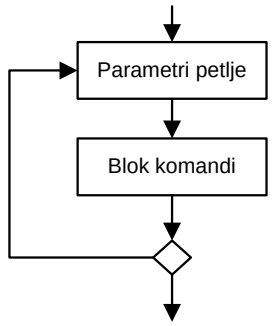
Naredbe ponavljanja

U većini programa se javljaju situacije kada je potrebno neku naredbu ili grupu naredbi izvršiti više puta. Ukoliko je naredbu potrebno izvršiti konačan i mali broj puta, problem je moguće razrešiti i korišćenjem linijskih struktura, tako što bi se naredba jednostavno ponovila određeni broj puta uzastopno. Međutim, može se desiti da je naredbu potrebno ponoviti veliki broj puta, a veoma često je taj broj promenljiv u zavisnosti od izvršenja ostatka programa. U takvim slučajevima nije moguće iskoristiti linijsku strukturu, već je neophodno uvesti takozvane **ciklične strukture**. Ciklične strukure omogućavaju izvršavanje jedne ili više naredbi određeni broj puta, pri čemu broj ponavljanja može biti definisan prirodnim brojem ili uslovom koji određuje kada se ponavljanje prekida.

Napomenimo i to da se ciklične strukture vrlo često nazivaju i **ciklusima** ili **petljama**.

FOR – ciklus

Naredba **for** umogućava bezuslovno ponavljanje nekog dela programa određeni broj puta. Na taj način moguće je ostvariti cikličnu strukturu prikazanu na slici:



Slika ### Algoritamska šema for ciklusa

U okviru **for** petlje neophodno je definisati brojačku promenljivu (brojač) koja će se u svakom prolasku kroz petlju (iteraciji) uvećavati ili umanjivati za jedan u zavisnosti od toga da li je u okviru **for** naredbe navedena rezervisana reč **to** (za uvećavanje) ili **downto** (za umanjivanje). Pored toga potrebno je navesti i početnu i krajnju vrednost brojačke promenljive. Naredbe unutar **for** petlje se izvršavaju za svaku vrednost brojačke promenljive, sve dok ona ne dostigne krajnju vrednost.

Sintaksa

```
for <brojacka_promenljiva> := <pocetna_vrednost> to <krajnja_vrednost> do
<naredba>
```

u slučaju da se brojačka promenljiva uvećava, odnosno

```
for <brojacka_promenljiva> := <pocetna_vrednost> downto <krajnja_vrednost> do
<naredba>
```

u slučaju kada se brojačka promenljiva umanjuje.

Upravljačka promenljiva može biti integer, boolean ili char tipa. U Pascal-u nije moguće definisati za koliko se uvećava ili umanjuje brojačka promenljiva. Korak promene je uvek 1 (u slučaju **to**), odnosno -1 (u slučaju **downto**). Početna i krajnja vrednost se izračunavaju na početku petlje i ne mogu se menjati tokom njenog izvršavanja.

Primer 1

```
for i:=1 to 5 do write('A');
```

Rezultat ovog primera je 5 puta ispisano slovo "A" na ekranu na sledeći način:

AAAAA

Primer 2

```
for i:=1 to 10 do write(i:5);
```

Nakon izvršenja ove petlje na ekranu će biti ispisani brojevi od 1 do 10:

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

Ukoliko bismo želeli da brojevi budu ispisani obrnutim redom, onda bi petlja trebala da izgleda ovako:

```
for i:=10 downto 1 do write(i:5);
```

a rezultat bi bio niz brojeva ispisan na sledeći način:

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

Primer 3

Sledeći primer pokazuje da početna i krajnja vrednost brojačke promenljive mogu biti i izrazi.

```
for i:=-4+6 to 4*3-5 do write(i:5);
```

Rezultat prethodnog primera je sledeći niz brojeva ispisan na ekranu:

```
    2    3    4    5    6    7
```

Primer 4

Napisati program koji za *n* zadatih vrednosti poluprečnika *r* izračunava obim i površinu kruga i prikazuje ih na ekranu.

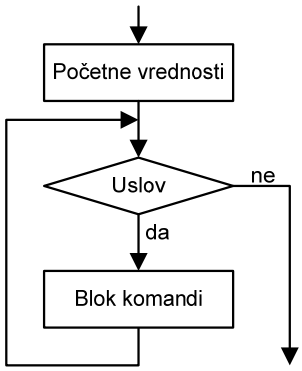
```
program Krug;
const   PI=3.14;
var  i,n:integer;
     r,O,P:real;
begin
   writeln('Unesite koliko izracunavanja zelite:');
   readln(n);

   for i:=1 to n do
      begin
         writeln('Unesite poluprecnik:');
         readln(r);

         O:=2.0*r*PI;
         P:=sqr(r)*PI;
         writeln('Obim      ','Povrsina');
         writeln(O:10:2,P:10:2);
      end;
end.
```


WHILE – ciklus

Da bi se koristila naredba **for** neophodno je znati tačan broj ponavljanja pre početka izvršavanja petlje. Pored toga, za vreme izvršavanja petlje nije moguće menjati početnu i krajnju vrednost brojačke promenljive. Međutim, veoma često je slučaj da određeni deo programa želimo ponavljati sve dok je neki uslov ispunjen. Na taj način iteracije se mogu ponavljati manji ili veći broj puta u zavisnosti od rezultata izvršavanja naredbi unutar petlje.

Jedna od naredbi koja omogućava realizaciju ovakvih ciklusa je naredba **while**. Osnovna karakteristika **while** petlje je da je to **petlja sa preduslovom**, što znači da se ispunjenost uslova proverava pre

izvršavanja svake iteracije. Posledica toga je da je moguće da se naredbe unutar petlje ne izvrše ni jednom, ukoliko uslov u startu nije zadovoljen. Šematski prikaz ciklusa sa preduslovom prikazan je na slici:



Slika ### Šematski prikaz ciklusa sa preduslovom

Naredba **while** omogućava ponavljanje određenog dela programa sve **dok je navedeni uslov ispunjen**. Na početku svake iteracije proverava se da li je uslov ispunjen, tj. da li je vrednost logičkog iskaza **tačno** (**true**), a zatim, ukoliko je jeste, izvršava se jedna naredba ili blok naredbi navedenih unutar komande **while**. Nakon izvršenja naredbi unutar **while** petlje otpočinje nova iteracija i nova provera uslova. U trenutku kada na početku neke iteracije vrednost logičkog izraza postane **netačno** (**false**), prekida se izvršavanje petlje i nastavlja se izvršavanje ostatka programa.

Sintaksa

```
while <logicki_izraz> do <naredba>;
```

NAPOMENA: **While** petlja mora biti tako napisana da garantuje da će u konačnom broju iteracija navedeni logički izraz postati netačan (**false**). Na taj način obezbeđuje se mehanizam izlaska iz petlje nakon konačnog broja iteracija. Ukoliko logički izraz nikada ne bi dobio vrednost **false** došlo bi do beskonačnog broja ponavljanja (tzv. **mrtva petlja**), odnosno do blokade izvršenja ostatka programa.

Primer 1

```
i:=2;
while i<10 do
   begin
      write(i:5);
      i:=i+2;
   end;
```

Prethodni primer štampa na ekranu sve jednocifrene parne brojeve na sledeći način:

```
    2    4    6    8
```

Primer 2

Ukoliko bismo prethodni primer promenili tako da izgleda ovako:

```
i:=20;
while i<10 do
   begin
      write(i:5);
      i:=i+2;
   end;
```

u tom slučaju se naredbe unutar **while** petlje ne bi izvršile ni jednom, iz razloga što uslov *i<10* nije ispunjen već na početku prve iteracije.

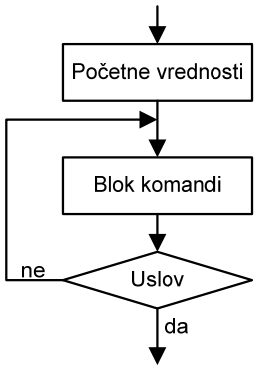
Primer 3

Napisati program koji zadatom celom broju *n* uklanja sve nule sa desne strane. Na primer, ukoliko unesemo broj 41000, rezultat treba da bude 41.

```
program Nule;
var  n:integer;
begin
   writeln('Unesite broj n:');
   readln(n);

   while n mod 10 = 0 do
      n:=n div 10;

   writeln('Dobijeni broj je ',n);
end.
```

REPEAT – ciklus

Za razliku od **while** petlje, naredba **repeat** omogućava realizaciju ciklusa sa **postuslovom**, kao što je prikazano na slici:



Slika ### Šematski prikaz postuslovnog ciklusa

Ciklus sa postuslovom podrazumeva to da se uslov za dalje izvršavanje petlje proverava na kraju svake iteracije. Na ovaj način naredbe unutar petlje će biti izvršene bar jednom, bez obzira na to da li je uslov bio ispunjen pre ulaska u petlju.

Naredba **repeat** omogućava ponavljanje određenog dela programa sve **dok navedeni uslov nije ispunjen**. To praktično znači da je (za razliku od **while** ciklusa) navedeni uslov ustvari "uslov za izlazak iz petlje". U slučaju **repeat** petlje, izvršava se jedna naredba ili blok naredbi navedenih unutar komande **repeat**. Nakon izvršenja ovih naredbi vrši se provera uslova navedenog iza rezervisane reči **until**. Ukoliko je vrednost logičkog izraza **netačno** (**false**), kreće se u izvršavanje sledeće iteracije. U suprotnom prekida se izvršavanje petlje i nastavlja se izvršavanje ostatka programa.

Sintaksa

```
repeat
   <naredba_1>;
   [naredba_2];
      ...
   [naredba_n];
until <logicki_izraz>;
```

Primetimo i to da u slučaju **repeat** naredbe, rezervisane reči **repeat** i **until** istovremeno predstavljaju i graničnike petlje, tako da je upotreba rezervisanih reči **begin** i **end** za označavanje bloka naredbi nepotrebna.

NAPOMENA: **Repeat** petlja mora biti tako napisana da garantuje da će u konačnom broju iteracija navedeni logički izraz postati tačan (**true**). Na taj način obezbeđuje se mehanizam izlaska iz petlje nakon konačnog broja iteracija. Ukoliko logički izraz nikada ne bi dobio vrednost **true** došlo bi do beskonačnog broja ponavljanja (tzv. **mrtva petlja**), odnosno do blokade izvršenja ostatka programa.

Primer 1

```
i:=2;
repeat
   write(i:5);
   i:=i+2;
until i>=10;
```

Prethodni primer štampa na ekranu sve jednocifrene parne brojeve, na sledeći način:

```
    2    4    6    8
```

Upoređivanjem ovog primera sa sličnim primerom realizovanim pomoću **while** petlje, može se jasno uočiti razlika u načinu definisanja uslova. U slučaju **while** naredbe petlja se izvršava "dok je *i*<10", a u slučaju **repeat** petlje "dok je *i* veće ili jednako 10", odnosno "dok *i* nije manje od 10".

Primer 2

Napisati program koji izračunava broj π sa zadatom tačnošću ε, korišćenjem formule:

$$\pi = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots \right)$$

```
program BrojPI;
var  pi,epsilon,delilac,znak,sabirak:real;
begin
   writeln('Unesite zeljenu tacnost:');
   readln(epsilon);

   pi:=1.0;
   delilac:=1.0;
   znak:=1.0;

   repeat
      delilac:=delilac+2.0;
      znak:=-znak;
      sabirak:=1.0/delilac;
      pi:=pi+znak*sabirak;
   until abs(sabirak)<epsilon;

   pi:=4.0*pi;
   writeln('PI=',pi);
end.
```

Naredbe BREAK i CONTINUE

Pojedini Pascal kompajleri (kao što je Turbo Pascal 7.0 i noviji) podržavaju korišćenje komandi za prevremeni prekid čitave petlje ili samo iteracije koja se trenutno izvršava. Obe ove komande su primenjive na **for**, **while** i **repeat** petlje.

Naredba BREAK

Naredba **break** se koristi za bezuslovno prekidanje izvršavanja čitave petlje. Nakon izvršenja ove komande prekida se izvršavanje naredbi unutar petlje i nastavlja se sa izvršavanjem ostatka programa.

Primer

Napisati program koji za *n* realnih brojeva sa ulaza izračunava funkciju $f(x)=\frac{1}{x}$. U slučaju da je na ulazu uneta nula, obavestiti korisnika da deljenje nulom nije moguće i prekinuti izvršavanje programa.

```
program Deljenje;
var  n,i:integer;
     x,f:real;
begin
   writeln('Unesite koliko brojeva zelite:');
   readln(n);

   for i:=1 to n do
      begin
         writeln('Unestite x:');
         readln(x);

         if x<>0.0 then
            begin
               f:=1.0/x;
               writeln('f(x)=',f:10:4);
            end
         else
            begin
               writeln('Deljenje nulom nije moguce.');
               break;
            end;
      end;
end.
```

Naredba CONTINUE

Naredba **continue** se koristi za bezuslovno prekidanje izvršavanje trenutne i otpočinjanje sledeće iteracije. Time se praktično postiže preskakanje naredbi koje slede u nastavku iza komande **continue**, a naredna iteracija se normalno izvršava ukoliko su zadovoljeni uslovi definisani petljom.