja, naravno, imaju očekivani odgovor: DA. U programskom jeziku Paskal postoji način da korisnik doda svoje funkcije, testove i komande koje često koristi. Time program postaje kraći, čitkiji i jednostavniji. Postoji mogućnost da se deo programa koji radi jedan nezavisan i smislen posao izdvoji u posebnu celinu, *potprogram*, koji se pozove kadgod nam trebaju njegove usluge. Paskal poznaje dve vrste potprograma: funkcije, i procedure. Funkcije predstavljaju potprograme koji nešto računaju i vraćaju rezultat. Rezultat funkcije se koristi unutar nekog komplikovanijeg algebarskog ili logičkog izraza. Procedure predstavljaju nove komande programskog jezika.

8.1 Funkcije

Funkcija je potprogram koji nešto računa i onda vrati neku vrednost. Pokazaćemo na jednom primeru zašto je zgodno kotistiti funkcije. Da se podsetimo, za prirodan broj n, Ojlerova funkcija $\varphi(n)$ se definiše kao broj elemenata skupa $\{1,2,3,\ldots,n\}$ koji su uzajamno prosti sa n.

Pored je dato jedno rešenje zadatka koje je napisano pod pretpostavkom da je $n \geqslant 1$. Zato što je $\mathrm{NZD}(1,n) = 1$ i zato što je $\mathrm{NZD}(n,n) = n > 1$ za $n > 1$, postavili smo odmah `phi` na 1, a u "for"-ciklusu proveravamo samo brojeve od 2 do $n-1$.

Rešenje bi bilo veoma jednostavno da nema jedne male poteškoće: Euklidov algoritam *uništava polazne vrednosti svojih promenljivih*. Zato je pre primene Euklidovog algoritma potrebno iskopirati vrednosti promenljivih `n` i `k` u pomoćne promenljive `a` i `b`, i onda pustiti algoritam da izračuna NZD brojeva `a` i `b`. (Ako bismo pustili Euklidov algoritam da radi direktno sa promenljivim `n` i `k`, vrednosti ovih promenljivih bi bile uništene i program ne bi radio korektno.)

```
program OjlerovaFja;
var
  n, k, phi : integer;
  a, b, r : integer;
begin
  readln(n);
  phi := 1;
  for k := 2 to n - 1 do
    begin
      a := n;
      b := k;
      repeat
        r := a mod b;
        a := b;
        b := r
      until r = 0;
      if a = 1 then
        phi := phi + 1
    end;
  writeln(phi)
end.
```

Evo sada elegantnijeg rešenja gde smo uveli novu funkciju `NZD`. Deklaracija funkcije kaže Paskal prevodiocu da se nova funkcija zove `NZD`, da prima dva cela broja koji će unutar funkcije biti označeni sa `a` i `b` i da će rezultat funkcije biti ceo broj. Funkcija ima i svoju pomoćnu promenljivu `r` koja se ne vidi u ostatku programa. Poslednji red u bloku sadrži specijalnu naredbu dodele koja kaže koji izraz treba uzeti kao rezultat funkcije.

```
program OjlerovaFja2;
var
  n, k, phi : integer;

function NZD(a, b: integer): integer;
var
  r : integer;
begin
  repeat
    r := a mod b;
    a := b;
    b := r
  until r = 0;
  NZD := a
end;
```

Između ostalog, vidimo da više nema potrebe za kopiranjem promenljivih pre poziva Euklidovog algoritma. Detalje o tome kako se tačno aktivira funkcija, tj. koje sve aktivnosti preduzima Paskal prevodilac kada se pozove nova funkcija videćemo u narednom odeljku.

```
begin
  readln(n);
  phi := 1;
  for k := 2 to n - 1 do
    if NZD(n, k) = 1 then
      phi := phi + 1;
  writeln(phi)
end.
```

Vidimo da je na ovaj način program postao razumljiviji i jednostavniji. Jednom kada smo uveli novu funkciju, ona se poziva kao i funkcije koje je Paskal poznavao od ranije.

Treće rešenje koje nudimo je još jednostavnije i elegantnije. Osim funkcije `NZD`, i raŭnanje Ojlerove funkcije smo izdvojili u poseban potprogram. Funkcija `Phi` koristi funkciju `NZD`, a glavni deo programa je postao krajnje jednostavan: nakon učitavanja broja `n` ispišemo vrednosti funkcije `Phi(n)`.

Potprogrami se navode između deklaracije promenljivih (`var`-odeljak) i glavnog dela programa. Može ih biti više, a svaki ima strukturu pravog malog programa. Treba jedino obratiti pažnju na to da se kao poslednji red u bloku javlja specijalna naredba dodele koja kaže koji izraz treba uzeti kao rezultat funkcije.

```
program OjlerovaFja3;
var
  n, k : integer;

function NZD(a, b: integer): integer;
var
  r : integer;
begin
  repeat
    r := a mod b;
    a := b;
    b := r
  until r = 0;
  NZD := a
end;

function Phi(n: integer): integer;
var
  f, k : integer;
begin
  f := 1;
  for k := 2 to n - 1 do
    if NZD(n, k) = 1 then
      f := f + 1;
  Phi := f
end;

begin
  readln(n);
  writeln(Phi(n))
end.
```

Test je funkcija koja ispituje osobinu nekog objekta i onda vraća logičku vrednost. Pogledajmo i jedan takav primer. Podsetimo se prvo da smo za neparan

prirodan broj p rekli da je je lep prost, ako su p i $q = \frac{p-1}{2}$ prosti brojevi. Program koji ispituje da li je broj lep prost dat je pored.

Prvo smo deklarisali novi test, `Prost`, koga Paskal ranije nije poznavao, a onda smo ga koristili da utvrdimo da li je učitani broj lep prost.

Uvođenjem novih funkcija programi postaju kraći, jasniji, prirodniji, "čitkiji". Čitkost je jedna od bitnih osobina programa. Program koji liči na špagete bolonjeze se najlakše piše, ali najteže shvata i kasnije popravlja. Doista, u računarskom žargonu i postoji termin "špageti programiranje" (*spaghetti programming*) koji označava zbrda-zdola skarabudžene programe. Postoje standardne preporuke čijim poštovanjem dolazimo do lepih, čitkih i prirodnih programa.

```
program LepProst;
var
  p, q : integer;

function Prost(n: integer): boolean;
var
  d, s : integer;
  ok : boolean;
begin
  ok := odd(n) or (n = 2); d := 3;
  s := trunc(sqrt(n));
  while ok and (d <= s) do
    begin
      ok := n mod d <> 0;
      d := d + 2
    end;
  Prost := ok
end;

begin
  readln(p);
  q := (p - 1) div 2;
  if not odd(p) or (p <= 2) then
    writeln('nije prost')
  else if Prost(p) and Prost(q) then
    writeln('jeste lep prost')
  else
    writeln('nije lep prost')
end.
```

Funkcija se poziva (aktivira) tako što se na odgovarajućem mestu u programu navede njeno ime, a u zagradi se navede vrednost za koju treba proveriti da li je prost broj. Prilikom pozivanja funkcije argument ne mora biti samo promenljiva, već to može biti i čitav izraz. Računar će prvo izračunati vrednost tog izraza, pa će primeniti funkciju na dobijenu vrednost. Glavni deo program `LepProst` smo, dakle, mogli napisati i ovako →

```
  readln(p);
  if not odd(p) or (p <= 2) then
    writeln('nije lep prost')
  else if Prost(p) and
          Prost((p - 1) div 2) then
    writeln('jeste lep prost')
  else
    writeln('nije lep prost')
```

Kao sledeći primer, napisaćemo Paskal program koji računa vrednost izraza

$$x^{23} - 8x^{11} + 12x^{-5}$$

koristeći funkciju `St` koja računa x^n. Funkcija radi korektno za $n \geqslant 0$. Funkcija `St` ima dva argumenta, `x` i `n`, i njihove deklaracije unutar liste argumenata su razdvojene simbolom tačka-zarez. Kada pozivamo funkciju `St`, uvek na prvo mesto moramo staviti promenljivu ili izraz čiji tip je realan, a na drugo mesto promenljivu ili izraz čiji tip je ceo broj. Naravno, *u pozivu funkcije* `St` *argumenti su razdvojeni običnim zarezom*.

```
program Izraz;
var
  x, izr : real;

function St(x: real; n: integer): real;
var
  rez: real;
  i: integer;
begin
  rez := 1.0;
  for i := 1 to n do
    rez := rez * x;
  St := rez
end;

begin
  write('x = '); readln(x);
  izr := St(x, 23) - 8*St(x, 11) +
         12/St(x, 5);
  writeln(izr : 20 : 8)
end.
```

I za kraj dajemo primer funkcije `max` koja određuje maksimum dva broja. U ovom primeru vidimo da naredba dodele koja imenu funkcije dodeljuje vrednost funkcije ne mora biti *fizički* poslednja naredba u telu funkcije, ali mora biti *poslednja izvršna naredba*.

```
function max(x, y: real): real;
begin
  if x > y then
    max := x
  else
    max := y
end;
```

Na osnovu ovih primera možemo da zaključimo sledeće:

- funkcija može imati više argumenata koji mogu, a ne moraju biti istog tipa;
- funkcija ima tip rezultata, i to može biti svaki standardni tip (`real`, `integer`, `longint`, `boolean`, `char`, `string`), kao i neki drugi tipovi o kojima ćemo govoriti kasnije;
- funkcija može imati svoje privatne promenljive;
- telo funkcije je blok čija *poslednja izvršna naredba* mora biti naredba dodele kojom se imenu funkcije simbolično dodeljuje vrednost koja predstavlja rezultat rada potprograma.

8.2 Deklaracija funkcije

Pogledajmo sada korak po korak kako se deklarišu funkcije. Funkcija je pravi jedan mali program. Ona ima svoje promenljive, konstante i sve drugo sto može da ima i program. Naravno, umesto rezervisane reči `program` stoji rezervisana reč `function`, a iza `end` u deklaraciji funkcije ne stoji tačka, već tačka-zarez.

Prvi red daje funkciji ime. U ovom slučaju, funkcija se zove `Prost`. Vidimo da ona ima jedan argument koji se zove `n` i mora biti tipa `integer`. Deklaracija "`: boolean`" na kraju se zove *tip rezultata*. Dakle, ova funkcija nešto radi sa promenljivom `n`, i onda vrati logičku vrednost.

Tip rezultata funkcije može biti *bilo koji prost tip*. To znači da može biti `integer`, `longint`, `real`, `boolean`, `char`, kao i svaki od prostih tipova sa kojima ćemo se sresti kasnije.

Potom slede deklaracije promenljivih funkcije. Ove promenljive se zovu *lokalne* promenljive (za razliku od promenljivih `p` i `q` koje su *globalne*). Promenljive `d`, `s` i `ok` postoje samo u ovoj funkciji i nigde više. Samo ona može da ih koristi, a nedostupne su izvan funkcije.

Lokalna promenljiva može imati isto ime kao i neka globalna promenljiva. Tada lokalna promenljiva "prekriva" globalnu, o čemu ćemo detaljno govoriti kasnije.

```
function Prost(n: integer): boolean;
var
  d, s : integer;
  ok : boolean;
begin
  ok := true; d := 3;
  s := trunc(sqrt(n));
  while ok and (d <= s) do
    begin
      ok := n mod d <> 0;
      d := d + 2
    end;
  Prost := ok
end;
```

```
function Prost(n: integer): boolean;
var
  d, s : integer;
  ok : boolean;
begin
  ok := true; d := 3;
  s := trunc(sqrt(n));
  while ok and (d <= s) do
    begin
      ok := n mod d <> 0;
      d := d + 2
    end;
  Prost := ok
end;
```

☞ *Jedna od prvih preporuka je da se imena promenljivih, konstanti, funkcija, procedura, . . . , biraju tako da podsećaju na ono čemu odgovarajući entitet služi. Mi smo mogli funkciju* `Prost` *nazvati* `F` *ili* `Glb` *i program bi radio jednako dobro, ali bi čitaocu trebalo mnogo više vremena da shvati šta on zapravo radi.*

Posle davanja imena funkciji i raznih deklaracija sledi blok, *telo funkcije*, koji objašnjava šta i kako funkcija računa. Telo funkcije je najobičniji blok: spisak naredbi ograđenih sa `begin` i `end` iza koga ide tačka-zarez.

Jedina razlika u odnosu na ostale blokove koje smo do sada videli je naredba kojom se govori koju vrednost će funkcija vratiti. Ta naredba ima oblik:

⟨*Ime funkcije*⟩ `:=` ⟨*Izraz*⟩

i nipošto je ne treba brkati sa naredbom dodele. *Ime funkcije nije promenljiva koja se može koristiti u procesu računanja!* Računanje se mora obaviti sa nekim običnim promenljivim, a na kraju se rezultat računanja koji predstavlja vrednost funkcije simbolično "dodeljuje" imenu funkcije.

```
function Prost(n: integer): boolean;
var
  d, s : integer;
  ok : boolean;
begin
  ok := true; d := 3;
  s := trunc(sqrt(n));
  while ok and (d <= s) do
    begin
      ok := n mod d <> 0;
      d := d + 2
    end;
  Prost := ok
end;
```

```
function Prost(n: integer): boolean;
var
  d, s : integer;
  ok : boolean;
begin
  ok := true; d := 3;
  s := trunc(sqrt(n));
  while ok and (d <= s) do
    begin
      ok := n mod d <> 0;
      d := d + 2
    end;
  Prost := ok
end;
```

Ako neka funkcija `F` ima dva argumenta, `p` i `q`, koji su istog tipa, recimo `integer`, onda možemo da je deklarišemo ovako:

```
function F(p: integer; q: integer) : real;
```

ali, može i ovako:

```
function F(p, q: integer) : real;
```

Naravno, funkcija može imati i više od dva argumenta i oni ne moraju biti istog tipa. Funkcija koja ima jedan realan i jedan celobrojni argument deklariše se ovako:

```
function F(p: real; q: integer) : real;
```

dok funkcija koja ima dva realna i dva celobrojna argumenta može biti deklarisana na bilo koji od ova četiri načina:

```
function F(p, q: real; r, s: integer) : real;
function F(p: real; q: real; r, s: integer) : real;
function F(p, q: real; r: integer; s: integer) : real;
function F(p: real; q: real; r: integer; s: integer) : real;
```

☞ *Argumenti funkcije su obične promenljive koje možemo tako i da koristimo. Ime funkcije* nije *promenljiva, i njega ne možemo koristiti u toj ulozi! Ime funkcije se javlja samo na kraju računanja, kada mu* simbolično *dodeljujemo vrednost računanja. Ova simbolična naredba dodele mora biti poslednja izvršna naredba u bloku.*

Kviz.

1. Označiti korektno zapisana zaglavlja funkcija:

☐ `function Sredina(x, z);`
☐ `function Veci(x, y : integer);`
☐ `function NOR(p, q : boolean) : boolean;`
☐ `function DoIt(x : char; c : real) : integer;`

2. Svaka od sledećih funkcija bi trebala da vrati ceo deo realnog broja x. Označiti one koje su korektno zapisane.

☐
```
function f(x): integer;
begin
  f := trunc(x)
end;
```

☐
```
function f(x: real): integer;
begin
  x := trunc(x)
end;
```

☐
```
function f(x);
begin
  f := trunc(x)
end;
```

☐
```
function f(x: real): integer;
begin
  trunc(x)
end;
```

☐
```
function f(x: real): integer;
begin
  f := trunc(x)
end;
```

3. Funkcija `max` određuje veći od dva broja:

```
function max(x, y: integer): integer;
begin
  if x > y then
    max := x
  else
    max := y
end;
```

Označiti pozive funkcije `max` koji su korektni:

☐ `max(0, 0)`	☐ `max(y, x)`
☐ `max(2, 3, -1)`	☐ `max(2)`
☐ `max(x+1, 2*a)`	☐ `max(-5, -52)`
☐ `max(max(a, b), max(c, d))`	☐ `max`

4. Kako se sve može deklarisati funkcija `F(...): boolean` koja ima pet argumenata: dva tipa `integer`, dva tipa `real` i jedan tipa `char`, tačno tim redom?

Zadaci.

8.1. Napisati funkciju `sgn` (*signum* – znak, lat.) koja se definiše ovako:

$$\mathtt{sgn(x)} = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0. \end{cases}$$

8.2. Funkcija f je data sa

$$f(x) = \begin{cases} \dfrac{2x-15}{x^2+4}, & \text{za } x < 0 \\ \dfrac{1+\sqrt{x+3}}{1+x}, & \text{za } 0 \leqslant x < \pi \\ \dfrac{1-x^2}{1+x^2}, & \text{za } x > \pi. \end{cases}$$

Napisati Paskal program koji od korisnika učitava prirodan broj n i računa

$$\frac{f(1)^3}{f(1/1)} + \frac{f(2)^3}{f(1/2)} + \frac{f(3)^3}{f(1/3)} + \ldots + \frac{f(n)^3}{f(1/n)}.$$

8.3. Primetimo da funkcija `Prost` nije univerzalna. Ona radi korektno ako i samo ako je broj koji joj je prosleđen neparan i pozitivan. Napisati, zato, univerzalnu funkciju `Prost` koja će raditi za *svaki* ceo broj n.

8.4. Napisati funkciju `BinKo(n, k)` koja računa vrednost binomnog koeficijenta

$$\binom{n}{k} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}.$$

8.5. Napisati Paskal program koji za pozitivne cele brojeve n i m računa vrednost izraza

$$\sum_{k=0}^{m} \binom{m}{k} (m-k)^n$$

8.6. Napisati Paskal program koji za pozitivan ceo broj n računa vrednost izraza

$$\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} (-1)^k \binom{n}{2k} (n-k)!$$

8.7. Za prirodan broj kažemo da je palindrom ako se i sleva i zdesna čita na isti način (recimo, 1, 22, 121, 1331, 12521 su palindromi). Napisati Paskal program koji od korisnika učitava prirodan broj n i štampa sve brojeve k iz skupa $\{1, 2, \ldots, n\}$ takve da su i k i k^2 palindromi.

8.8. Napisati program koji ispisuje prvih n prirodnih brojeva koji su deljivi sa tačno dva od brojeva 2, 3, 5, 7.

8.9. Napisati program koji ispisuje prvih n prirodnih brojeva koji nemaju prost faktor veći od 5 (dakle, deljivi su samo sledećim prostim brojevima: 2, 3, 5).

8.10. (C) Napisati program koji utvrđuje koliko dana je prošlo između dva datuma.

8.2.1 Aktivacija funkcije

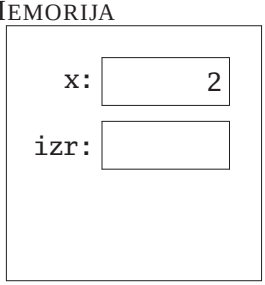
Pogledajmo na jednom primeru kako se koriste funkcije i šta sve računar preživljava kada korisnik pozove funkciju. Niz postupaka koje računar preduzme tim povodom zove se *aktivacija funkcije*.

Pored je dat Paskal program koji računa vrednost izraza

$$25 + x^{10} - 8x^{13}.$$

Kada startujemo program, u memoriji će biti napravljene dve kućice: jedna će se zvati `x`, a druga `izr`. Potom će program učitati od korisnika neki realan broj i upisati ga u fijoku `x`. Recimo da je korisnik uneo broj 2.

MEMORIJA

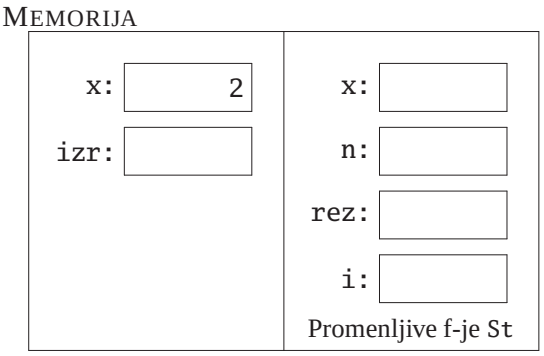



```
program Izraz;
var
  x, izr : real;

function St(x: real; n: integer): real;
var
  rez: real;
  i: integer;
begin
  rez := 1.0;
  for i := 1 to n do
    rez := rez * x;
  St := rez
end {St};

begin
  write('x = '); readln(x);
  izr := 25 + St(x,10) - 0.5*St(x,13);
  writeln(izr : 20 : 8)
end.
```

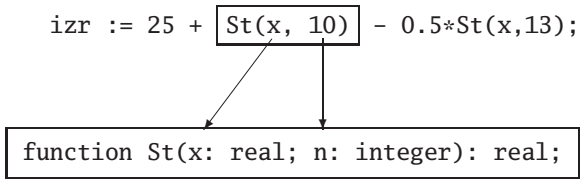
Sledeća naredba je naredba dodele. Njeno izvršavanje je prilično komplikovano zbog dva poziva funkcije `St`. Sada ćemo korak po korak objasniti kako se izračunava vrednost izraza na desnoj strani.

```
program Izraz;
...
begin
  write('x = '); readln(x);
  izr := 25 + St(x,10) - 0.5*St(x,13);
  writeln(izr : 20 : 8)
end.
```

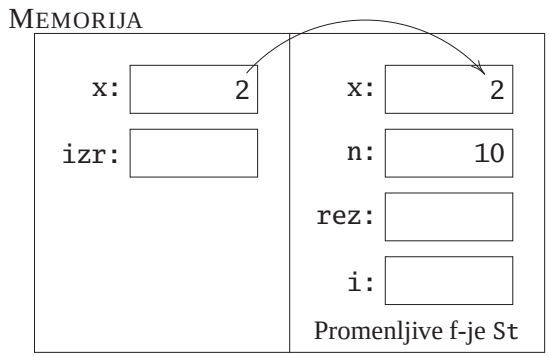
Prvo se računa `St(x, 10)`. Nakon poziva, funkcija `St` otvori svoje četiri fijoke u memoriji: dve za argumente `x` i `n`, i dve za svoje lokalne promenljive `rez` i `i`.



Potom se vrednost promenljive `x` u pozivu funkcije prepiše u fijoku `x` koja pripada funkciji `St`, a broj 10 se upiše u fijoku `n` funkcije `St`.

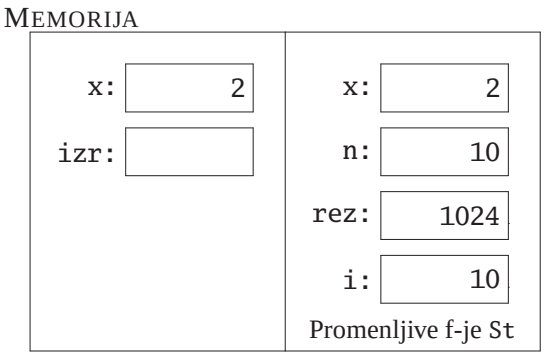



```
var
  rez: real;
  i: integer;
begin
  rez := 1.0;
  for i := 1 to n do
    rez := rez * x;
  St := rez
end {St};
```



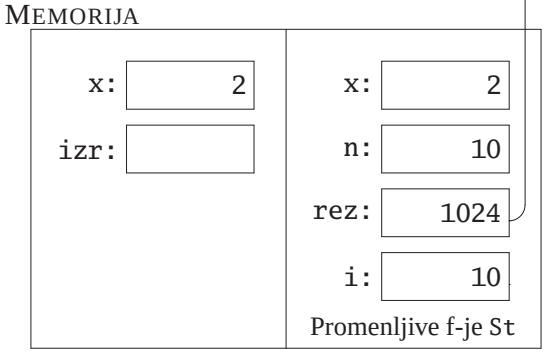
Kada je sve ovo odrađeno, izvrši se telo funkcije `St`. Time se izračuna x^n, u našem primeru 2^{10}, i vrednost upiše u promenljivu `rez`.

```
  izr := 25 + St(x,10) - 0.5*St(x,13);

function St(x: real; n: integer): real;
var
  rez: real;
  i: integer;
begin
  rez := 1.0;
  for i := 1 to n do
    rez := rez * x;
  St := rez
end {St};
```



Poslednjom naredbom u telu funkcije se imenu funkcije simbolično dodeli vrednost. To je rezultat rada funkcije kojim se zamenjuje poziv funkcije.

```
  izr := 25 + 1024 - 0.5 * St(x, 13);

function St(x: real; n: integer): real;
var
  rez: real;
  i: integer;
begin
  rez := 1.0;
  for i := 1 to n do
    rez := rez * x;
  St := rez
end {St};
```



Na kraju, funkcija počisti svoje promenljive iz memorije. Na isti način se izračuna vrednost `St(x, 13)` u izrazu

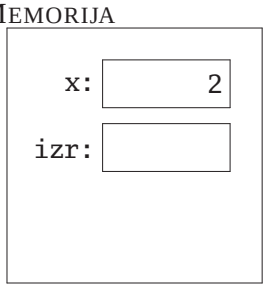
```
izr := 25 + 1024 - 0.5 * St(x, 13);
```

Ponovi se ceo proces: funkcija ponovo otvori svoje četiri fijoke, prepiše odgovarajuće vrednosti, izračuna rezultat i njime zameni poziv funkcije:

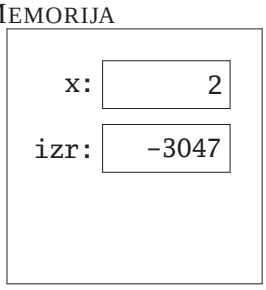
```
izr := 25 + 1024 - 0.5 * 8192;
```

Sada su u izrazu na desnoj strani sve vrednosti poznate, računar izvrši naznačene operacije i dobijenu vrednot upiše u fijoku `izr`.

MEMORIJA



MEMORIJA



☞ Da rezimiramo, aktivacija (poziv) funkcije se izvršava u nekoliko koraka:

- Funkcija rezerviše memorijski prostor za svoje argumente i lokalne promenljive. Pri tome, neki argument ili lokalna promenljiva smeju da imaju isto ime kao i neka globalna promenljiva. Kažemo da je u tom slučaju lokalna promenljiva *sakrila* globalnu promenljivu. Funkcija gleda pre svega svoj memorijski prostor, pa tek ako tu ne uspe da nađe promenljivu sa odgovarajućim imenom, gleda dalje po memoriji.
- Prenesu se argumenati: vrednosti izraza koji se javljaju u pozivu funkcije se upišu u fijoke dodeljene argumentima funkcije.
- Izvrši se telo funkcije. Poslednja izvršna naredba je simbolična dodela rezultata računanja imenu fukcije. Time se funkciji stavlja do znanja šta treba vratiti kao rezultat rada.
- Poziv funkcije se zameni rezultatom računanja funkcije. (Još jedom napominjemo da do fizičke zamene poziva vrednošću zapravo ne dolazi, ali da nam je za sada lakše da ovako razmišljamo!)
- Oslobodi se prostor rezervisan za argumente i privatne promenljive funkcije.

Zadaci.

8.11. Napisati sledeće funkcije:

(a) `IsDigit(c : char) : boolean` koja proverava da li je `c` cifra, tj. jdan od karaktera `'0'`, `'1'`, ..., `'9'`.

(b) `IsLetter(c : char) : boolean` koja proverava da li je `c` slovo, malo ili veliko.

(c) `CAP(c : char) : char` koja za malo slovo vraća odgovarajuće veliko slovo, dok ostale karaktere ne menja.

8.12. Za prirodan broj k, sa $\sigma(k)$ ćemo označiti zbir svih delilaca broja k koji su strogo manji od k. Na primer, $\sigma(8) = 1+2+4 = 7$. Za par prirodnih brojeva (a,b) kažemo da su *prijateljski* ako je $\sigma(a) = b$ i $\sigma(b) = a$. Na primer, $(220,284)$ je par prijateljskih brojeva.

Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 2$ i nalazi sve parove prijateljskih brojeva (a,b) takve da je $a < b$ i $a,b \in \{2,\ldots,n\}$.

8.13. Za prirodan broj n kažemo da je *super-složen* ako je on rekorder po broju delitelja, tj. ima više delitelja nego bilo koji prirodan broj manji od njega. Na primer, 60 je super-složen broj, zato što je to najmanji prirodan broj sa 12 delitelja. Prvih nekoliko super-složenih brojeva su: 2, 4, 6, 12, 24, 36, 48, 60, 120, 180.

Napisati Paskal program koji od korisnika učitava prirodan broj $n \geqslant 2$ i štampa sve super-složene brojeve iz skupa $\{1,2,\ldots,n\}$.

8.14. Neka je $\beta(k)$ broj prostih delilaca broja k. Napisati program koji od korisnika učitava prirodan broj $n \geqslant 2$ i računa vrednost sume

$$\frac{1}{\beta(2)} + \frac{1}{\beta(3)} + \frac{1}{\beta(4)} + \ldots + \frac{1}{\beta(n)}.$$

8.15. Da se podsetimo: za prirodan broj n, Ojlerova funkcija $\varphi(n)$ označava broj elemenata skupa $\{1,2,3,\ldots,n\}$ koji su uzajamno prosti sa n. Napisati Paskal program koji učitava prirodan broj k i nalazi najmanji broj n iz skupa $\{1,2,3,\ldots,2\,147\,483\,647\}$ sa osobinom $\varphi(n) = k$. Ako takav broj ne postoji, prijaviti da ne postoji.

8.16. (C) Uz pomoć računara naći šest rešenja jednačine $\varphi(n) = 16$.

8.17. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj n i potom računa $\varphi(d_1) + \varphi(d_2) + \ldots + \varphi(d_k)$ gde su $d_1,\ldots,d_k$ svi pozitivni delioci broja n.

8.18. Za broj q kažemo da je *pseudoprost* ako nije prost, ali $q|2^q - 2$. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj n i određuje njemu najbliži pseudoprost broj.

8.19. Napisati funkciju

```
function Shorten(s : string; w : integer) : string;
```

koja formatira string `s` na dužinu `w` na sledeći način: ako je dužina stringa `s` manja ili jednaka sa `w`, ne treba ništa menjati. Ukoliko je dužina stringa `s` strogo veća od `w`, onda treba iz sredine stringa izbaciti odgovarajući broj karaktera i zameniti taj deo stringa stringom `'...'` (tri tačke), tako da dužina dobijenog stringa bude tačno `w`. Na primer, poziv funkcije

```
Shorten('Ajaoj sto je skola zgodna', 10)
```

treba da vrati `'Aja...odna'`. Pretpostavlja se da je `w` > 5.

8.3 Procedure

Procedure predstavljaju drugu vrstu potprograma. One predstavljaju nove komande kojima obogaćujemo programski jezik. Ne mogu se pozivati unutar izraza, već se pozivaju kao nezavisne komande.

Deklaracija procedura je veoma slična deklaraciji funkcija. Jedina razlika je u tome što procedura ne vraća nikakvu vrednost (pa nema potrebe za tipom rezultata i dodelom vrednosti imenu procedure kako je to bio slučaj kod funkcija), i što deklaracija, naravno, počinje rezervisanom reči `procedure`.

Pogledajmo za početak jedan jednostavan primer. Program koji sledi ispisuje brojeve od 1 do *n* tim redom, ali svaki unatraške. Na primer, broj 1275 će biti ispisan kao 5721.

```
program IspisiUnatraske;
var
  i, n : integer;

procedure Unatraske(k: integer);
begin
  while k > 0 do
    begin
      write(k mod 10);
      k := k div 10
    end;
  writeln
end;

begin
  readln(n);
  for i := 1 to n do
    Unatraske(i)
end.
```

Primer. Napisati program koji ispisuje kalendar za 2000. godinu.

Rešenje. Ključni deo rešenja je procedura `IspisiMesec(d, n)` ispisuje kalendar za neki mesec u godini, gde je `d` redni broj dana kojim mesec počinje (1 = ponedeljak, 2 = utorak, . . . , 7 = nedelja), a `n` je broj dana u mesecu.

```
program Kalendar2000;
var
  mesec : integer;

  procedure IspisiMesec(d, n : integer);
  var
    i : integer;
  begin
    writeln(' po ut sr ce pe su ne');
    for i := 1 to d-1 do write('   ');
    for i := 1 to n do begin
      write(i:3);
      if (i + d - 1) mod 7 = 0 then writeln
    end
  end;
```

```
begin
  write('Mesec (1..12) -> ');
  readln(mesec);
  if      mesec =  1 then IspisiMesec(6, 31)
  else if mesec =  2 then IspisiMesec(2, 29)
  else if mesec =  3 then IspisiMesec(3, 31)
  else if mesec =  4 then IspisiMesec(6, 30)
  else if mesec =  5 then IspisiMesec(1, 31)
  else if mesec =  6 then IspisiMesec(4, 30)
  else if mesec =  7 then IspisiMesec(6, 31)
  else if mesec =  8 then IspisiMesec(2, 31)
  else if mesec =  9 then IspisiMesec(5, 30)
  else if mesec = 10 then IspisiMesec(7, 31)
  else if mesec = 11 then IspisiMesec(3, 30)
  else if mesec = 12 then IspisiMesec(5, 31)
  else writeln('Pogresan broj meseca')
end.
```

8.3.1 Argumenti procedura

Procedure takođe mogu da vrate neke vrednosti delu programa koji ih je pozvao. One, međutim, za to ne koriste mehanizam koji koriste funkcije, već svoje specijalne argumente. Argumenti procedure mogu biti

- ulazni, i
- ulazno-izlazni.

Ulazni argumenti se deklarišu isto kao kod funkcija:

⟨*ime argumenta*⟩ `:` ⟨*tip argumenta*⟩

Ulazno-izlazni argumenti se deklarišu navođenjem reči `var` pre deklaracije u listi argumenata:

`var` ⟨*ime argumenta*⟩ `:` ⟨*tip argumenta*⟩

Na primer, kod sledeće procedure

```
                          ↓              ↕    ↕
procedure NadjiMinMax(n: integer; var min, max: integer);
```

argument `n` je *ulazni*, a argumenti `min` i `max` su *ulazno-izlazni*. Smisao je sledeći: procedura će kroz ova tri argumenta dobiti neke vrednosti, sa njima će nešto izračunati, i onda će ono što se bude nalazilo u `min` i `max` biti vraćeno kao "rezultat" rada procedure. U telu procedure argumenti procedure se, inače, koriste kao obične promenljive, što oni u stvari i jesu.

Primer. Napisati program koji od korisnika učitava $n \geqslant 5$ brojeva i ispisuje najveći i najmanji od njih.

Rešenje.

```
program MinMax;
var
  n, min, max : integer;

procedure UcitajKolicinu(var k : integer);
begin
  repeat
    write('Unesite broj >= 5: ');
    readln(k)
  until k >= 5;
end;

procedure NadjiMinMax(n : integer; var mi, ma : integer);
var
  i, k : integer;
begin
  readln(k);
  mi := k; ma := k;
  for i := 2 to n do
    begin
      readln(k);
      if k > ma then ma := k;
      if k < mi then mi := k
    end
end;

begin
  UcitajKolicinu(n);
  NadjiMinMax(n, min, max);
  writeln('min = ', min, '  max = ', max)
end.
```

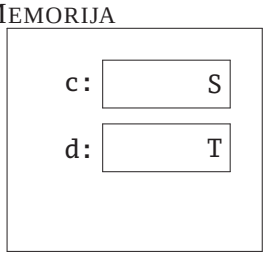
8.3.2 Aktivacija procedure

Aktivacija procedure u svemu podseća na aktivaciju funkcije. Razlike nastaju u tome što procedura ne vraća nikakvu vrednost, dok, s druge strane, kada menjamo vrednost argumentu procedure koji je označen sa `var` tada zapravo menjamo vrednost promenljivoj koja u pozivu procedure stoji na tom mestu. Za razliku od ulazno-izlaznih, izmenom vrednosti ulaznog argumenta ne može promeniti izraz/promenljiva čiju vrednost je argument dobio na početku.

Pogledajmo na primeru jednostavnog programa UI kako funkcionišu ulazno-izlazni argumenti procedure. Kada startujemo program, u memoriji će biti napravljene dve kućice, c i d. U prvu će biti upisan simbol 'S', a u drugu 'T', i na monitoru će biti ispisano ST.

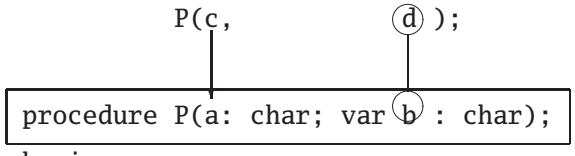
```
program UI;
var
  c, d : char;

procedure P(a: char; var b: char);
begin
  a := 'X';
  b := 'Y'
end;

begin
  c := 'S'; d := 'T';
  writeln(c, d);
  P(c, d);
  writeln(c, d)
end.
```

MEMORIJA



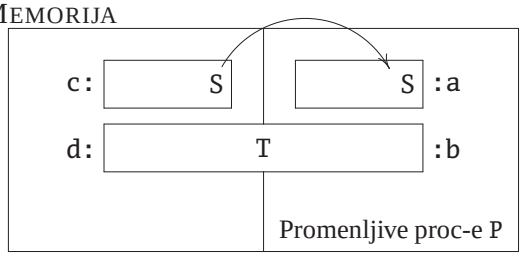
Sledi poziv (aktivacija) procedure P, nakon čega se u memoriji otvore ne dve nove kućice za argumente a i b, nego samo jedna nova kućica, i to za argument a. Vrednost iz kućice c se prepiše u kućicu a, dok promenljiva b *zauzima isti memorijski prostor* kao i promenljiva d.

```
begin
  c := 'S'; d := 'T';
  writeln(c, d);
  P(c, d);
  writeln(c, d)
end.
```




```
begin
  a := 'X';
  b := 'Y'
end;
```

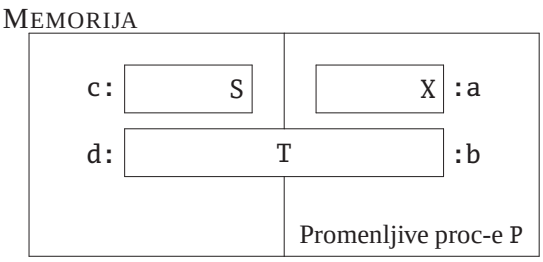
MEMORIJA



Nakon prenosa vrednosti argumenata krećemo sa izvršavanjem tela procedure.

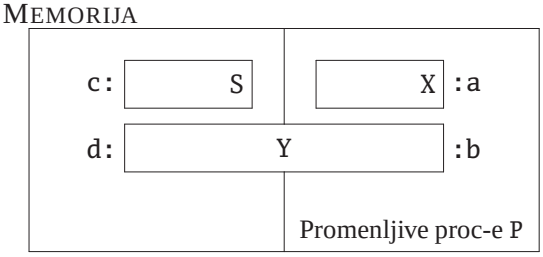
```
procedure P(a: char; var b : char);
begin
  a := 'X';
  b := 'Y'
end;
```

Prva naredba dodele u kućicu a upisuje simbol 'X'.



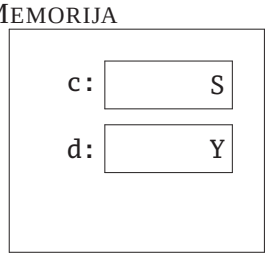

```
procedure P(a: char; var b : char);
begin
  a := 'X';
  b := 'Y'
end;
```

Druga naredba dodele u kućicu b upisuje simbol 'Y'. *No, promenljive* b *i* d *dele isti memorijski prostor!*



Kada računar izvrši sve naredbe procedure P, njene promenljive se uklone iz memorije, i nastavi se sa izvršavanjem glavnog programa. Primetimo da je vrednost promenljive c ostala nepromenjena, dok je promenljiva d dobila novu vrednost. Zato se ovaj put na monitoru pojavljuje SY.

```
begin
  c := 'S'; d := 'T';
  writeln(c, d);
  P(c, d);
  writeln(c, d)
end.
```



☞ Da rezimiramo, aktivacija (poziv) procedure se izvršava u nekoliko koraka:

- Procedura rezerviše memorijski prostor za svoje argumente i lokalne promenljive. Pri tome, neki argument ili lokalna promenljiva smeju da imaju

isto ime kao i neka globalna promenljiva. Kažemo da je u tom slučaju lokalna promenljiva *sakrila* globalnu promenljivu. Procedura gleda pre svega svoj memorijski prostor, pa tek ako tu ne uspe da nađe promenljivu sa odgovarajućim imenom, gleda dalje po memoriji.

- Prenesu se argumenti:
  - za *ulazne* argumente: vrednosti odgovarajućih *izraza* koji se javljaju u pozivu procedure se upišu u kućice dodeljene argumentima procedure;
  - za *ulazno-izlazne* argumente: *promenljive* koje se javljaju u pozivu procedure se *vežu* za odgovarajuće argumente procedure; time se izmenom vrednosti argumenta zapravo menja promenljiva iz poziva procedure.

  Dakle, u pozivu procedure, na mestu ulaznog argumenta može stajati proizvoljan izraz, ali na mestu ulazno-izlaznog argumenta se mora navesti *promenljiva!*
- Izvrši se telo procedure.
- Oslobodi se prostor rezervisan za argumente i privatne promenljive procedure.

Navedimo još i primer programa koji ispisuje prvih n iteracija Newtonovog iterativnog postupka za približno računanje korena pozitivnog realnog broja a. Newtonov iterativni postupak se svodi na sledeću rekurentnu relaciju:

$$x_0 = \frac{a}{2}, \qquad x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

```
program Koren;
var
  a, x : real;
  n, i : integer;

procedure Ucitaj(var a: real; var n: integer);
begin
  repeat
    write('Pozitivan realan broj: ');
    readln(a)
  until a > 0;
  repeat
    write('Broj iteracija (>0): ');
    readln(n)
  until n > 0;
end;
```

```
procedure Popravi(var x: real);
begin
  x := 0.5 * (x + a/x)
end;

begin
  Ucitaj(a, n);
  x := a/2;
  writeln('Iter', 0:3, x:15:10);
  for i := 1 to n do
    begin
      Popravi(x);
      writeln('Iter', i:3, x:15:10)
    end
end.
```

Argument procedure `Popravi` je "pravi" ulazno-izlazni: njegova vrednost "uđe" u proceduru, popravi se, i onda nova vrednost "izađe" iz procedure.

8.3.3 Argumenti funkcija i procedura – cela istina

Sve što je do sada rečeno za argumente procedura *važi i za argumente funkcija*. Dakle, argumenti *i procedura i funkcija* se dele na dve grupe:

- ulazni argumenti, i
- ulazno-izlazni argumenti.

Na primer:

```
procedure P(var n : integer; q : char); ...
function F(x : real; var ok : boolean) : real; ...
```

Deklaracije argumenata procedure i funkcije su razdvojene sa tačka-zarez, ali se prilikom pozivanja procedure ili funkcije odgovarajući izrazi razdvajaju običnim zarezima. Na primer:

```
P(m, '$');
r := F(sqr(x), noErr) / 12;
```

Prilikom poziva (aktivacije) procedure ili funkcije, na mestu ulazno-izlaznog argumenta se *mora navesti promenljiva* (ne sme se navesti konstanta, npr. `12`, ili izraz, npr. `x + 2`). Na mestu ulaznog argumenta može stajati i konstanta i izraz i, naravno, samo promenljiva. Razlog je taj što ulazno-izlazni argument menja vrednost odgovarajuće promenljive, dok ulazni argument preuzme vrednost odgovarajućeg izraza, i posle toga postoji nezavisno od njega. Na primer:

`P(m, '$');`	dobro
`P(m + 1, '$');`	nije dobro: prvi argument procedure `P` ne sme biti izraz
`P(13, c);`	nije dobro: prvi argument procedure `P` ne sme biti konstanta
`r := F(sqr(x), noErr)`	dobro
`r := F(sqr(x), not ok)`	nije dobro: drugi argument funkcije `F` ne sme biti izraz
`r := F(sqr(x), true)`	nije dobro: drugi argument funkcije `F` ne sme biti konstanta

Funkcija i procedura ne moraju imati argumente. Na primer:

```
program Pr1;
var
  i : integer;

  procedure Zdravo;
  begin
    writeln('Zdravo!');
    writeln('Zdravo!');
    writeln('Zdravo!')
  end;

begin
  for i := 1 to 10 do
    Zdravo
end.
```

```
program Pr2;
var
  r, Ob : integer;

  function Pi : real;
  begin
    Pi := 3.14
  end;

begin
  readln(r);
  Ob := 2 * Pi * r;
  writeln(Ob)
end.
```

Ako se kao argument procedure ili funkcije pojavi promenljiva tipa `text`, ona obavezno mora biti deklarisana kao `var` argument. Na primer funkcija koja broji slova u nekoj tekstualnoj datoteci mora biti deklarisana ovako:

```
function BrojSlova(var f: text): integer;
```

dok procedura koja prepisuje sadržaj tekstualne datoteke `f` u tekstualnu datoteku `g` mora biti deklarisana na jedan od ova dva (ekvivalentna) načina:

```
procedure Prepisi(var f, g: text);
procedure Prepisi(var f: text; var g: text);
```

☞ *Iako funkcija sme da ima ulazno-izlazne argumente, pisanje i upotreba takvih funkcija se ne preporučuje, osim u slučaju kada se radi o tekstualnoj datoteci kao parametru funkcije.*

Pisanje funkcija koje vraćaju vrednosti na sve strane smatra se lošim stilom!

Ako imate potrebu da napišete potprogram koji vraća više od jedne vrednosti, ozbiljno razmislite da ga koncipirate kao proceduru koja ima potreban broj ulazno-izlaznih argumenata.

Kviz.

1. Označiti korektno zapisana zaglavlja procedura:

☐ `procedure Print;`
☐ `procedure Err(b : real; c : char) : boolean;`
☐ `procedure Data(i : integer; var x : char);`
☐ `procedure Output(x, var y : real);`

2. Data je procedura `R` ovako:

```
procedure R(var x, y : integer);
begin
    y := x + 1
end;
```

Ako su `c` i `d` celobrojne promenljive, označiti korektne pozive procedure `R`:

☐ `R(1, c);`
☐ `R(c, sqr(d));`
☐ `R(c, d, c);`
☐ `R(c, 1+c);`
☐ `R(-c, d);`
☐ `R(c, d);`
☐ `R(d, c);`
☐ `R(c, c);`
☐ `R(1 - c, c + d);`
☐ `R(c)`

Zadaci.

8.20. Napisati program koji ispisuje kalendar za tekuću godinu.

8.21. Napisati proceduru `procedure Zameni(var a, b : integer);` koja zamenjuje vrednosti svojih argumenata.

8.22. Napisati proceduru `procedure Uredi(var a, b, c : integer);` koja preuredi svoje argumente tako da oni budu u neopadajućem redosledu: $a \leqslant b \leqslant c$.

8.23. Napisati Paskal program koji od korisnika učitava broj $n \geqslant 3$, a potom učitava n razlomaka i štampa njihov zbir. Razlomak se unosi u jednom redu kao par celih brojeva razdvojenih bar jednim razmakom. Razlomak koji se dobija kao rezultat treba da bude skraćen.

8.24. Kutija je određena trojkom realnih brojeva (x,y,z) koji predstavljaju dimenzije kutije. Napisati Paskal program koji od korisnika učitava ceo broj n, potom n kutija (x_i,y_i,z_i) i određuje dužinu najdužeg mogućeg niza kutija koji ima sledeće osobine: na prvom mestu u nizu je prva kutija; na drugom mestu u nizu je prva naredna kutija koja može da stane u prvu kutiju; na trećem mestu u nizu je prva naredna kutija koja može da stane u drugu kutiju, itd. Kutije se smeštaju jedna u drugu tako da im strane budu paralelne.

8.25. Krug je određen trojkom realnih brojeva (x,y,r), gde je (x,y) centar kruga, a r je poluprečnik kruga. Napisati Paskal program koji od korisnika učitava ceo broj n, potom n krugova (x_i,y_i,r_i) i određuje dužinu najdužeg mogućeg niza krugova koji ima sledeće osobine: na prvom mestu u nizu je prvi krug; na drugom mestu u nizu je prvi naredni krug koji je sadržan u prvom krugu; na trećem mestu u nizu je prvi naredni krug koji je sadržan u drugom krugu, itd.

8.26. (C) Napisati proceduru `WriteBig(c : char)` koja ispisuje simbol `c` zvezdicama krupno, u nekoliko redova. Na primer, pored je krupno ispisan simbol `A`. Potom napisati program koji učitava jedan red teksta i ispisuje ga velikim slovima, pri čemu se slova ređaju jedno ispod drugog.

```
  ***
 ** **
**   **
*******
*******
**   **
**   **
```

8.27. Goldbachova hipoteza tvrdi da se svaki paran broj $\geqslant 4$ može predstaviti kao zbir dva prosta broja. Napisati program koji proverava Goldbachovu hipotezu za sve parne brojeve koji ne prelaze dati broj n tako što za svaki od njih ispisuje jednu reprezentaciju u obliku zbira dva prosta broja.

8.4 Lokalne i globalne promenljive

Promenljive koje se deklarišu na nivou programa zovu se *globalne promenljive*. Promenljive deklarisane na nivou procedure ili funkcije, kao i argumenti procedure ili funkcije zovu se *lokalne promenljive*, Slika 8.1.

Globalne promenljive postoje sve dok program radi. Lokalne promenljive postoje samo dok je odgovarajuća procedura ili funkcija aktivna.

```
program MinMax;
var
  n, min, max : integer;
```
← *globalne promenljive*

```
procedure NadjiMinMax(n : integer; var mi, ma : integer);
var
  i, k : integer;
```
← *lokalne promenljive/argumenti*

```
begin
  readln(k);
  mi := k; ma := k;
  for i := 2 to n do
    begin
      readln(k);
      if k > ma then ma := k;
      if k < mi then mi := k
    end
end;

begin
  repeat
    write('Unesite broj >= 5: ');
    readln(n)
  until n >= 5;
  NadjiMinMax(n, min, max);
  writeln('min = ', min, '  max = ', max)
end.
```

Slika 8.1: Globalne i lokalne promenljive

Odatle se jasno vidi da su vrednosti lokalnih promenljivih na raspolaganju samo unutar potprograma. Kažemo još da se *lokalne promenljive ne vide spolja*. Zato u jednostavnom primeru `LokProm` koji je pokazan pored nije moguće iz glavnog programa pristupiti lokalnim promenljivim procedure `P`.

Lokalna promenljiva sme imati isto ime kao neka globalna promenljiva. Tada se unutar potprograma odgovarajuća globalna promenljiva "ne vidi" zato što je "prekrivena" istoimenom lokalnom promenljivom. Inače, sve globalne promenljive se vide u potprogramima (ukoliko, naravno, nisu "pokrivene" nekom lokalnom promenljivom). U primeru `Sakrivanje` lokalna promenljiva `k` procedure `P` prekriva globalnu promenljivu `k`, tako da ispis izgleda ovako:

MONITOR

```
20  30
20   5
20  30
```

```
program LokProm;

procedure P(c: char);
var
  d : char;
begin
  ...
end;

begin
  c := 'a'; ← compilation error!
  d := '+'; ← compilation error!
end.
```

```
program Sakrivanje;
var
  n, k : integer;

procedure P;
var
  k : integer;
begin
  k := 5;
  writeln(n:3, k:3)
end;

begin
  n := 20;
  k := 30;
  writeln(n:3, k:3);
  P;
  writeln(n:3, k:3)
end.
```

Potprogram može da menja vrednosti globalnih promenljivih. Tada se kaže da on pravi *bočne efekte* (engl. *side effects*). Ovu mogućnost treba koristiti umereno i veoma pažljivo zato što njena prekomerna upotreba može da dovede do nejasnih i nečitkih programa, kao i do grešaka koje se veoma teško otkrivaju.

U primeru `BocniEfekti` koji je dat pored procedura `P` menja vrednost globalne promenljive `n` i to je bočni efekat. Ispis izgleda ovako:

MONITOR

```
20 30
 5 30
 5 30
```

```
program BocniEfekti;
var
  n, k : integer;

procedure P;
begin
  n := 5;
  writeln(n:3, k:3)
end;

begin
  n := 20;
  k := 30;
  writeln(n:3, k:3);
  P;
  writeln(n:3, k:3)
end.
```

Kod "for"-ciklusa postoji jedan kuriozitet na koga treba obratiti pažnju: kada se javi u telu procedure ili funkcije, kontrolna promenljiva mora biti *lokalna* promenljiva. Evo primera korektno i nekorektno napisanog for ciklusa u telu potprograma:

```
program DOBRO;
var
  i : integer;

procedure P;
var
  j : integer;
begin
  for j := 1 to 10 do
    writeln('dobro je')
end;

begin
  P
end.
```

```
program POGRESNO;
var
  i : integer;

procedure P;
var
  j : integer;
begin
  for i := 1 to 10 do
    writeln('pogresno')
end;

begin
  P
end.
```

Kviz.

1. Šta ispisuje svaki od sledećih programa?

```
program A;
var
  x : real;

procedure P(y: real);
begin
  y := 1
end;

begin
  x := 0;
  P(x);
  writeln(x)
end.
```

```
program B;
var
  x : real;

procedure P(var y: real);
begin
  y := 1
end;

begin
  x := 0;
  P(x);
  writeln(x)
end.
```

```
program C;
var
  x : real;

procedure P;
begin
  x := 1
end;

begin
  x := 0;
  P;
  writeln(x)
end.
```

```
program D;
var
  x : real;

procedure P;
var
  x : integer;
begin
  x := 1
end;

begin
  x := 0;
  P;
  writeln(x)
end.
```

```
program E;
var
  x : real;

procedure P(y: real);
begin
  x := y
end;

begin
  x := 0;
  P(2);
  writeln(x)
end.
```

2. Šta ispisuje sledeći program:

```
program Kviz1;
var
  a, b, c, d : integer;

  procedure P(var b : integer; c : integer);
  var
    d : integer;
  begin
    a := 5; b := 6; c := 7; d := 8;
    writeln(a, b, c, d)
  end;

begin
  a := 1; b := 2; c := 3; d := 4;
  writeln(a, b, c, d);
  P(a, b);
  writeln(a, b, c, d)
end.
```

3. Šta ispisuje sledeći program:

```
program Kviz2;
var
  x, y, z : integer;

  procedure P(x : integer; var y : integer;
              a, b : integer; var c : integer);
  begin
    x := x+1; y := y+x; z := z+y;
    a := a+z; b := b+a; c := c+b
  end;

begin
  x := 0; y := 1; z := 2;
  P(y, x, z, x, z);
  writeln(x:3, y:3, z:3)
end.
```

Zadaci.

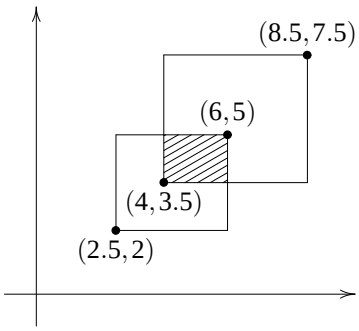
8.28. (C) Napisati program koji štampa kalendar za dati mesec date godine između 1582. i 4096.

8.29. Napisati Paskal program koji nalazi najbliži prost broj datom prirodnom broju.

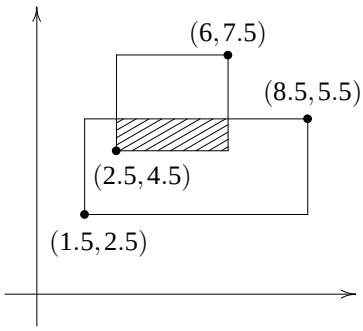
8.30. Napisati Paskal program koji od korisnika učitava ceo broj $n \geqslant 2$ i nalazi najmanji broj koji je strogo veći od sume kvadrata cifara broja n, a može se predstaviti u obliku zbira dva prosta broja.

8.31. Duž na x-osi određena je svojim krajnjim tačkama koje su predstavljene kao brojevi. Na primer, duž $[7,12]$. Napisati Paskal program koji od korisnika učitava dve ovako zadate duži i određuje duž koja im se nalazi u preseku. Na primer, $[7,12] \cap [9,15] = [9,12]$, $[7,12] \cap [9,11] = [9,11]$, $[7,12] \cap [12,15] = [12,12]$, $[7,12] \cap [15,17] = \emptyset$.

8.32. Pravougaonik čije stranice su paralelne koordinatnim osama određen je koordinatama donjeg levog ugla i koordinatama gornjeg desnog ugla. Napisati program koji od korisnika učitava dva ovako određena pravougaonika i određuje površinu njihovog preseka (ako se pravougaonici ne seku, odgovarajuća površina je 0).



Površina je 3



Površina je 3.5

8.33. Napisati program koji za dato $k \in \{2,3,4,5,6\}$ određuje sve k-cifrene brojeve koji su jednaki sumi faktorijela svojih cifara.

8.5 Sistematizacija

⟨*Paskal program*⟩ ≡ `program` ⟨*ime programa*⟩ `;`
[⟨*Definicije konstanti*⟩]
[⟨*Deklaracije promenljivih*⟩]
[⟨*Potprogrami*⟩]
⟨*Blok*⟩ `.`

⟨*Potprogrami*⟩ ≡ ⟨*Potprogram*$_1$⟩
⟨*Potprogram*$_2$⟩
⋮
⟨*Potprogram*$_k$⟩

⟨*Potprogram*⟩ ≡ ⟨*Procedura*⟩ | ⟨*Funkcija*⟩

⟨*Procedura*⟩ ≡ `procedure` ⟨*ime*⟩ [⟨*Argumenti*⟩] `;`
[⟨*Definicije konstanti*⟩]
[⟨*Deklaracije promenljivih*⟩]
[⟨*Potprogrami*⟩]
⟨*Blok*⟩ `;`

⟨*Funkcija*⟩ ≡ `function` ⟨*ime*⟩ [⟨*Argumenti*⟩] `:` ⟨*Tip*⟩ `;`
[⟨*Definicije konstanti*⟩]
[⟨*Deklaracije promenljivih*⟩]
[⟨*Potprogrami*⟩]
⟨*Blok funkcije*⟩ `;`

⟨*Blok funkcije*⟩ ≡ `begin`
⟨*naredba*⟩ `;`
⋮
⟨*naredba*⟩ `;`
⟨*ime funkcije*⟩ `:=` ⟨*izraz*⟩
`end`

⟨*Argumenti*⟩ ≡ `(`⟨*Arg*⟩`;` ⟨*Arg*⟩`; ...)`

⟨*Arg*⟩ ≡ [`var`] ⟨*ime*⟩ `:` ⟨*Tip*⟩

Glava 9

Razvoj programa

U programiranju postoje dve osnovne grupe problema: algoritamski problemi i organizacioni problemi. Do sada smo se sretali samo sa algoritamskim problemima, zato što su zadaci koje smo rešavali bili namerno tako odabrani. Sada kada smo naučili procedure i funkcije možemo da pređemo i na ozbiljnije programerske poduhvate. Problem sa većim programima je u tome što programer mora da vodi računa o izuzetno mnogo pojedinosti i što količina detalja i sitnica o kojima se treba brinuti veoma brzo preraste intelektualne sposobnosti prosečnog ljudskog bića. Neko je zato svojevremeno dao ovakav pogled na programiranje:

Programming is the art of managing complexity!

Postoje mnogi načini (programerske tehnike) koje omogućuju da se izađe na kraj sa ogromnom količinom informacija i zahteva sa kojima se sreću programeri tokom rada. U ovoj glavi ćemo naučiti kako se procedure i funkcije mogu grupisati u biblioteke i videćemo *top-down* tehniku programiranja (tj. tehnika programiranja s vrha na niže), čija osnovna ideja je da se program profinjuje u malim koracima.

9.1 Biblioteke procedura i funkcija (UNITS)

Moderne verzije programskog jezika Paskal omogućuju da se grupe srodnih procedura i funkcija koje operišu nad istom grupom podataka organizuju u *biblioteke*. Kao i običan program, i biblioteka mora da se iskompajlira. Za razliku od običnog programa biblioteka ne može da se izvrši nakon kompilacije. Ideja je da se biblioteke uključuju u druge programe koji onda koriste funkcije i procedure iz tih biblioteka.

Biblioteke se u Paskalu zovu *units* zato što su to nezavisne *jedinice* prevođenja. Svaka Paskal biblioteka ima sledeći oblik:

```
unit ⟨ime biblioteke⟩;
interface
    ⟨javne konstante⟩
    ⟨javni tipovi⟩
    ⟨javne promenljive⟩
    ⟨zaglavlja javnih procedura i funkcija⟩
implementation
    ⟨privatne konstante⟩
    ⟨privatni tipovi⟩
    ⟨privatne promenljive⟩
    ⟨javne i privatne procedure i funkcije⟩
begin
    ⟨inicijalizacija javnih promenljivih⟩
end.
```

Odeljak `interface` sadrži spisak *javnih* konstatni, tipova i promenljivih (što su konstante, tipovi i promenljive koje biblioteka nudi drugim programima), kao i spisak deklaracija javnih funkcija i procedura (što su funkcije i procedure koje biblioteka nudi drugim programima).

Odeljak `implementation` sadrži implementaciju procedura i funkcija koje je biblioteka obećala u odeljku `interface`. Za implementaciju javnih procedura i funkcija će nam možda trebati neke pomoće konstante, tipovi, promenljive, procedure i funkcije, i sve one se navode u ovom odeljku biblioteke. Poenta je da su ovi pomoćni objekti *privatni* i oni se *ne vide* u programu koji koristi biblioteku.

☞ *Samo objekti deklarisani u odeljku `interface` se vide u programu koji koristi biblioteku. Objekti koji se nalaze u odeljku `implementation`, a nisu najavljeni u odeljku `interface` se **ne vide** u programu koji koristi biblioteku.*

Ukoliko je to potrebno, i privatne i javne promenljive se mogu inicijalizovati u bloku koji sledi na kraju biblioteke. Ukoliko potreba za inicijalizacijom ne postoji, ključna reč `begin` se može izostaviti i biblioteka se završava ključnom reči `end.`

☞ *Da bi prevodilac mogao da pronadje biblioteku, ime biblioteke mora biti isto kao ime fajla biblioteke!*

Program može da uveze biblioteku tako što se odmah nakon deklaracije programa navede `uses` ⟨*ime fajla biblioteke*⟩:

`program` ⟨*ime programa*⟩`;`
`uses` ⟨*ime fajla biblioteke*⟩
⟨*konstante programa*⟩
⟨*tipovi programa*⟩
⟨*promenljive programa*⟩
⟨*procedure i funkcije programa*⟩
`begin`
    ⟨*telo programa*⟩
`end.`

Kada program uveze biblioteku, sve *javne* konstante, tipovi, promenlive, procedure i funkcije nam od tog trenutka stoje na raspolaganju. Na primer, evo jedne male biblioteke koja sadrži nekoliko algoritama teorije brojeva koje smo do sada pominjali.

```
unit CeliBrojevi;

interface
  function Prost(n : longint) : boolean;
  function NZD(a, b : longint) : longint;
  function Ojler(n : longint) : longint;

implementation

  function Prost(n : longint) : boolean;
  var
    d, L : longint;
    ok : boolean;
  begin
    if n < 2 then Prost := false
    else begin
      L := trunc(sqrt(n));
      d := 3;
      ok := (n = 2) or odd(n);
      while ok and (d <= L) do begin
        ok := n mod d <> 0;
        d := d + 2
      end;
      Prost := ok
    end
  end;
```

```
  function NZD(a, b : longint) : longint;
  var
    r : longint;
  begin
    repeat
      r := a mod b; a := b; b := r
    until r = 0;
    NZD := a
  end;

  function Ojler(n : longint) : longint;
  var
    i, br : longint;
  begin
    br := 0;
    for i := 1 to n do
      if NZD(n, i) = 1 then inc(br);
    Ojler := br
  end;
end.
```

Ona mora biti smeštena u datoteku `celibrojevi.pas`. Neki drugi program može da upotrebi ovu biblioteku, na primer, ovako:

```
  program ProstiDelioci
  uses CeliBrojevi;

  var
    n, d : integer;
  begin
    writeln('Unesi n'); readln(n);
    for d := 2 to n do
      if Prost(d) then
        writeln(d)
  end.
```

9.2 Niz kao imenovani tip podataka

Neka su promenljive `a` i `b` deklarisane ovako:

```
var a : array [1 .. MaxN] of integer;
    b : array [1 .. MaxN] of integer;
```

Ako je potrebno iskopirati niz a u niz b, nema nam druge nego da to uradimo u jednoj for petlji:

```
for i := 1 to MaxN do b[i] := a[i];
```

Prirodno se postavlja pitanje: da li ovo može da se uradi brže? Zašto računar ne bi to mogao da uradi automatski? Da li može da se kaže nešto kao b := a?

I odgovor na sva ova pitanja je, naravno, DA, čim uspemo da ubedimo prevodilac da promenljive a i b zauzimaju istu količinu memorije, kao i to da je je u oba slučaja memorija organizovana na isti način. U tom slučaju se kopiranje sadržaja može uraditi jednom super-brzom mašinskom instrukcijom. Prevodilac bi mogao da bude veštački inteligentan i da sam dođe do zaključka da promenljive a i b zauzimaju istu količinu memorije koja je organizovana na isti način u oba slučaja, ali bi se time kompilacija značajno usporila. Zato Paskal pribegava jednom veoma jednostavnom i efikasnom rešenju: ako bismo postigli da *ime tipa* obe promenljive bude isto, prevodilac ne bi imao nikakvih problema da zaključi da sme da kopira jednu promenljivu u drugu bajt po bajt. Trik se zato sastoji u tome da prvo deklarišemo novi tip i da onda kažemo da su a i b promenljive tog, novog tipa:

```
type Niz = array [1 .. MaxN] of integer;
var  a, b : Niz;
```

Sada je naredba b := a legalna, a efekt je tačno onaj koji smo želeli: najbrže moguće kopiranje sadržaja promenljive a u promenljivu b. Uopšte, u Paskalu važi sledeće pravilo:

> Ukoliko promenljive a i b nisu nekog prostog tipa, onda je naredba b := a dozvoljena ako i samo ako obe promenljive imaju tip sa *istim imenom*.

Pored je dat kompletan primer koji baš i ne radi nešto jako pametno, ali pokazuje kako se definiše i koristi novi *strukturirani tip podataka*. Nagradno pitanje: Šta radi ovaj program?

```
program NeBasJakoBistarPrimer;
const
  MaxEl = 20;
type
  Niz = array [1 .. MaxEl] of integer;
var
  i    : integer;
  a, b : Niz;
begin
  for i := 1 to MaxEl do readln(a[i]);
  b := a;
  for i := 1 to MaxEl do writeln(b[i])
end.
```

Nizovi mogu da se pojave kao argumenti procedura i funkcija, s tim da moramo da vodimo računa o nekoliko stvari:

- Niz se može pojaviti kao argument procedure ili funkcije *samo kao* imenovani tip podataka; dakle, ovako nešno *ne sme* da se pojavi u programu:

```
function BrPoz(a : array [1 .. MaxEl] of real) : integer;
               ^^^^^^^^^^^ NE SME ^^^^^^^^^^^
```

- Funkcija *ne može* da vrati niz kao svoju vrednost; zato ovako nešto *ne sme* da se pojavi u programu:

```
function F(n : integer) : Niz;
                          ^^^ NE SME
```

- Iako su stringovi zapravo nizovi karaktera, njihov specijalan status kao posledicu ima da funkcija *može* da vrati string kao rezultat svog rada.

Primer. Napisati funkciju koja utvrđuje koliko pozitivnih brojeva ima u datom nizu brojeva.

```
const
  MaxEl = 100;
type
  Niz = array [1 .. MaxEl] of real;

function BrPoz(a : Niz) : integer;
var
  i, n : integer;
begin
  n := 0;
  for i := 1 to MaxEl do
    if a[i] > 0 then
      n := n + 1;
  BrPoz := n
end;
```

Ako niz treba preneti u proceduru ili funkciju, onda je kao u primeru na početku potrebno definisati odgovarajući tip podataka. Ako želimo da vratimo niz kao rezultat nekog računanja, onda moramo koristiti proceduru u koju se niz prosleđuje kao ulazno-izlazni argument, na primer ovako:

```
procedure P(n : integer; var REZ : Niz);  { ovo je OK }
```

Zadaci.

9.1. *Norma 1* niza brojeva $(a_1, a_2, \ldots, a_n)$ je sledeći broj $|a_1| + |a_2| + \ldots + |a_n|$, *norma 2* istog niza je sledeći broj: $\sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$, a *norma* ∞ je broj: $\max\{|a_1|, |a_2|, \ldots, |a_n|\}$. Napisati funkcije `Norma1`, `Norma2` i `NormaInf` koje računaju odgovarajuću normu niza brojeva, pri čemu je tip podataka niza dat sa

```
const
  MaxEl = 1000;
type
  Niz = array [1 .. MaxEl] of real;
```

9.2. Napisati proceduru `VelikaSlova(var s : NizSlova; n : integer)` koja u delu niza `s` od prvog do *n*-tog mesta mala slova konvertuje u velika, dok ostale simbole ostavlja neizmenjene. Pri tome je:

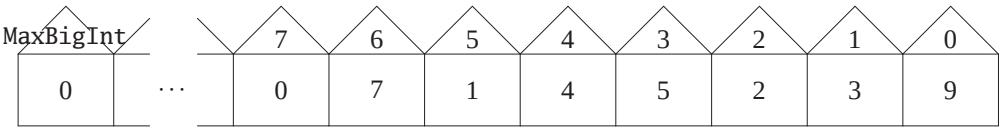
```
const
  MaxEl = 1000;
type
  NizSlova = array [1 .. MaxEl] of char;
```

9.3 Veliki brojevi

Svaki programski jezik ima ograničenje na maksimalan broj cifara celih brojeva sa kojima zna da računa. Ako želimo da radimo sa celim brojevima koji su veći od onih koje jezik inicijalno podržava, moramo sami da napišemo odgovarajuće procedure. Veliki broj ćemo pamtiti kao niz cifara:

```
const
  MaxBigInt = 3000; {max broj cifara za BigInt}
type
  BigInt = array [0 .. MaxBigInt] of integer;
```

pri čemu usvajamo konvenciju da u kućici redni broj k stoji cifra koja odgovara stepenu 10^k. Na primer, broj 7 145 239 ćemo predstaviti sledećim nizom:



Procedura `ReadBigInt` učitava veliki broj i uz put brine o tome da na početak dopiše dovoljnu količinu nula.

```
procedure ReadBigInt(var a : BigInt);
var
  s : string;
  i : integer;
begin
  readln(s);
  for i := 1 to length(s) do
    a[length(s) - i] := ord(s[i]) - ord('0');
  for i := length(s) to MaxBigInt do a[i] := 0
end;
```

Procedura `IspisiVBroj` ispisuje veliki broj uz preskakanje vodećih nula:

```
function LeadingPos(a : BigInt) : integer;
var
  i : integer;
begin
  i := MaxBigInt;
  while (a[i] = 0) and (i > 0) do dec(i);
  LeadingPos := i
end;

procedure WriteBigInt(a : BigInt);
var
  i : integer;
begin
  i := LeadingPos(a);
  while i >= 0 do begin
    write(a[i]);
    dec(i)
  end
end;
```

Sledi procedura koja sabira dva velika broja tako što simulira sabiranje "peške". Ukoliko rezultat ne može da stane u predviđeni broj mesta promenljiva `BigIntOverflow` dobija vrednost `true`.

```
var
  BigIntOverflow : boolean;

procedure AddBigInt(var c : BigInt; a, b : BigInt); { c := a + b }
var
```

```
  i, rez, prenos : integer;
begin
  prenos := 0;
  for i := 0 to MaxBigInt do begin
    rez := prenos + a[i] + b[i];
    c[i] := rez mod 10;
    prenos := rez div 10
  end;
  if prenos > 0 then BigIntOverflow := true
end;
```

Pogledajmo sada proceduru koja oduzima dva velika broja simulirajući oduzimanje "peške". Pretpostavljamo da prvi broj nije manji od drugog jer ne radimo sa negativnim brojevima.

```
procedure SubtractBigInt(var c : BigInt; a, b : BigInt);
{ c := a - b;  USLOV: a >= b }
var
  i : integer;
begin
  for i := 0 to MaxBigInt - 1 do begin
    if a[i] < b[i] then begin
      a[i] := a[i] + 10;
      a[i+1] := a[i+1] - 1
    end;
    c[i] := a[i] - b[i]
  end;
  c[MaxBigInt] := a[MaxBigInt] - b[MaxBigInt]
end;
```

Na kraju pokazujemo proceduru koja poredi dva velika broja. Ona vraća -1 ako je prvi broj manji, 0 ako su jednaki, odnosno 1 ako je prvi broj veći.

```
function CompareBigInt(a, b : BigInt) : integer;
var
  i : integer;
begin
  i := MaxBigInt;
  while (a[i] = b[i]) and (i > 0) do dec(i);
  if a[i] = b[i] then CompareBigInt := 0
  else if a[i] < b[i] then CompareBigInt := -1
  else CompareBigInt := 1
end;
```

Zadaci.

9.3. Napisati proceduru `AssignBigInt(var a : BigInt; n : longint);` koja "mali" prirodan broj `n` upisuje u veliki broj `a` i tako simulira naredbu `a := n`.

9.4. Napisati Paskal program koji od korisnika učitava prirodan broj n, potom n velikih brojeva i potom računa i ispisuje njihov zbir.

9.5. Napisati proceduru `ShortMulBigInt(var a : BigInt; n : integer);` koja množi veliki broj `a` "malim" brojem `n`.

9.6. Napisati program koji računa i ispisuje 1000-ti Fibonačijev broj (on ima manje od 300 cifara).

9.7. Napisati Paskal program koji od korisnika učitava n i računa i ispisuje n-ti član niza koji je dat sledećom rekurentnom vezom: $a_0 = 0$, $a_1 = 13$, $a_n = 4a_{n-1} + 9a_{n-2}$.

9.8. Napisati Paskal program koji od korisnika učitava n i računa i ispisuje n-ti član niza koji je dat sledećom rekurentnom vezom: $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, $a_n = n^2 a_{n-1} - 2a_{n-2} - 3a_{n-3}$.

9.9. Napisati Paskal program koji od korisnika učitava n i potom računa i štampa $n!$ koristeći velike brojeve.

9.10. Napisati Paskal program koji od korisnika učitava n i potom računa i štampa $\sum_{k=1}^{n} k!$ koristeći velike brojeve.

9.11. Napisati proceduru `MulBigInt(var c : BigInt; a, b : BigInt);` koja množi dva velika broja.

9.12. Napisati Paskal program koji od korisnika učitava n potom n velikih brojeva $a_1, \ldots, a_n$ i računa i štampa vrednost izraza $a_1^2 + \ldots + a_n^2$.

9.13. Niz brojeva zadat je na sledeći način: $L_0 = 2$, $L_1 = 5$, $L_n = 3L_{n-1} - 2L_{n-2}$. Napisati Paskal program koji računa L_{1000} ako se zna da taj broj ima manje od 400 cifara.

9.14. Napisati Paskal program koji od korisnika učitava n i računa i ispisuje n-ti član niza koji je dat sledećom rekurentnom vezom: $a_0 = 1$, $a_1 = 5$,

$$a_n = \begin{cases} a_{n-1}a_{n-2}, & n \text{ neparno} \\ (n-1)a_{n-1} + a_{n-2}, & \text{inače.} \end{cases}$$

9.15. Napisati proceduru

```
DivModBigInt(x, y : BigInt; var q, r : BigInt);;
```

koja deli veliki broj x velikim brojem y. Argument q sadrži količnik, a argument r ostatak celobrojnog deljenja.

9.16. Za prirodan broj n kažemo da je pseudoprost ako nije prost, ali $n|2^n - 2$. Napisati Paskal program koji od korisnika učitava pozitivan ceo broj n i utvrđuje da li je to pseudoprost broj. Obratiti pažnju na to da 2^n može biti veći od maxlongint.

9.4 Unit BInt

U ovom odeljku ćemo pokazati kompletnu biblioteku za rad sa velikim brojevima. Promenljiva BigIntOverflow će dobiti vrednost true ako je prilikom sabiranja ili množenja velikih brojeva došlo do prekoračenja maksimalne dužine broja.

```
unit BInt;  {Biblioteka za rad sa nenegativnim velikim brojevima}

interface
  const
    MaxBigInt = 3000; {max broj cifara za BigInt}
  type
    BigInt = array [0 .. MaxBigInt] of integer;
  var
    BigIntOverflow : boolean;

  procedure ReadBigInt(var a : BigInt);
  procedure WriteBigInt(a : BigInt);
  procedure AssignBigInt(var a : BigInt; n : longint);
    { a := n }
  procedure AddBigInt(var c : BigInt; a, b : BigInt);
    { c := a + b }
  function  CompareBigInt(a, b : BigInt) : integer;
    {-1 za a < b, 0 za a = b, 1 za a > b}
  function  IsZeroBigInt(a : BigInt) : boolean;
    { a = 0 ?}
  function  IsOneBigInt(a : BigInt) : boolean;
    { a = 1 ?}
  procedure SubtractBigInt(var c : BigInt; a, b : BigInt);
    { c := a - b;  USLOV: a >= b }
  procedure ShortMulBigInt(var a : BigInt; n : integer);
    {a := a * n}
  procedure MulBigInt(var c : BigInt; a, b : BigInt);
    {c := a * b}
```

```
  procedure DivModBigInt(x, y : BigInt; var q, r : BigInt);
    { USLOV: y <> 0 }

implementation

  function LeadingPos(a : BigInt) : integer;
    { privatna funkcija; ne vidi se izvan biblioteke }
    { vraca prvu cifru sleva koja nije nula }
  var
    i : integer;
  begin
    i := MaxBigInt;
    while (a[i] = 0) and (i > 0) do dec(i);
    LeadingPos := i
  end;

  procedure Rearrange(var a : BigInt);
    { privatna funkcija; ne vidi se izvan biblioteke }
    { sredjuje prenose }
  var
    i : integer;
  begin
    for i := 0 to MaxBigInt - 1 do begin
      a[i+1] := a[i+1] + a[i] div 10;
      a[i] := a[i] mod 10
    end;
    if a[MaxBigInt] > 9 then begin
      BigIntOverflow := true;
      a[MaxBigInt] := a[MaxBigInt] mod 10
    end
  end;

  procedure ReadBigInt(var a : BigInt);
  var
    s : string;
    i : integer;
  begin
    readln(s);
    for i := 1 to length(s) do
      a[length(s) - i] := ord(s[i]) - ord('0');
    for i := length(s) to MaxBigInt do a[i] := 0
  end;
```

```
procedure WriteBigInt(a : BigInt);
var
  i : integer;
begin
  i := LeadingPos(a);
  while i >= 0 do begin
    write(a[i]);
    dec(i)
  end
end;

procedure AssignBigInt(var a : BigInt; n : longint);
  { a := n }
var
  i : integer;
begin
  i := 0;
  while n > 0 do begin
    a[i] := n mod 10;
    n := n div 10;
    inc(i)
  end;
  while i <= MaxBigInt do begin
    a[i] := 0;
    inc(i)
  end
end;

procedure AddBigInt(var c : BigInt; a, b : BigInt);
  { c := a + b }
var
  i, rez, prenos : integer;
begin
  prenos := 0;
  for i := 0 to MaxBigInt do begin
    rez := prenos + a[i] + b[i];
    c[i] := rez mod 10;
    prenos := rez div 10
  end;
  if prenos > 0 then BigIntOverflow := true
end;
```

```
function CompareBigInt(a, b : BigInt) : integer;
  {-1 za a < b, 0 za a = b, 1 za a > b}
var
  i : integer;
begin
  i := MaxBigInt;
  while (a[i] = b[i]) and (i > 0) do dec(i);
  if a[i] = b[i] then CompareBigInt := 0
  else if a[i] < b[i] then CompareBigInt := -1
  else CompareBigInt := 1
end;

function IsZeroBigInt(a : BigInt) : boolean;
  { a = 0 ?}
begin
  IsZeroBigInt := (LeadingPos(a) = 0) and (a[0] = 0)
end;

function IsOneBigInt(a : BigInt) : boolean;
  { a = 1 ?}
begin
  IsOneBigInt := (LeadingPos(a) = 0) and (a[0] = 1)
end;

procedure SubtractBigInt(var c : BigInt; a, b : BigInt);
  { c := a - b;  USLOV: a >= b }
var
  i : integer;
begin
  for i := 0 to MaxBigInt - 1 do begin
    if a[i] < b[i] then begin
      a[i] := a[i] + 10;
      a[i+1] := a[i+1] - 1
    end;
    c[i] := a[i] - b[i]
  end;
  c[MaxBigInt] := a[MaxBigInt] - b[MaxBigInt]
end;
```

```
procedure ShortMulBigInt(var a : BigInt; n : integer);
  {a := a * n}
var
  i : integer;
begin
  for i := 0 to LeadingPos(a) do
    a[i] := a[i] * n;
  Rearrange(a)
end;

procedure MulBigInt(var c : BigInt; a, b : BigInt);
  {c := a * b}
var
  i, j : integer;
begin
  AssignBigInt(c, 0);
  for i := 0 to LeadingPos(a) do
    for j := 0 to LeadingPos(b) do
      if i + j > MaxBigInt then
        BigIntOverflow := true
      else
        c[i + j] := c[i + j] + a[i] * b[j];
  if not BigIntOverflow then
    Rearrange(c)
end;

procedure DivModBigInt(x, y : BigInt; var q, r : BigInt);
  { USLOV: y <> 0 }
var
  cmp, i, j, k, p, xN, yN, qN, temp, prenos, rez : integer;
  t : BigInt;
  ok : boolean;
begin
  cmp := CompareBigInt(x, y);
  if IsZeroBigInt(x) then
    begin
      AssignBigInt(q, 0);
      AssignBigInt(r, 0)
    end
  else if IsOneBigInt(y) then
    begin
      q := x;
      AssignBigInt(r, 0)
    end
  else if cmp = 0 { x = y } then
```

```
    begin
      AssignBigInt(q, 1);
      AssignBigInt(r, 0)
    end
  else if cmp = -1 { x < y } then
    begin
      AssignBigInt(q, 0);
      r := x
    end
  else begin
    AssignBigInt(q, 0);
    xN := LeadingPos(x); yN := LeadingPos(y);
    k := xN - yN;

    { y := y * 10^k }
    if k > 0 then begin
      for i := yN downto 0 do y[i + k] := y[i];
      for i := 0 to k - 1 do y[i] := 0;
      yN := yN + k
    end;

    qN := -1;
    for j := 0 to k do begin
      ok := false;
      p := x[xN] div y[xN]; { NB. xN = yN }
      while not ok and (p > 0) do begin
        { t := y * p, ali na specijalan nacin }
        AssignBigInt(t, 0);
        prenos := 0;
        for i := 0 to yN - 1 do begin
          rez := prenos + y[i] * p;
          t[i] := rez mod 10;
          prenos := rez div 10
        end;
        t[yN] := prenos + y[yN] * p;

        if CompareBigInt(x, t) = -1 then
          dec(p)
        else begin
          ok := true;
          SubtractBigInt(x, x, t)
        end
      end;
```

```
        { dodaj cifru p u niz q }
        inc(qN);
        q[qN] := p;

        { skrati x }
        x[xN - 1] := 10 * x[xN] + x[xN - 1];
        x[xN] := 0;
        dec(xN);

        { y := y div 10 }
        for i := 0 to yN - 1 do
          y[i] := y[i + 1];
        y[yN] := 0;
        dec(yN)
      end {for};

      { okreni q }
      for i := 0 to qN div 2 do begin
        temp := q[i];
        q[i] := q[qN - i];
        q[qN - i] := temp
      end;

      r := x;
    end {if}
  end;

begin
  BigIntOverflow := false
end.
```

Pogledajmo na primeru programa koji računa faktorijel celog broja (a koji vrlo lako može da izađe iz opsega tipa `longint`) kako se koristi ova biblioteka.

```
program Faktorijel;
uses BInt;

var
  i, n : integer;
  fakt : BigInt;
begin
  writeln('Unesi n'); readln(n);
  AssignBigInt(fakt, 1); { fakt := 1 }
  for i := 2 to n do
    ShortMulBigInt(fakt, i); { fakt := fakt * i }
```

```
  if BigIntOverflow then
    writeln('Doslo je do prekoracenja tokom racunanja')
  else begin
    WriteBigInt(fakt);
    writeln
  end
end.
```

9.5 Sistematizacija

Svaka Paskal biblioteka ima sledeći oblik:

```
unit ⟨ime biblioteke⟩;
interface
    ⟨javne konstante⟩
    ⟨javni tipovi⟩
    ⟨javne promenljive⟩
    ⟨zaglavlja javnih procedura i funkcija⟩
implementation
    ⟨privatne konstante⟩
    ⟨privatni tipovi⟩
    ⟨privatne promenljive⟩
    ⟨javne i privatne procedure i funkcije⟩
begin
    ⟨inicijalizacija javnih promenljivih⟩
end.
```

Program može da uveze biblioteku tako što se odmah nakon deklaracije programa navede `uses` ⟨*ime fajla biblioteke*⟩:

```
program ⟨ime programa⟩;
uses ⟨ime fajla biblioteke⟩
⟨konstante programa⟩
⟨tipovi programa⟩
⟨promenljive programa⟩
⟨procedure i funkcije programa⟩
begin
    ⟨telo programa⟩
end.
```

Da bi prevodilac mogao da pronadje biblioteku, ime biblioteke mora biti isto kao ime fajla biblioteke!

Deklaracija promenljive nizovnog tipa ima sledeći oblik:

```
var
    ⟨ime⟩ : array [⟨indeksi⟩] of ⟨proizvoljan_tip⟩;
```

Pri tome ⟨*indeksi*⟩ označava način indeksiranja niza:

```
⟨indeksi⟩  ≡  ⟨konstanta⟩ .. ⟨konstanta⟩
           |   integer | boolean | char
```

Niz se može definisati i kao poseban tip:

```
type
    ⟨novo_ime_tipa⟩ = array [⟨indeksi⟩] of ⟨proizvoljan_tip⟩;
```

Za promenljive čiji tipovi nisu prosti, ali imaju isto ime dozvoljeno je kopiranje sadržaja jedne od njih u drugu:

```
type
    ⟨novo_ime_tipa⟩ = ⟨neka_definicija_tipa⟩;
var
    ⟨var1⟩, ⟨var2⟩ : ⟨novo_ime_tipa⟩;
begin
    ...
    ⟨var2⟩ := ⟨var1⟩;
    ...
end.
```