Funkcije i procedure

Kompjuterski programi sadrže nizove instrukcija koje za cilj imaju izvršavanje određenog zadatka, što u opštem slučaju podrazumeva preuzimanje ulaznih veličina i njihovu obradu u cilju dobijanja određenih rezultata. Imajući ovo u vidu, kompjuterske programe možemo uporediti sa preduzećima koja nabavljaju ulazne sirovine i njihovom obradom dolaze do konačnih proizvoda.

U slučaju malih zanatskih radionica nabavku materijala, obradu i prodaju proizvoda može obavljati samo jedan čovek. Međutim, u velikim preduzećima je neophodno napraviti organizacione jedinice specijalizovane za obavljanje samo određenih zadataka. Na taj način vrši se podela zaduženja i odgovornosti za izvršavanje određenih zadataka, čime se znatno olakšava organizacija posla u preduzeću. Pored toga, zaposleni su specijalizovani za određene zadatke, što znatno povećava efikasnost. Takođe, u slučaju potrebe za promenama u određenim segmentima proizvodnje, ne narušava se funkcionalnost ostalih celina, već se izmene vrše samo u sektoru koji je odgovoran za izvršenje određenog zadatka.

Prilikom rešavanja manjih problema korišćenjem računara, svi zadaci se mogu izvršiti u okviru glavnog programa, baš kao što jedan radnik može obaviti sve zadatke u zanatskoj radionici. Međutim, sa povećanjem složenosti kompjuterskih programa, neophodno je uvesti organizacione koncepte slične onima koji se koriste u velikim preduzećima. Iz tog razloga, potrebno je program podeliti na odgovarajuće logičke i funkcionalne celine, takozvane **potprograme**. Potprogrami omogućavaju organizovanje određenog broja instrukcija u celine koje obavljaju određene zadatak i na koje se možemo pozivati više puta u toku izvršavanja programa.

Korišćenje potprograma donosi niz prednosti od kojih možemo izdvojiti sledeće:

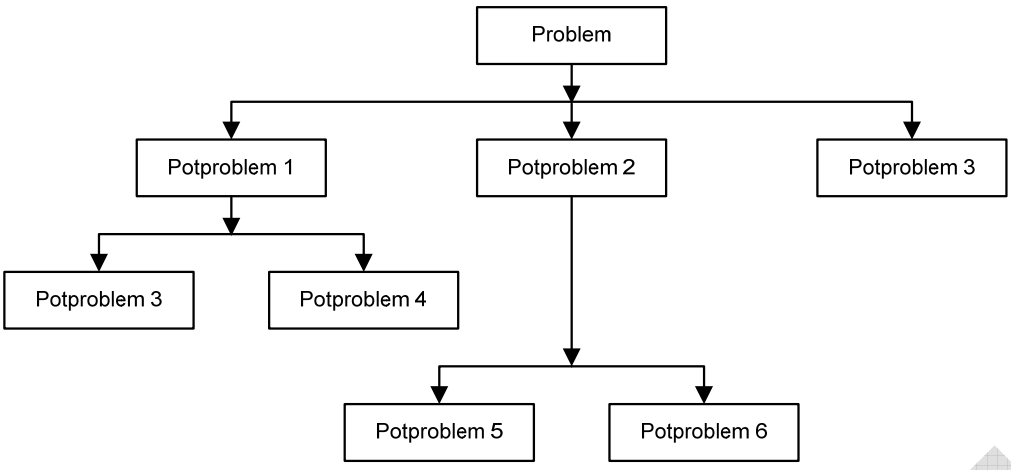
- Omogućava koncentrisanje na izvršavanje samo određenog zadatka
- Različiti članovi razvojnog tima mogu razvijati različite potprograme koji se kasnije sklapaju u jednu celinu
- Omogućava izvršavanje istog zadatka na više mesta u programu jednostavnim pozivanjem odgovarajućeg potprograma

Realizacija potprograma u Pascal-u

U programskom jeziku Pascal potprograme je moguće realizovati korišćenjem **funkcija** i **procedura**. **Funkcija** je samostalan deo programa koji obavlja određeni zadatak i preko svog naziva i liste parametara, delu programa iz koga je pozvana vraća odgovarajuće rezultate. Svaka funkcija ima jedinstveni naziv preko koga se može pozvati proizvoljan broj puta iz bilo kog dela programa u cilju izvršenja tog zadatka. **Procedura** je potprogram sličan funkciji sa tom razlikom što procedura preko svog imena ne vraća nikakvu vrednost delu programa iz koga je pozvana.

Pored toga što se korišćenjem funkcija i procedura izbegava nepotrebno ponavljanje delova koda, pa samim tim povećava i čitljivost programa, funkcije i procedure omogućavaju realizaciju programiranja "odozgo na dole". Ovaj metod podrazumeva razbijanje problema na manje "potprobleme", a zatim i novodobijenih "potproblema" na još manje celine, sve do trenutka dok se glavni problem ne svede na rešavanje elementarnih problema. Prateći ovaj princip većina programa se može razbiti na određeni broj potproblema kao što je prikazano na slici:

FINALNA VERZIJA



Slika ### Šematski prikaz metoda "odozgo na dole"

Sintaksa

```
function <naziv>[(lista parametara)]:<tip_funkcije>;
[deklaracija_lokalnih_promenljivih]
<blok_naredbi>

procedure <naziv>[(lista parametara)];
[deklaracija_lokalnih_promenljivih]
<blok_naredbi>
```

Svaka funkcija ili procedura mora imati zaglavlje, koje sadrži jedinstveni naziv naveden odmah iza rezervisane reči **function**, odnosno **procedure**. Iza naziva potprograma sledi lista parametara unutar oblih zagrada. Za svaki parametar potprograma mora biti naveden i njegov tip, na sličan način kao i kod deklaracija promenljivih naredbom **var**. Korišćenje ili izostavljanje naredbe **var** u okviru liste parametara ima specifičnu funkciju, o čemu će kasnije biti reči.

Nakon definisanja zaglavlja potprograma sledi blok za deklarisanje lokalnih promenljivih. Deklaracija lokalnih promenljivih se vrši na potpuno isti način kao i deklarisanje promenljivih unutar glavnog programa, korišćenjem naredbe **var**.

Iza bloka za deklaraciju lokalnih promenljivih sledi blok naredbi, odnosno telo funkcije ili procedure. Kao i svaki drugi blok naredbi, telo potprograma se sastoji od niza komandi ograničenih rezervisanim rečima **begin** i **end**. Iza rezervisane reči **end** u ovom slučaju se piše znak **;**.

S obzirom na činjenicu da funkcija mora vratiti odgovarajuću vrednost, neophodno je da unutar tela funkcije postoji bar jedna naredba dodele koja izračunatu vrednost dodeljuje nazivu funkcije.

Svaki potprogram može biti pozvan iz glavnog programa ili drugog potprograma navođenjem njegovog imena i liste parametara unutar oblih zagrada. Prilikom poziva funkcija potrebno je vrednost koju funkcija vraća dodeliti odgovarajućoj promenljivoj korišćenjem operatora dodele, ili je direktno iskoristiti unutar nekog izraza ili naredbe.

Primer 1

Napisati program koji izračunava i štampa rastojanje između dve tačke čije su koordinate date na ulazu.

```
program Rastojanje;
var  x1,y1,x2,y2,r:real;

   function rast(x1,y1,x2,y2:real):real;
   begin
      rast:=sqrt(sqr(x2-x1)+sqr(y2-y1));
   end;
```

RADNA VERZIJA

```
   procedure stampaj(d:real);
   begin
      writeln('Rastojanje izmedju tacaka je ',d:10:4);
   end;

begin
   writeln('Unesite koordinate prve tacke:');
   read(x1,y1);

   writeln('Unesite koordinate druge tacke:');
   read(x2,y2);

   r:=rast(x1,y1,x2,y2);

   stampaj(r);
end.
```

U prethodnom primeru definisana su dva potprograma. Prvi potprogram je funkcija koja na osnovu koordinata dve tačke izračunava rastojanje između njih. Drugi potprogram je procedura koja vrši štampanje rezultata uz odgovarajući komentar.

Nakon unosa koordinata tačaka glavni program poziva funkciju *rast* i šalje joj koordinate tačaka. Kada funkcija završi izračunavanje, ona glavnom programu vraća rastojanje između navedenih tačaka. Glavni program, zatim, dobijenu vrednost dodeljuje promenljivoj *r*. Program dalje poziva proceduru *stampaj* i prosleđuje joj rastojanje *r*, nakon čega procedura vrši štampanje ove vrednosti uz odgovarajući komentar.

Primer 2

Napisati program koji za *n* vrednosti sa ulaza vrši izračunavanje polinoma $x^5 + 3x^4 - 6x^2 + 2$.

```
program Polinom;
var  x:real;
     n,i:integer;

   function stepen(x:real;eksp:integer):real;
   var  i:integer;
        s:real;
   begin
      s:=1;
      for i:=1 to eksp do  s:=s*x;

      stepen:=s;
   end;

   function poli(x:real):real;
   begin
      poli:=stepen(x,5)+3.0*stepen(x,4)-6.0*sqr(x)+2;
   end;

begin
   writeln('Unesite koliko brojeva zelite:');
   readln(n);

   for i:=1 to n do
   begin
      writeln('Unesite ',i,'. broj:');
      readln(x);

      writeln('Vrednost polinoma za ',i,'. broj je ',poli(x):10:4);
```

RADNA VERZIJA

```
   end;
end.
```

U prethodnom primeru su definisane dve funkcije. Funkcija *stepen* izračunava vrednost x^{eksp}, a funkcija *poli* izračunava vrednost polinoma $x^5 + 3x^4 - 6x^2 + 2$. Primetimo da funkcija *poli* vrši izračunavanje polinoma korišćenjem prethodno definisane funkcije *stepen*.

U glavnom programu korisnik unosi *n* realnih brojeva i za svaki od njih se izračunava vrednost polinoma pozivom funkcije *poli*, a dobijeni rezultat štampa na ekranu.

Direktiva FORWARD

Da bi određeni potprogram (funkcija ili procedura) mogao da se pozove iz nekog drugog potprograma, neophodno je da potprogram koji se poziva bude definisan pre potprograma koji ga poziva. Ukoliko raspored potprograma nije takav da je ovaj zahtev obezbeđen, moguće je koristiti direktivu **forward** kako bi se naznačilo da će definicija potprograma koji se poziva uslediti kasnije u kodu. To se postiže tako što se ispred potprograma koji poziva navede samo zaglavlje potprograma koji se poziva praćeno direktivom **forward**, a sam potprogram koji se poziva se navodi kasnije u kodu. Na primer:

```
procedure p2(...); forward;

procedure p1(...);
begin
   ...
   p2(...);
   ...
end;

procedure p2(...);
begin
   ...
end;
```

Opseg važenja i životni vek promenljivih

Opseg važenja neke promenljive je deo programa u kome ta promenljiva može biti korišćena. U tom delu programa data promenljiva ima značenje i za nju kažemo da je "vidljiva" u tom opsegu. Na primer, promenljiva deklarisana unutar funkcije ili procedure ima značenje i vidljiva je samo unutar tog potprograma. Takvu promenljivu nazivamo **lokalna promenljiva**. Sa druge strane, **globalna promenljiva** se deklariše u glavnom programu i vidljiva je iz bilo kog dela programa.

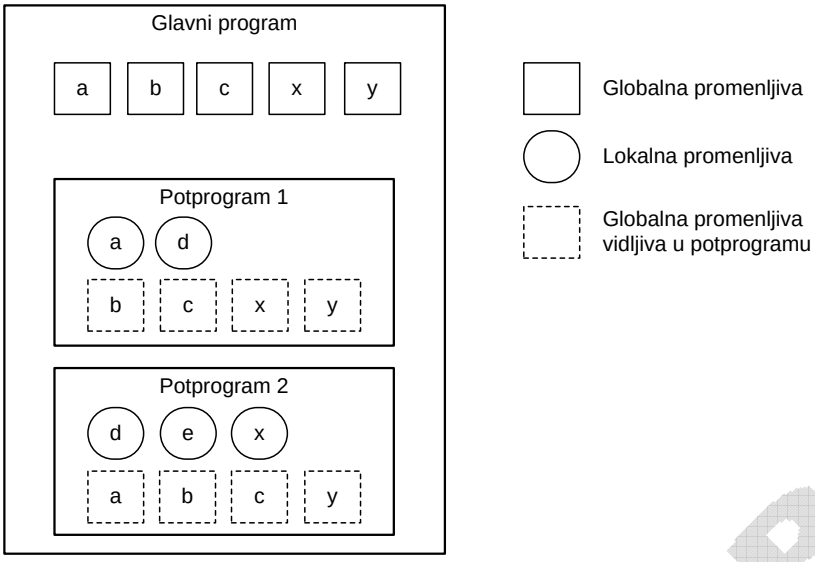
Na Slici ### su šematski prikazani opsezi važenja pojedinih promenljivih. U glavnom programu su deklarisane globalne promenljive *a*, *b*, *c*, *x* i *y*, koje su vidljive u svim delovima programa pa samim tim i u Potprogramu 1 i Potprogramu 2.

U potprogramu 1 su deklarisane lokalne promenljive *a* i *d*, koje su vidljive samo u ovom potprogramu i ne mogu se koristiti van njega. Pored ove dve promenljive u Potprogramu 1 su vidljive i globalne promenljive *b*, *c*, *x* i *y*. Svaka promena vrednosti ovih promenljivih u bilo kom delu programa je vidljiva i u Potprogramu 1. Takođe, promena vrednosti ovih promenljivih u Potprogramu 1 izazvala bi promenu njihovih vrednosti i u svim ostalim delovima programa, iz razloga što su to ustvari jedne te iste promenljive. Međutim, globalna promenljiva *a* nije vidljiva u Potprogramu 1 iz razloga što istoimena lokalna promenljiva ima prioritet. Iako globalna i lokalna promenljiva *a* imaju isti naziv, to su zapravo dve različite promenljive. Ukoliko bi u Potprogramu 1 došlo do promene vrednosti lokalne promenljive *a*, to ne bi imalo nikakvog uticaja na globalnu promenljivu *a*. Takođe, promena vrednosti globalne promenljive *a* nema uticaja ne vrednost lokalne promenljive *a* u Potprogramu 1.

U Potprogramu 2 su deklarisane lokalne promenljive *d*, *e* i *x*, koje su vidljive samo u ovom potprogramu i ne mogu se koristiti van njega. Pored ovih promenljivih, u Potprogramu 2 su vidljive i globalne promenljive *a*, *b*, *c* i *y*, čije smo osobine opisali na primeru Potprograma 1.

PROBNA VERZIJA

Primetimo da oba potprograma poseduju lokalnu promenljivu *d*, ali zahvaljujući tome što su ove dve promenljive deklarisane u različitim potprogramima, one imaju potpuno odvojen opseg važenja. To praktično znači da promena promenljive *d* u jednom potprogramu nema nikakvog uticaja na istoimenu promenljivu u drugom potprogramu. Iako imaju isti naziv, to su zapravo dve različite promenljive.



Slika ### Šematski prikaz opsega važenja globalnih i lokalnih promenljivih

S obzirom da promenljiva nema značenje van svog opsega, besmisleno je njeno korišćenje van dela programa u kome je deklarisana. U kompajlerskim jezicima se u trenutku prevođenja programa vrši analiza korišćenja promenljivih unutar programa. U slučaju da je pokušano korišćenje neke promenljive van njenog opsega, kompajler prijavljuje grešku.

Životni vek promenljive opisuje u kom delu programa određena promenljiva ima vrednost, odnosno u kojim trenucima izvršenja programa promenljiva započinje i završava svoje postojanje. Na početku životnog veka promenljive, program obezbeđuje deo memorije u kome će biti smeštena njena vrednost. Životni vek promenljive prestaje oslobađanjem prethodno zauzetog dela memorije.

Opseg važenja direktno utiče na životni vek promenljive. Životni vek promenljive obično počinje ulaskom u njen opseg važenja, a prestaje izlaskom iz njega, kako bi se izbeglo nepotrebno trošenje memorije na promenljive koje nisu vidljive u trenutnom opsegu. Ovakve promenljive se često nazivaju i automatske. Pored automatskih promenljivih, u nekim jezicima postoje i takozvane statičke promenljive, čiji životni vek traje i nakon izlaska iz opsega, tako da se ove promenljive mogu ponovo koristiti ukoliko se program vrati u pomenuti opseg.

Dobra programerska praksa podrazumeva pravljenje što manjih opsega važenja pojedinih promenljivih, kako ne bi došlo do slučajnog mešanja promenljivih dva različita dela programa. Iako su globalne promenljive vidljive u svim potprogramima, treba izbegavati njihovo menjanje u funkcijama i procedurama. Ukoliko potprogrami menjaju globalne promenljive, prilikom čitanja glavnog dela programa veoma je teško uočiti šta se zapravo dešava sa pojedinim promenljivama, što može izazvati neželjene efekte. U slučaju da je potrebno da određeni potprogram promeni neku od globalnih promenljivih, najbolje je tu promenljivu proslediti funkciji ili proceduri kao parametar. Na taj način i u glavnom programu postaje jasno da su određene globalne promenljive "poverene" potprogramu i da ih on može promeniti. O načinima prenošenja parametara iz glavnog programa u potprogram biće više reči u narednoj sekciji.

Primer

```
program Opseg;
var  a,b,c:integer;
     x,y:real;

   procedure Potprogram1;
```

RADNA VERZIJA

```
   var  a,d:integer;
   begin
      a:=11;
      d:=c+12;
      b:=13;
      x:=3.14;
   end;

   function Potprogram2:integer;
   var  d,e,x:integer;
   begin
      x:=5;
      d:=a+x;
      a:=d+1;

      Potprogram2:=-1;
   end;

begin
   a:=1;
   b:=2;
   c:=3;
   x:=4.0;
   y:=5.0;

   Potprogram1; {  Nakon izvrsenja ove linije vrednosti promenljivih su:
                   a=1, b=13, c=3, d nije vidljivo, e nije vidljivo,
                   x=3.14, y=5.0
                }

   Potprogram2; {  Nakon izvrsenja ove linije vrednosti promenljivih su:
                   a=7, b=13, c=3, d nije vidljivo, e nije vidljivo,
                   x=3.14, y=5.0
                }
end.
```

PROBNA VERZIJA

Prenos parametara

Do sada smo videli da glavni program ili neki potprogram mogu pozvati određenu funkciju ili proceduru u cilju izvršenja određenog zadatka. Da bi zadatak mogao biti izvršen najčešće je potrebno potprogramu proslediti određene podatke koje nazivamo parametrima ili argumentima funkcije ili procedure.

Takođe, videli smo i da funkcije preko svog imena mogu glavnom programu vratiti neku vrednost. Za razliku od funkcija, procedure programu ili potprogramu koji ih je pozvao ne vraćaju nikakvu vrednost već samo izvršavaju određeni zadatak. Međutim, u realnim problemima je često slučaj da je potrebno da potprogram glavnom programu vrati više od jedne vrednosti. Vraćanje više od jedne vrednosti moguće je izvršiti preko liste parametara, o čemu će u nastavku biti reči.

Za pravilno korišćenje funkcija i procedura veoma je važno potpuno razumevanje funkcionisanja prenosa parametara između glavnog programa i potprograma. Iz tog razloga će u nastavku biti detaljno obrađen ovaj proces.

Svaka funkcija i procedura u svom zaglavlju imaju definisanu listu parametara koje zahtevaju od programa ili potprograma koji ih poziva. Ove parametre nazivamo **formalni parametri**. U programskom jeziku Pascal postoje dve vrste formalnih parametara:

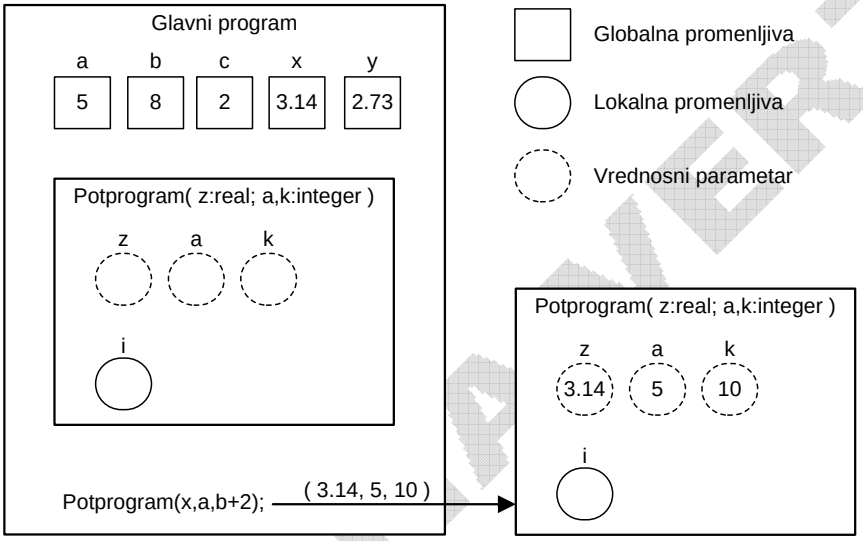
- vrednosni
- promenljivi (varijabilni)

Kada glavni program ili neki potprogram pozove neku funkciju ili proceduru, on joj prosleđuje potrebne vrednosti koje nazivamo **stvarni parametri**. Lista stvarnih parametara mora da odgovara listi formalnih parametara po broju, tipu i redosledu. To znači da glavni program mora potprogramu da prosledi tačno onoliko parametara koliko potprogram zahteva, pri čemu parametri moraju da budu istog tipa i navedeni u potpuno istom redosledu kao što su navedeni u zaglavlju potprograma.

Prenos vrednosnih parametara

Razmotrimo šta se dešava prilikom pozivanja potprograma koji zahteva samo vrednosne parametre. U trenutku pozivanja takvog potprograma, vrednosti koje su u pozivu navedene kao stvarni parametri se kopiraju u formalne parametre funkcije i procedure. Vrednosni formalni parametri potprograma se ponašaju potpuno isto kao i lokalne promenljive tog potprograma. Oni su vidljivi samo unutar te funkcije ili procedure i ne može im se pristupiti iz glavnog programa ili nekog drugog potprograma. Zahvaljujući tome, bilo kakva promena vrednosti tih promenljivih unutar potprograma, nema nikakvog uticaja na ostatak programa, pa samim tim ni na stvarne parametre koji su prosleđeni potprogramu.

Na Slici ### je prikazan postupak prosleđivanja vrednosnih parametara potprogramu.



Slika ### Šematski prikaz prenosa vrednosnih parametara

Na Slici ### šematski je prikazan program koji sadrži celobrojne promenljive *a*, *b* i *c* i realne promenljive *x* i *y* sa vrednostima navedenim u kvadratima. U okviru programa je definisan i Potprogram koji ima tri vrednosna parametra i to realni parametar *z* i celobrojne parametre *a* i *k*. Potprogram, takođe, koristi i lokalnu promenljivu *i*.

Glavni program poziva Potprogram korišćenjem linije

```
Potprogram(x,a,b+2);
```

U trenutku poziva vrednosti navedenih stvarnih parametara (*x*, *a*, *b*+2) se prosleđuju Potprogramu, i upisuju u formalne parametre (*z*, *a*, *k*). Primetimo da stvarni vrednosni parametri mogu biti promenljive, izrazi i konstante iz razloga što se Potprogramu proseđuju samo vrednosti ovih parametara.

Unutar Potprograma promenljive *z*, *a* i *k* se ponašaju kao lokalne promenljive i njihova promena unutar Potprograma neće imati nikakvog uticaja na ostatak programa. Takođe, Glavni program ne može pristupiti ovim promenljivama, a njihove vrednosti može zadati samo prilikom poziva Potprograma. S obzirom da se fiktivni parametri ponašaju kao lokalne promenljive potprograma, njihovi nazivi moraju biti jedinstveni samo unutar Potprograma i nisu ni u kakvoj vezi sa nazivima stvarnih parametara. Tako, vrednost stvarnog parametra *x* se kopira u formalni parametar *z*, vrednost stvarnog parametra *a* se kopira u formalni parametar *a*, a vrednost izraza *b*+2 se upisuje u formalni parametar *k*. Iako stvarni parametar *a* i formalni

RADNA VERZIJA

parametar *a* nose isti naziv, oni međusobno nisu povezani. To su zapravo dve različite promenljive, od kojih je jedna deklarisana unutar Glavnog programa kao globalna promenljiva, a druga kao formalni parametar Potprograma, što znači da su njen opseg važenja i životni vek samo unutar ovog potprograma.

Primer

```
program Vrednosni;
var  a,b,c:integer;
     x,y:real;

   procedure Potprogram(z:real; a,k:integer);
   var  i:integer;
   begin
      a:=11;

      for i:=1 to k do z:=z+1;

      writeln(a,z);
   end;

begin
   a:=5;
   b:=8;
   c:=2;
   x:=3.14;
   y:=2.73;

   Potprogram(x,a,b+2);    { Nakon izvrsenja ove linije vrednosti promenljivih su:
                             a=5, b=8, c=2, x=3.14, y=2.73 }

   writeln(a,b,c,x,y);
end.
```

Prenos promenljivih parametara

U dosadašnjem razmatranju smo imali prilike da vidimo da jedino funkcija može vratiti glavnom programu neku vrednost preko svog naziva. Međutim, u realnim problemima je često slučaj da je potrebno da potprogram glavnom programu vrati više od jedne vrednosti.

U prethodnom delu smo mogli videti kako se vrši prenos vrednosnih parametara iz glavnog programa u potprogram. S obzirom da se vrednosni parametri ponašaju kao lokalne promenljive, ni jedna njihova promena neće biti vidljiva u glavnom programu. Samim tim nije moguće glavnom programu vratiti neku vrednost putem vrednosnih parametara.

Iz tog razloga postoji i druga vrsta formalnih parametara koje nazivamo promenljivim ili varijabilnim parametrima. Kod ove vrste parametara potprogramu se ne prenose vrednosti stvarnih parametara, već njihove adrese. Na taj način i stvarni i formalni parametri ukazuju praktično na istu memorijsku lokaciju, čime se postiže da svaka promena formalnog parametra unutar potprograma direktno menja vrednost odgovarajućeg stvarnog parametra u glavnom programu. Za razliku od *prenosa parametara po vrednosti*, ovaj način prenosa se naziva *prenos parametara po referenci*.

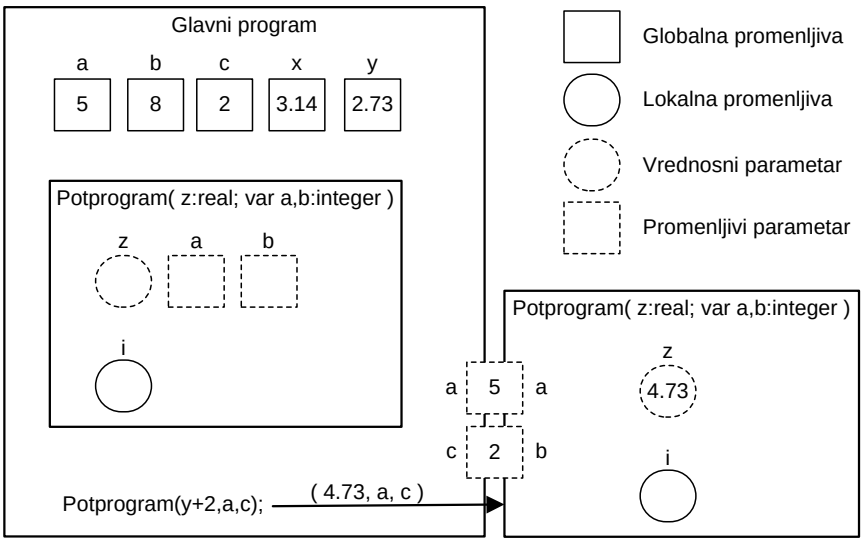
Promenljivi parametri se u programskom jeziku Pascal deklarišu korišćenjem rezervisane reči **var** unutar liste parametara. Tako na primer procedura *Test* definisana kao

```
procedure Test(z:real; var a,b:integer);
```

ima jedan realan vrednosni parametar *z* i dva promenljiva parametra *a* i *b*. Promena parametra *z* unutar procedure neće imati uticaja na promenljive u glavnom programu. Međutim, promena parametara *a* i *b* će direktno značiti i promenu odgovarajućih stvarnih parametara koji su prosleđeni proceduri.

Na Slici ### dat je šematski prikaz prenosa promenljivih parametara:

RADNA VERZIJA



Slika ### Šematski prikaz prenosa promenljivih parametara

Na Slici ### šematski je prikazan program koji sadrži celobrojne promenljive *a*, *b* i *c* i realne promenljive *x* i *y* sa vrednostima navedenim u kvadratima. U okviru programa je definisan i Potprogram koji ima tri formalna parametra i to realni vrednosni parametar *z* i celobrojne promenljive parametre *a* i b. Potprogram, takođe, koristi i lokalnu promenljivu *i*.

Glavni program poziva Potprogram korišćenjem linije

```
Potprogram(y+2,a,c);
```

U trenutku poziva vrednost stvarnog parametara (y+2) se prosleđuje Potprogramu, i upisuje u formalni parametar *z*. Pored toga, Potprogramu se prosleđuju i stvarni parametri *a* i *c*, koji odogovaraju promenljivim formalnim parametrima *a* i *b*. Za razliku od vrednosnog parametra *z*, u ovom slučaju se ne prosleđuju vrednosti promenljivih *a* i *c*, već njihove adrese u memoriji. Umesto konkretnih vrednosti, formalnim parametrima *a* i *b* se dodeljuje da ukazuju na istu memorijsku lokaciju prosleđenih promenljivih *a* i *c*. Od tog trenutka promenljiva *a* unutar Potprograma deli istu memorijsku lokaciju sa globalnom promenljivom *a*. Slično i promenljiva *b* deli istu memorisjku lokaciju sa globalnom promenljivom *c*. To praktično znači da će svaka promena promenljive *a* unutar Potprograma istovremeno izazvati promenu i globalne promenljive *a*. Takođe, svaka promena promenljive *b* unutar Potprograma izaziva promenu globalne promenljive *c*. Primetimo da, iako dele isti memorijski prostor, stvarni i formalni parametri potprograma mogu imati iste ili različite nazive, iz razloga što imaju različite opsege važenja.

Korišćenjem opisanog mehanizma moguće je u određenom potprogramu dodeliti vrednosti pojedinim promenljivim parametrima i na taj način direktno promeniti vrednosti više promenljivih u glavnom programu. To praktično omogućava da potprogram glavnom programu vrati više od jedne vrednosti.

Primer

```
program Promenljivi;
var  a,b,c:integer;
     x,y:real;

   procedure Potprogram(z:real; var a,b:integer);
   var  i:integer;
   begin
     a:=11;
     b:=a div 2;

     for i:=1 to b do z:=z+1;
```

RADNA VERZIJA

```
      writeln(a,b,z);
   end;

begin
   a:=5;
   b:=8;
   c:=2;
   x:=3.14;
   y:=2.73;

   Potprogram(y+2,a,c);    { Nakon izvrsenja ove linije vrednosti promenljivih su:
                             a=11, b=8, c=5, x=3.14, y=2.73 }

   writeln(a,b,c,x,y);
end.
```

Primer 2

Napisati program koji za zadati prirodan broj *n* pozivanjem potprograma izračunava sumu i proizvod svih prirodnih brojeva manjih od *n*.

```
program Primer2;
var  n:integer;
     s,p:real;

   procedure SumaProizvod(k:integer; var suma,proizvod:integer);
   var  i:integer;
   begin
      suma:=0;
      proizvod:=1;

      for i:=1 to k-1 do
      begin
         suma:=suma+i;
         proizvod:=proizvod*i;
      end;

   end;

begin
   writeln('Unesite prirodan broj n:');
   readln(n);

   SumaProizvod(n,s,p);

   writeln('Suma prirodnih brojeva manjih od ',n,' je ',s);
   writeln('Proizvod prirodnih brojeva manjih od ',n,' je ',p);
end.
```

RADNA VERZIJA

Rekurzivne funkcije i procedure

Do sada smo imali prilike da vidimo kako se problem može razbiti na više manjih potproblema, koji se mogu dalje poveriti funkcijama i procedurama na rešavanje. Štaviše, svaki od potproblema se može dalje deliti na još manje potprobleme koji se rešavaju novim potprogramima. Na taj način veliki problemi se dekomponuju na manje, logički povezane celine, kako bi se pojednostavilo njihovo rešavanje. Potprogrami su specijalizovani za rešavanje samo određenih problema, čime se poboljšava čitljivost programa, omogućava njegovo lakše menjanje i olakšava otkrivanje eventualnih grešaka.

U prethodnoj sekciji smo mogli videti da glavni program može pozivati funkcije i procedure, ali i da ti potprogrami mogu dalje pozivati druge potprograme, kako bi im poverili rešavanje dela problema. Međutim, neki problemi su takve prirode da se mogu rešiti rešavanjem jednog ili više problema sličnih početnom problemu. Posmatrajmo, na primer, izračunavanje faktorijela nekog prirodnog broja. Faktorijel prirodnog broja *n* je proizvod svih prirodnih brojeva od 1 do *n*, što se može napisati kao

$$n! = n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

Ako pažljivo pogledamo prethodni izraz, primetićemo da on može da se napiše i kao

$$n! = n \cdot (n-1)!$$

Na ovaj način smo izračunavanje faktorijela broja *n* sveli na izračunavanje faktorijela broja *n-1*, koji zatim množimo brojem *n*.

Recimo da u Pascal-u želimo da napravimo funkciju za izračunavanje faktorijela koja bi imala jedan celobrojni parametar *n* i koja bi vraćala celobrojnu vrednost:

```
function fakt(n:integer):integer;
begin
   ...
end;
```

Pored klasičnog načina izračunavanja faktorijela množenjem brojeva od 1 do *n* u petlji, ovaj problem se može rešiti i na način prikazan u prethodnim izrazima. Dakle, funkciju za računanje faktorijela broja *n* ćemo realizovati pozivanjem iste te funkcije za broj *n-1*, a zatim ćemo dobijeni rezultat pomnožiti sa *n*:

```
function fakt(n:integer):integer;        /* Ova funkcija nije dobra */
begin
   fakt:=n*fakt(n-1);
end;
```

Ovakva situacija kada funkcija, odnosno procedura, poziva samu sebe se naziva **rekurzija**. Da bi se u potpunosti ovladalo korišćenjem rekurzivnih potprograma, neophodno je razumeti mehanizam rekurzivnih poziva.

Podsetimo se da svaki potprogram ima svoje lokalne promenljive, kao i formalne parametre koji se ponašaju potpuno isto kao i lokalne promenljive. Nakon pozivanja potprograma kreiraju se njegove lokalne promenljive i formalni parametri, čime počinje njihov životni vek. Na kraju izvršavanja potprograma sve ove promenljive se uništavaju, što predstavlja kraj njihovog životnog veka. Dakle, kada pozovemo neki potprogram, praktično se stvara jedan njegov primerak (instanca), koji u sebi sadrži sve lokalne promenljive i formalne parametre. Svaki poziv funkcije, odnosno procedure, stvara njen novi primerak sa svim potrebnim promenljivama. Svaki primerak funkcije je potpuno nezavistan od svih drugih primeraka iste funkcije ili drugih funkcija. Zahvaljujući ovom mehanizmu moguće je da jedan primerak neke funkcije pozove drugi primerak te iste funkcije, što nazivamo rekurzijom. Imajući u vidu da su ova dva primerka potpuno nezavisna i da svaki od njih ima svoje promenljive, neće doći do njihovog mešanja, već će se poziv odvijati potpuno isto kao da jedna funkcija poziva neku drugu funkciju.

Posmatrajmo rekurzivni poziv funkcije *fakt* u prethodnom primeru. Recimo da glavni program poziva funkciju *fakt* sa parametrom *n=5*. Prilikom poziva formira se prvi primerak funkcije *fakt* čiji je parametar 5. Da bi izračunao faktorijel broja 5, ovaj primerak funkcije poziva novi primerak te iste funkcije i prosleđuje joj parametar *n-1*, odnosno 4. Kada bude dobio rezultat drugog primerka funkcije *fakt*, prvi primerak će ga pomnožiti sa *n* i tako dobiti faktorijel broja 5.

Međutim, da pogledamo kako će drugi primerak funkcije *fakt* izračunati faktorijel broja 4. Na sličan način pozvaće treći primerak te iste funkcije i proslediti mu parametar 3, a zatim rezultat pomnožiti sa 4. Primećujemo da svaki primerak funkcije računa faktorijel tako što pozove novi primerak te iste funkcije i prosledi joj parametar za jedan manji od broja za koji računa faktorijel. Ako bi se tako nastavilo, pozivi novih primeraka funkcije *fakt* bi išli u nedogled i nijedan primerak ne bi obavio svoj posao, već bi svaki od njih prenosio deo problema na novi primerak funkcije. Iz tog razloga ovako napisana funkcija *fakt* nije korektna i nikada neće izračunati faktorijel nekog broja.

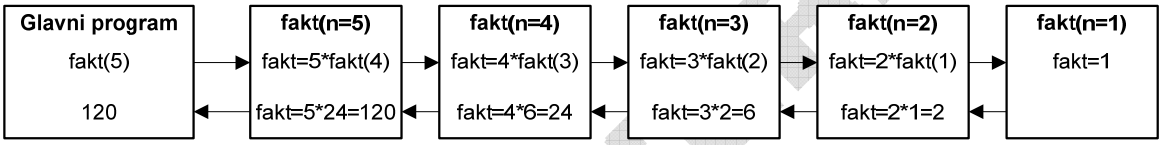
RADNA VERZIJA

Da pozivi novih primeraka ne bi išli u nedogled, neophodno je da postoji **uslov za izlazak iz rekurzije**, odnosno uslov pri kome funkcija više neće pozivati samu sebe, već će se sama pobrinuti za rešenje problema. U slučaju funkcije *fakt* taj uslov bi bio da je *n<2*, zato što je u tom slučaju faktorijel jednak 1 i funkcija nema potrebe da problem dalje razbija i prosleđuje ga novim primercima. Dakle, ispravan oblik funkcije *fakt* bi izgledao ovako:

```
function fakt(n:integer):integer;
begin
   if n<2 then
      fakt:=1;
   else
      fakt:=n*fakt(n-1);
end;
```

Na ovaj način svaki primerak funkcije će pozivati novi primerak i prosleđivaće mu broj za jedan manji od njenog parametra, sve dok se ne stigne do primerka koji kao parametar dobije broj manji od 2. Kada neki primerak dobije parametar manji od 2, on će prethodnom primerku vratiti rezultat 1. Prethodni primerak će rezultat pomnožiti sa 2 i to rešenje vratiti primerku koji je njega pozvao. Ovi rekurzivni pozivi će se tako razmotavati unazad, sve dok se ne stigne do prvog primerka, koji će konačno vratiti rezultat glavnom programu.

Na Slici ### je šematski prikazano rekurzivno pozivanje funkcije *fakt* i zatim vraćanje rezultata unazad.



Slika ### Rekurzivni pozivi funkcije za računanje faktorijela *fakt*

Rekurzija omogućava veoma elegantno rešavanje nekih problema uz vrlo malo programskog koda. Međutim, prilikom korišćenja rekurzije treba biti oprezan iz razloga što svaki novi primerak funkcije zauzima određeni deo memorije koja je ograničenog kapaciteta, tako da za veliki broj rekurzivnih poziva vrlo lako može doći do prekoračenja i greške u izvršavanju programa. Na sreću, svaki rekurzivni algoritam se može transformisati u iterativni, čime se izbegavaju navedeni problemi. Jednostavne probleme, koji ne zahtevaju korišćenje rekurzije, treba u svakom slučaju rešavati iterativno. Tipičan primer je računanje faktorijela koje se vrlo jednostavno moglo izvesti iterativno, korišćenjem samo jedne *for* petlje. Rekurzivno rešenje će se moći izvršiti samo za manje brojeve, koji ne zahtevaju preveliki broj primeraka funkcije. U slučaju većih brojeva, vrlo brzo će doći do zagušenja memorije usled velikog broja rekurzivnih poziva.

Primer

Napisati program koji pomoću rekurzije izračunava sumu geometrijskog niza dužine *n* čiji je prvi član *a,* a koeficijent *q*. Da se podsetimo, geometrijski niz je niz brojeva takvih da je količnik svakog broja i njegovog prethodnika konstantan i jednak koeficijentu *q*. Drugim rečima, svaki član niza može se dobiti množenjem prethodnog člana koeficijentom *q*, pri čemu je prvi član niza zadat. Sada geometrijski niz dužine *n* možemo napisati kao

$$a,\ aq,\ aq^2,\ aq^3,\ aq^4,...,\ aq^{n-1}$$

a njegovu sumu u obliku

$$S = a + aq + aq^2 + aq^3 + aq^4 +,...,+aq^{n-1} = a \cdot \left(1 + q + q^2 + q^3 + q^4 +,...,+q^{n-1}\right)$$

Ako pažljivo pogledamo izraz u zagradi, uočićemo da se radi o geometrijskom nizu dužine *n* čiji je prvi član 1, a koeficijent *q*, pa zaključujemo da se računanje sume bilo kog geometrijskog niza može svesti na računanje sume geometrijskog niza čiji je prvi član 1 i koju ćemo obeležiti sa S_n.

Međutim, izraz u zagradi možemo napisati i na sledeći način

$$S_n = 1 + q + q^2 + q^3 + q^4 + ,...,+ q^{n-1} = 1 + q \cdot \left(1 + q + q^2 + q^3 + ,...,+ q^{n-2}\right) = 1 + q \cdot S_{n-1}$$

što jasno pokazuje da se suma S_n može izraziti preko sume geometrijskog niza koji ima jedan element manje, kao

$$S_n = 1 + q \cdot S_{n-1}$$

i što nas upućuje na korišćenje rekurzivne funkcije. Pored rekurzivnog izraza koji je dat, neophodno je definisati i uslov za izlazak iz rekurzije, odnosno uslov pri kome nije moguće dalje razbijati problem da sličan potproblem. To je svakako slučaj kada se traži izračunavanje sume geometrijskog niza dužine 1, za koju sigurno znamo da je jednaka jedinici ($S_1 = 1$).

```
program Geom;
var  n:integer;
     a,q,s:real;

   function sumageo(n:integer;q:real):real;
   begin
      if n=1 then
         sumageo:=1;
      else
         sumageo:=1.0+q*sumageo(n-1,q);
   end;

begin
   writeln('Unesite duzinu niza:');
   readln(n);

   writeln('Unesite prvi clan i koeficijent:');
   read(a,q);

   s:=a*sumageo(n,q);

   writeln('Suma geometrijskog niza je:',s);
end.
```

RADNA VERZIJA