Funkcije

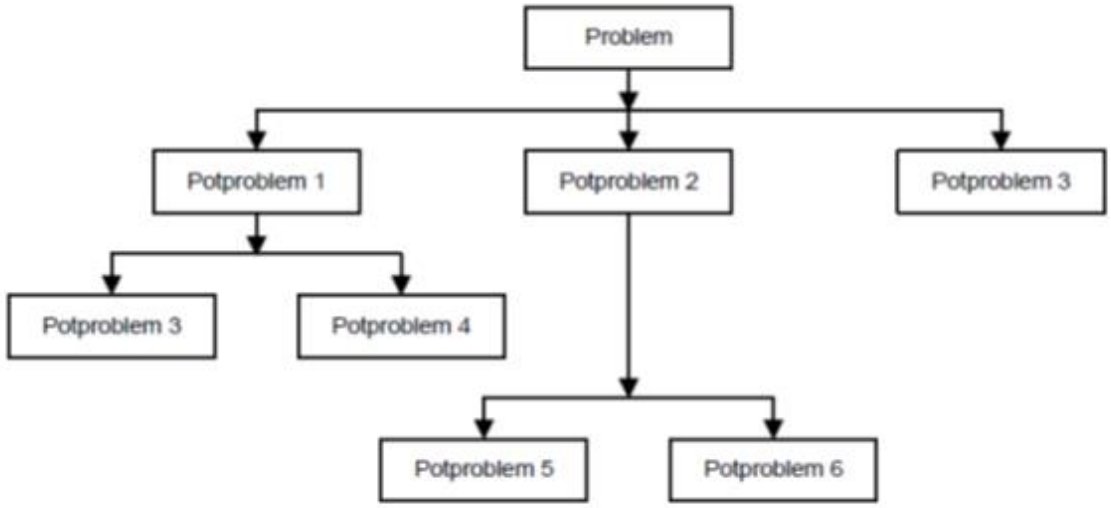
Prilikom rešavanja manjih problema korišćenjem računara, svi zadaci se mogu izvršiti u okviru glavnog programa. Međutim, sa povećanjem složenosti kompjuterskih programa, potrebno je program podeliti na odgovarajuće logičke i funkcionalne celine, takozvane **potprograme**. Potprogrami omogućavaju organizovanje 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

U programskom jeziku C potprograme je moguće realizovati korišćenjem **funkcija**. **Funkcija** je samostalan deo programa koji obavlja određeni zadatak i preko svog naziva i liste parametara, delu programa iz koga je pozvana može da 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.

Pored toga što se korišćenjem funkcija izbegava nepotrebno ponavljanje delova koda, pa samim tim povećava i čitljivost programa, funkcije 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:



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

Sintaksa

<tip_funkcije> <naziv>([lista parametara])
<blok_naredbi>

Svaka funkcija mora imati zaglavlje, koje sadrži jedinstveni naziv naveden odmah iza rezervisane reči **tipa funkcije**. Iza naziva funkcije sledi lista parametara unutar oblih zagrada. Za svaki parametar funkcije mora biti naveden i njegov tip, na sličan način kao i kod deklaracija promenljivih. Ukoliko funkcija ne zahteva nikakve ulazne podatke, moguće je ne navesti nijedan parametar.

Nakon definisanja zaglavlja funkcije sledi blok naredbi, odnosno telo funkcije. Kao i svaki drugi blok naredbi, telo funkcije se sastoji od niza komandi ograničenih rezervisanim znakovima **{** i **}**.

U slučaju da funkcija treba da vrati neku vrednost, neophodno je na kraju tela funkcije pomoću rezervisane reči **return** naglasiti koju vrednost funkcija vraća.

return <promenljiva>;

Svaka funkcija može biti pozvana iz glavnog programa (glavne funkcije) ili druge funkcije navođenjem njegovog imena i liste parametara unutar oblih zagrada. Prelim 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.

```
#include <stdio.h>
#include <math.h>
float rast(float x1, float y1, float x2, float y2)
{
float s;
s= sqrt(pow(x2-x1,2) + pow(y2-y1,2));
return (s);
}
main()
{
float x1, y1, x2, y2, r;
printf("Unesite koordinate prve tacke:\n");
scanf("%f%f", &x1, &y1);
printf("Unesite koordinate druge tacke:\n");
scanf("%f%f", &x2, &y2);
r = rast(x1, y1, x2, y2);
printf("Rastojanje izmedju tacaka je %10.4f\n", r);

}
```

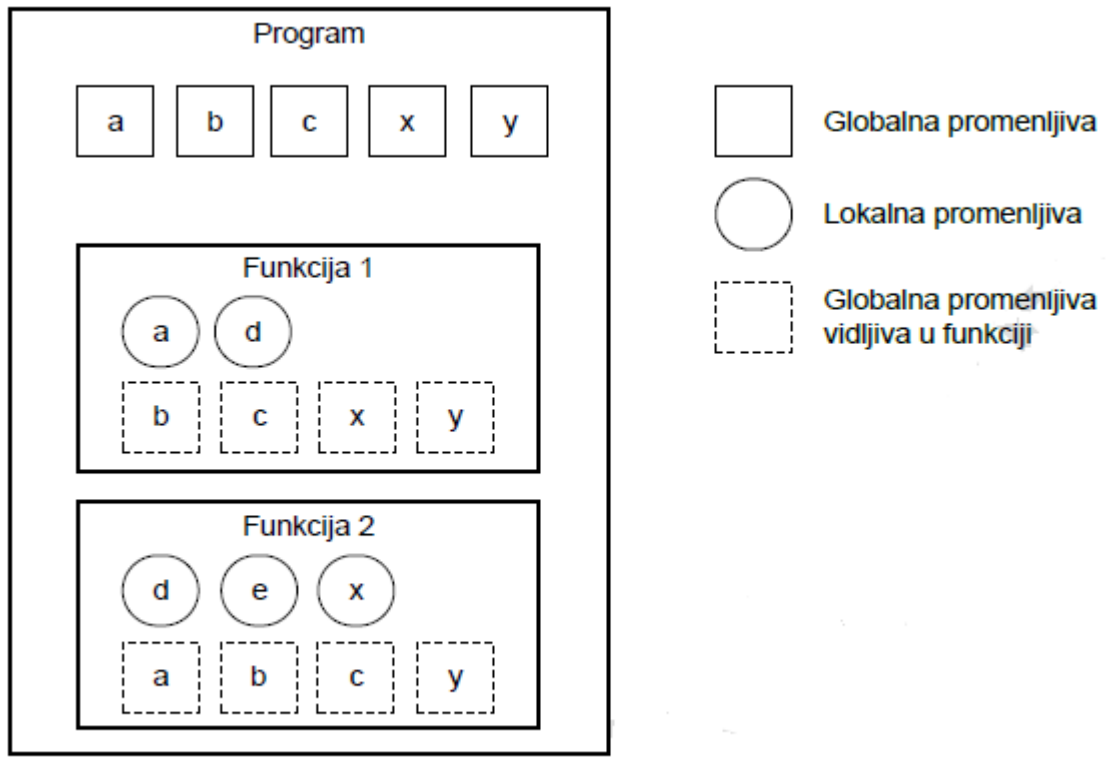
U prethodnom primeru definisana je funkcija rast. Ova funkcija na osnovu koordinata dve tačke izračunava rastojanje između njih.
Nakon unosa koordinata tačaka glavni program poziva funkciju *rast* i šalje joj koordinate tačaka. Kada funkcija završi izračunavanje, ona glavnom programu vraća rastojanje između navedenih tačaka. Glavni program, zatim, dobijenu vrednost dodeljuje promenljivoj *r*. Program dalje stampa rastojanje *r*.

Deklaracija funkcije

Da bi određena funkcija mogla da se pozove iz neke druge funkcije, neophodno je da funkcija koja se poziva bude definisana pre funkcije koja je poziva. Ukoliko raspored funkcija nije takav da je ovaj zahtev obezbeđen, moguće je deklarisati funkciju kako bi se naznačilo da će definicija funkcije koja se poziva uslediti kasnije u kodu. To se postiže tako što se ispred funkcije koja poziva navede samo zaglavlje funkcije koja se poziva, a sama funkcija koja se poziva se navodi kasnije u kodu. Na primer:

```
int f2(...);
void f1(...)
{
...
f2(...);
...
}
int f2(...)
{
...
}
```

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 ima značenje i vidljiva je samo unutar te funkcije. Takvu promenljivu nazivamo **lokalna promenljiva**. Sa druge strane, **globalna promenljiva** se deklariše van svih funkcija i vidljiva je iz bilo kog dela programa.
Na Slici ### su šematski prikazani opsezi važenja pojedinih promenljivih. Van svih funkcija su deklarisane globalne promenljive *a*, *b*, *c*, *x* i *y*, koje su vidljive u svim delovima programa pa samim tim i u Funkciji 1 i Funkciji 2.



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

U Funkciji 1 su deklarisane lokalne promenljive *a* i *d*, koje su vidljive samo u ovoj funkciji i ne mogu se koristiti van nje. Pored ove dve promenljive u Funkciji 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 Funkciji 1. Takođe, promena vrednosti ovih promenljivih u Funkciji 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 Funkciji 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 Funkciji 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 Funkciji 1.

U Funkciji 2 su deklarisane lokalne promenljive *d*, *e* i *x*, koje su vidljive samo u ovoj funkciji i ne mogu se koristiti van njega. Pored ovih promenljivih, u Funkciji 2 su vidljive i globalne promenljive *a*, *b*, *c* i *y*, čije osobine su opisane na primeru Funkciji 1.

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.

Prenos parametara

Kao što je ranije navedeno, izvršavanje svakog programa u programskom jeziku C započinje izvršavanjem funkcije **main**. Iz tog razloga funkciju **main** često nazivamo **glavni program**, kao što je uobičajeno u nekim drugim programskim jezicima. Radi jednostavnosti pisanja, u tekstu koji sledi ćemo koristiti termin **glavni program**, ali se navedeni principi mogu primeniti na poziv bilo koje funkcije iz bilo koje druge funkcije.

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

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

Svaka funkcija u svom zaglavlju imaju definisanu listu parametara koje zahteva od programa ili funkcije koja ih poziva. Ove parametre nazivamo **formalni parametri**. U većini programskih jezika postoje dve vrste formalnih parametara:

- vrednosni
- promenljivi (varijabilni)

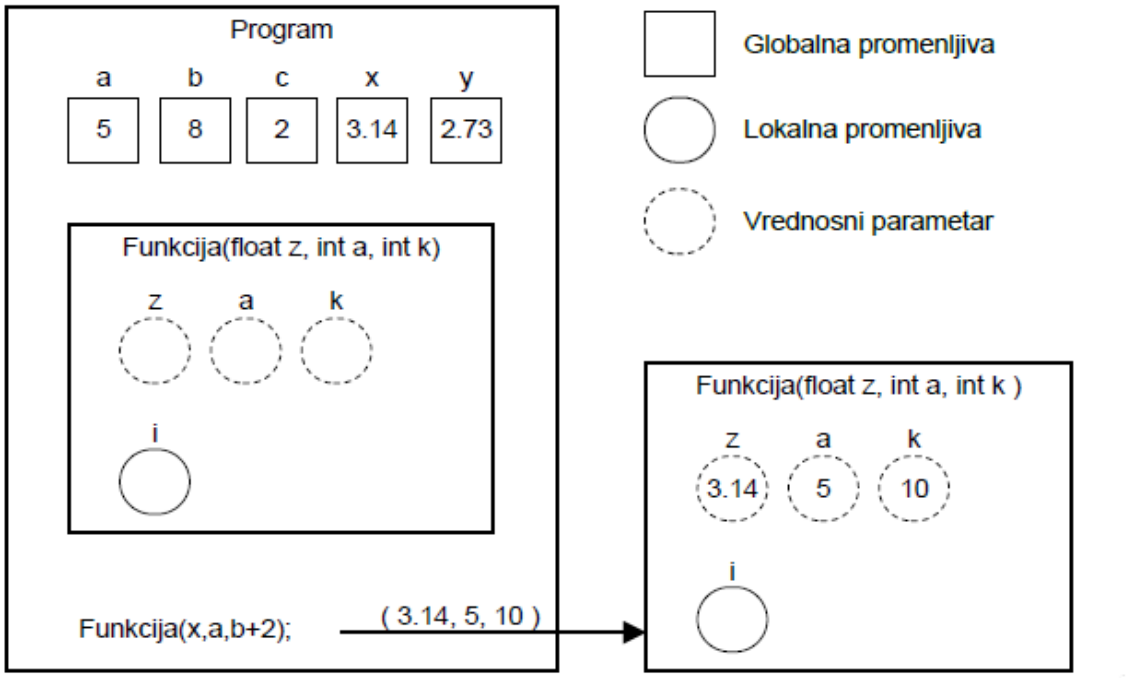
U programskom jeziku C se promenljivi parametri realizuju korišćenjem pokazivačkih promenljivih, o čemu će biti reči u višim razredima o programskom jeziku C.

Kada glavni program ili neka funkcija pozove neku funkciju, 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 funkciji da prosledi tačno onoliko parametara koliko funkcija zahteva, pri čemu parametri moraju da budu istog tipa i navedeni u potpuno istom redosledu kao što su navedeni u zaglavlju funkcije.

Prenos vrednosnih parametara

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

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



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 definisana i Funkcija koja ima tri vrednosna parametra i to realni parametar *z* i celobrojne parametre *a* i *k*. Funkcija, takođe, koristi i lokalnu promenljivu *i*.

Glavni program poziva Funkciju korišćenjem linije

Funkcija(x,a,b+2);

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

Unutar Funkcije promenljive *z*, *a* i *k* se ponašaju kao lokalne promenljive i njihova promena unutar Funkcije neće imati nikakvog uticaja na ostatak programa. Takođe, Glavni program ne može pristupiti ovim promenljivama, a

njihove vrednosti može zadati samo prilikom poziva Funkcije. S obzirom da se fiktivni parametri ponašaju kao lokalne promenljive funkcije, njihovi nazivi moraju biti jedinstveni samo unutar Funkcije 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 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 Funkcije, što znači da su njen opseg važenja i životni vek samo unutar ove funkcije.

Primer

```
#include <stdio.h>
int a, b, c;
float x, y;
void Funkcija(float z, int a, int k)
{
int i;
a = 11;
for(i = 1; i <= k; i++)
z = z+1;
printf("%d, %d\n", a, z);
}
main()
{
a = 5;
b = 8;
c = 2;
x = 3.14;
y = 2.73;
Funkcija(x, a, b+2); { Nakon izvrsenja ove linije vrednosti promenljivih su:
a=5, b=8, c=2, x=3.14, y=2.73 }
printf("%d %d %d %.2f %.2f\n", a, b, c, x, y);
 }
```

Zadaća

1. Napisati funkciju koja izračunava faktorijel prirodnog broja n, a zatim primeniti tu funkciju na :

- Prostu linijsku strukturu

Napisati program koji za unete vrednosti k,m,x izračunava s po formuli

$$s = \frac{k!}{(m+x)!}$$

- Razgranatu strukturu

Napisati program koji za unete vrednosti a i b izračunava

$$y = \begin{cases} (a-b)!, & a-b > 0 \\ (a+b)!, & a-b \leq 0 \end{cases}$$

- Cikličnu strukturu

Napisati program koji štampa sve trocifrene brojeve koji imaju osobinu da su jednaki zbiru faktorijela svojih cifara.