Funkcije

predavač: Nadežda Jakšić

Programiranje programski jezik C

Pojam

funkcije – delovi programa koji izvršavaju neki zadatak, celinu; dele na ugrađene, korisničke i **main** funkciju

ugrađene funkcije – printf,scanf... da bi se one izvršile potrebno je uključiti datoteke zaglavlja; nesamostalne su

korisničke funkcije – imenovani i samostalni delovi kôda koji izršavaju neki zadatak; funkcija ima jedinstveno ime koje joj daje korisnik, to ime služi za to da bi bila pozvana preko njega; mogu da se izvrše bez interakcije sa ostatkom kôda

korisničke funkcije se pišu nezavisno i pozivaju se iz **main** funkcije; obrađuju vrednosti koje dobiju putem argumenata i vraćaju ih nazad funkcijama koje su ih pozvale

Primer

```
#include <stdio.h>
 float a, stranica, povrsina;
 float kvadrat (float stranica);
 int main()
 {
    printf ("Unesite duzinu stranice kvadrata\n ");
    scanf (“%f", &a);
    povrsina = kvadrat (a);
    printf ("Povrsina kvadrata je %.2f\n", povrsina);
    return 0;
  }
   float kvadrat (float stranica)
  {
   return stranica * stranica;
  }
```

prototip funkcije, informiše kompajler o tipu promenljive koja će biti vraćena, o imenu funkcije i koje argumente funkcija koristi

poziva se funkcija **kvadrat**; prethodno je promenljiva **a** dobila vrednost; pošto je **a** unutar zagrada naziva se **argument** funkcije; pri prenošenju izvršavanja prenosi se samo **vrednost** argumenta

korisnička funkcija **kvadrat**; ona zahteva argument, preuzima ga, tj. uzima vrednost od promenljive **a** i pridružuje je svojoj promenljivoj **stranica**; računa površinu kvadrata i vraća vrednost naredbom **return**

Definicija

sintaksa:
tipRezultata imeFunkcije (formalniArgumenti)
{
definicije i deklaracije;
naredbe;
}

- **tipRezultata** je tip koji ima rezultat funkcije; rezultat je ono što funkcija vraća kada se pozove; ako se ne navede podrazumeva se **int**; kod funkcija koje ne vraćaju vrednost navodi se **void**
- **imeFunkcije** je identifikator
- **formalniArgumenti** su argumenti pomoću kojih se unose početni podaci u funkciju; oni rezervišu mesta za veličine sa kojima funkcija operiše posle njenog poziva; za svaki argument mora posebno da se navede tip **(int a, int b, float c...)**
- vrednosti koje se dodeljuju formalnim argumentima nazivaju se **stvarni argumenti**

Definicija

- dva načina dodeljivanja vrednosti formalnim argumentima: **pozivanje po vrednosti** (formalnim argumentima se dodeljuje vrednost stvarnog argumenta, nema izmene stvarnog argumenta po izlasku iz funkcije); **pozivanje po referenci** (ako se umesto vrednosti stvarnih argumenata prosleđuju adrese tih argumenata onda se menja vrednost stvarnim argumentima kada se izađe iz funkcije)
- **definicije i deklaracije** i **naredbe** čine telo funkcije
- naredba **return** se koristi za povratak iz funkcije; rezultat koji ova funkcija vraća može da bude promenljiva, pokazivač ili aritmetički izraz
- dva slučaja naredbe**: return (povratnaVrednost)** kada funkcija vraća neku vrednost ili **return** kada ne vraća vrednost; tip funkcije koja ne vraća vrednost je **void**
- ako se funkcija piše ispod **main** funkcije, onda iznad **main** funkcije mora da se navede prototip funkcije

  tipFunkcije imeFunkcije (formalniParametri);
- **poziv funkcije**: ako funkcija vraća vrednost onda **imeFunkcije (stvarniParametri),** a ako ne vraća vrednost onda **imeFunkcije ()**

Primer

```
//funkcija ne vraća vrednost
.....
void pozdrav ()    //ako funkciju pišemo ispod funkcije main, onda ovde
{                  //mora da se navede prototip funkcije void pozdrav ();
    printf ("Zdravo\n");
}
int main ()
{
    int i;
    for (i=1;i<=10;i++)
    {
        pozdrav (); //poziv funkcije pozdrav koja ne vraća vrednost
    }
return 0;
}
```

Primer

```
//funkcija vraća vrednost – zbir dva broja
.....
int zbir (int a, int b)    //funkcija će vratiti rezultat tipa int, formalni
{      //argumenti su int a, int b, za svaki mora posebno da se navede tip
    int c;
    c=a+b;
return c; //može i return a+b; naredba return vraća zbir koji je tipa int
}
int main ()
{
    int broj1, broj2;
    printf ("Unesite dva broja\n");
    scanf ("%d%d",&broj1,&broj2);
    printf ("Zbir ova dva broja je %d",zbir (broj1, broj2);
return 0;}
```

poziv funkcije **zbir**, **broj1** i **broj2** su **stvarni argumenti**

Primer

```
//funkcija ne vraća vrednost – zbir dva broja
.....
void zbir (int a, int b) //funkcija ima tip void
{
    int c;
    c=a+b;
    printf ("Zbir ova dva broja je %d", c);
}
int main ()
{
    int broj1, broj2;
    printf ("Unesite dva broja\n");
    scanf ("%d%d",&broj1,&broj2);
    zbir (broj1, broj2); //poziv funkcije zbir
    return 0;}
```

Primer

```
//funkcija vraća vrednost – kub nekog broja
int kub (int x)
{
    int rez;
    rez=x*x*x;
    return rez;
}
int main ()
{
    int a,k;
    printf ("Unesite broj\n");
    scanf ("%d", &a);
    k=kub (a);
    printf ("Kub unetog broja je %d", k);
return 0; }
```

Primer

```
//funkcija računa prosek dva broja

float nadjiProsek (float a, float b)
{
    float prosek;
    prosek=(a+b)/2;
    return (prosek);
}
int main ()
{
    float a=5, b=15, rezultat;
    rezultat=nadjiProsek (a,b);
    printf ("prosek je %.2f", rezultat);
}
```

Primer

```
//funkcija prikazuje kvadrate brojeva od 1 do 9

void kvadrati ()
{
    int i;
    for (i=1;i<=9;i++)
    printf ("%d\n", i*i);
}
int main ()
{
    kvadrati ();
    return 0;
}
```

Prost broj

```
//funkcija ispituje da li je broj prost u intervalu od m do n
int prostBroj (int broj);
int main()
{  int m,n,i;
   printf ("Unesite dva broja (interval):\n");
   scanf ("%d%d", &m,&n);
   printf ("Prosti brojevi u intervalu od %d do %d su:\n", m, n);
   for (i=m; i<=n; i++)
   {  if (prostBroj (i))
         printf ("%d", i); }
   return 0;}
int prostBroj (int broj) {
int i, prost=1;
for( i=2;i<=broj/2;i++)
 {      if (broj%i==0) {
            prost=0;
            break; }
  }  return prost; } //obezbediti da početak intervala bude broj koji je veći od
                                  jedan
```

vrednost promenljive **prost** se vraća u glavnu funkciju i ako promenljiva ima vrednost jedan prikazuje se broj

Programi

1. funkcija koja **n** puta ispisuje neki tekst, broj **n** se unosi u glavnom programu
2. u glavnoj funkciji se unose tri broja, napisti funkciju koja ispisuje najveći od tri broja
3. u glavnoj funkciji učitati **n** brojeva, napisti funkciju koja računa aritmetičku sredinu unetih brojeva
4. napisti funkciju koja prihvata dva broja i vraća veći od ta dva broja
5. u glavnoj funkciji učitati dva broja, napisti funkciju koja računa zbir brojeva u intervalu od prvog do drugog broja
6. u glavnoj funkciji učitati broj **n**, napisati dve funkcije, prva treba da računa sumu parnih brojeva do **n**, a druga proizvod neparnih brojeva do **n**
7. napisati funkciju koja zamenjuje vrednosti dvema promenljivima
8. učitavati brojeve dok su pozitivni, za svaki od pozitivnih brojeva u funkciji izračunati i ispisati njegov koren

Rekurzivne funkcije

rekurzivne funkcije su funkcije koje pozivaju **same sebe**
primer: računanje faktorijela unetog broja (npr. 3!=3*2*1) **bez rekurzije**

```
int f1 (int n) //n je 3
{
    if (n==0)
        return 1;
    else
        return (n*f2(n-1));
}
int f2 (int n) //n je 2
{
    if (n==0)
        return 1;
    else
        return (n*f3(n-1));}
```

```
int f3 (int n)
{
    if (n==0)
        return 1;
    else
        return (n*f4(n-1));// n je 1
}
int f4 (int n) //n je nula
{
    if (n==0)
        return 1;
    else
        return (n*f5(n-1));
}
```

Faktorijel broja 3 - rekurzivno

```
1.  int f1(int n)
2.  {
3.  if (n==0)
4.      return 1;
5.  else
6.  return (n*f1(n-1));
7.  }
8.  int main ()
9.  {
10. int r:
11. r=f1(3);
12. printf ("tri faktorijel je %d", r);
13. return 0;
14. }
```

dok **main** izvršava liniju **11** poziva **f1(3)**

dok **f1(3)** izvršava liniju **6** poziva **f1(2)**

dok **f1(2**) izvršava liniju **6** poziva **f1(1)**

dok **f1(1)** izvršava liniju **6** poziva **f1(0)**

dok **f1(0)** izvršava liniju **4** (n==0), vraća vrednost **1** funkciji **f1(1)**

f1(1) vraća vrednost **1*1=1** funkciji **f1(2)**

f1(2) vraća vrednost **2*1=2** funkciji **f1(3)**

f1(3) vraća vrednost **3*2=6** funkciji **main**

funkcija **main** u liniji **12** prikazuje broj **6**

Rekurzivne funkcije

1. suma kvadrata od jedan do **n**
2. suma kvadrata od **m** do **n**
   - uzlazna rekurzija
   - silazna rekurzija
3. funkcija računa sumu neparnih brojeva od jedan do zadatog (u **main** funkciji) broja **n**; u **main** funkciji ispisati rezulatat
   - funkcija **main** proverava da li je broj neparan
   - rekurzivna funkcija proverava da li je broj neparan
4. suma brojeva deljivih sa tri od jedan do zadatog u (u **main** funkciji) broja n; u **main** funkciji ispisati rezulatat
   - funkcija **main** proverava da li je broj deljiv sa tri
   - rekurzivna funkcija proverava da li je broj deljiv sa tri
5. naći n–ti Fibonačijev broj (Fibonačijev niz je niz brojeva u kome zbir prethodna dva broja daju vrednost narednog člana)
   0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144…

Organizacija memorije

kada se izvršni program učita u radnu memoriju računara, biće mu dodeljena određena memorija i započinje njegovo izvršavanje; dodeljena memorija sadrži:

- segment koda (**code segment** ili **text segment**)
- segment podataka (**data segment**)
- stek segment (**stack segment**)
- hip segment (**heap segment**)

u **stek** segmentu čuvaju se svi podaci koji karakterišu izvršavanje funkcija; podaci koji odgovaraju jednoj funkciji (ili, preciznije, jednoj instanci jedne funkcije – jer rekurzivna funkcija može da poziva samu sebe i da tako u jednom trenutku bude aktivno više njenih instanci) organizovani su u takozvani stek okvir (**stack frame**);

Organizacija memorije

stek okvir jedne instance funkcije obično, između ostalog, sadrži:

- argumente funkcije;
- lokalne promenljive (promenljive deklarisane unutar funkcije);
- međurezultate izračunavanja;
- adresu povratka (koja ukazuje na to odakle treba nastaviti izvršavanje programa nakon povratka iz funkcije);
- adresu stek okvira funkcije pozivaoca

hip memorija je neiskorišćena memorija programa koja može biti upotrebljena da se u toku izvršavanja programa dinamički zauzme za određene potrebe; programeri ne znaju unapred koliko im je potrebno memorije za čuvanje određenog tipa informacija ili podataka; taj podatak će biti određen tek u toku izvršavanja programa; u toku izvršavanja programa moguće je dinamički alocirati memoriju u hipu korišćenjem **malloc** ili **calloc** naredbi; korišćenjem naredbe **free** izvršiće se oslobađanje memorijskog prostora koji je deo **hip** memorije