PROGRAMIRANJE

PROGRAMSKI JEZIK C

Funkcije

Funkcije

- Funkcija je imenovani niz naredbi koji predstavlja neku logičku celinu
  - potprogram, procedura … u drugim programskim jezicima
  - gradivni blokovi C programa

```
/* Izračunavanje binomnog koeficijenta C(n,k) */
#include <stdio.h>
main() {
    int n, k;
    int nf=1, kf=1, nkf=1;

    printf("Unesite n i k: ");
    scanf("%d%d", &n, &k);

    for(i=2; i<=n; i++) nf*=i; /* izracunava n! */
    for(i=2; i<=k; i++) kf*=i; /* izracunava k! */
    for(i=2; i<=(n-k); i++) nkf*=i; /* izracunava (n-k)! */

    printf("C(%d, %d)=%d\n", n, k, nf/kf/nkf);
}
```

Funkcije

- C program se deli na manje logičke celine
  - lakše za održavanje
  - izbegava se višestruko ispisivanje istog koda
  - funkcije su raspoložive drugim programima
    - kao što je to slučaj sa funkcijama standardnih biblioteka C jezika
- Funkcija u C jeziku
  - ne mora imati argument
  - nije neophodno da izračunava neku vrednost

Definicija i poziv funkcije

```
double average(double a, double b) {
    return (a+b)/2;
}
```

- double na početku predstavlja tip vrednosti rezultata funkcije average
- Identifikatori a, b (parametri funkcije) su dve vrednosti koje se zadaju kada se pozove funkcija average
  - Tip svakog parametra se navodi posebno
  - Parametar funkcije je promenljiva koja se inicijalizuje prilikom poziva funkcije
- Definicija funkcije je izvršni deo funkcije (telo funkcije) i to je blok naredbi
  - Telo funkcije average se sastoji samo od jedne naredbe (naredba return)
    - Izvršavanjem ove naredbe vrši se povratak na mesto u kome je poziv funkcije izvršen
  - Vrednost (a+b)/2 za date a, b će biti vrednost koju ova funkcija izračunava (povratna vrednost funkcije average)
- Funkcija se poziva navodjenjem imena funkcije za kojim sledi lista (stvarnih) argumenata izmedju zagrada

```
average(x,y)
```

Definicija i poziv funkcije

- Efekat poziva funkcije
  - kopiraju se vrednosti (stvarnih) argumenata x i y
  - one postaju vrednosti parametara a i b
  - Izvršava se telo funkcije
- Argument u pozivu funkcije ne mora bti promenljiva već to moze biti izraz odgovarajućeg tipa

```
average(5.1, 8.9)
average(x/2, y/3)
```

- Poziv funkcije se vrši u trenutku kada je potrebna njena povratna vrednost

```
printf("Average: %g\n", average(x, y));
```
← izračunava srednju vrednost od x i y i štampa je

- Rezultat funkcije average nije sačuvan (štampa se i potom odbacuje)
- Ako je potrebno može se sačuvati povratna vrednost funkcije

```
avg= average(x, y);
```
← izračunata vrednost se dodeljuje nekoj promenljivoj avg

- Funkcija se može pozivati proizvoljan broj puta

Definicija i poziv funkcije

- Ne mora funkcija uvek izračunavati neku vrednost
  - Činjenica da funkcija ne vraća nikakvu vrednost se specificira tako što joj se pridružuje prazan tip podataka void

```
void odbrojavanje(int n) {
    printf("%d\n", n);
}
```

```
void main() {
    int i;
    for(i=10; i>0; --i) odbrojavanje(i);
}
```

    - void je tip podataka koji ne uključuje nijednu vrednost
    - Funkcija ne vraća nikakvu vrednost
      - ispuštena je naredba return u telu funkcije
    - Dalje kako funkcija ne vraća nikakvu vrednost
      - Poziv funkcije ne može biti deo izraza
      - Sam poziv funkcije je naredba

Definicija i poziv funkcije

- Dalje funkcija ne mora imati parametre

```
void print_zdravo(void) {
        printf("Zdravo Svete!\n");
}
```

  - void na mestu liste parametara funkcije eksplicitno specificira da funkcija nema parametara
  - Poziv funkcije bez argumenata

```
print_zdravo();
```

    - Zagrade su obavezne iako poziv funkcije nema argumenata

Definicija funkcije

```
Povratni_tip ime_funkcije(lista parametara) {
    deklaracije
    naredbe
}
```

- Povratna vrednost funkcije ne može biti niz
- Ako je tip vrednosti funkcije void funkcija ne vraća vrednost
- Svakom parametru u listi prethodi tip vrednosti parametra

```
double average(double a, b)
```

pogrešno

- Telo funkcije je blok
  - Blok može sadržati i deklaracije (navode se pre bilo koje naredbe)

```
double average(double a, double b) {
    double sum;
    sum=a+b;
    return (a+b)/2;
}
```

Definicija funkcije

- Promenljive deklarisane u telu funkcije
  - Vidljive su (može im se pristupati) samo u toj funkciji (telu funkcije)
- Telo funkcije čiji je tip vrednosti void može biti prazno

```
void do_nothing() {
}
```

Poziv funkcije

- Sastoji se iz imena funkcije za kojim sledi lista argumenata u zagradama ()

```
average(x, y);
odbrojavanje(i);
print_zdravo();
```

- Ako funkcija nema parametara lista argumenata je prazna (zagrade su obavezne)

```
print_zdravo;
```
← pogrešno

Vrednost koju vraća ne-void funkcija je moguće odbaciti

```
average(x, y);
```

  - printf funkcija pored toga sto vrši formatirani ispis, vraća broj odštampanih karaktera

    ```
    num_chars=printf("Ciao!\n");
    ```
    ili samo
    ```
    printf("Ciao!\n");
    ```

  - Ovo se može i eksplicitno naglasiti cast operatorom

    ```
    (void)printf("Ciao!\n");
    ```

Poziv funkcije

```
#include <stdio.h>
typedef int Bool;
#define TRUE 1
#define FALSE 0
Bool is_prime(int n) {
    int divisor;
    if(n<=1) return FALSE;
    for(divisor=2; divisor*divisor<=n; divisor++)
        if(n%divisor == 0) return FALSE;
    return TRUE;
}
```

```
void main(void) {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if(is_prime(n)) printf("Prime\n");
    else printf("Not prime\n");
}
```

- U funkciji main deklarisana je promenljiva n sa istim imenom kao i parameter funkcije is_prime
  - Promenljive deklarisane u različitim funkcijama reprezentuju različite memorijske lokacije
  - Ovo važi i za parametar funkcije
    - Parametar funkcije je promenljiva koja se kreira pozivom funkcije i inicijalizuje se vrednošću odgovarajućeg argumenta

Deklaracija funkcije

```
#include <stdio.h>
main() {
   double x, y, z;
   printf("Enter three numbers: ");
   scanf("%lf%lf%lf", &x, &y, &z);
   printf("Average of %g and %g: %g\n", x, y, average(x, y));
   printf("Average of %g and %g: %g\n", x, z, average(x, z));
   printf("Average of %g and %g: %g\n", y, z, average(y, z));
}

double average(double a, double b) {
   return (a+b)/2;
}
```

- Kada kompajler naiđe na poziv funkcije average on nema nikakvu informaciju o
  - Tipu funkcije
  - Tipu argumenata

Deklaracija funkcije

- Definicija funkcije ide pre dela programa u kome je izvršen poziv funkcije
- Deklaracija (prototip) funkcije se zadaje pre njenog korišćenja
  - Daje sažet opis funkcije čija definicija ide kasnije
    - tip vrednosti same funkcije
    - tipove vrednosti svih njenih parametara
  - Po strukturi je slična prvoj liniji definicije funkcije

```
return_type function_name(parameters);
```

Deklaracija funkcije

```
#include <stdio.h>

double average(double a, double b);

main() {
    double x, y, z;
    printf("Enter three numbers: ");
    scanf("%lf%lf%lf", &x, &y, &z);
    printf("Average of %g and %g: %g\n", x, y, average(x, y));
    printf("Average of %g and %g: %g\n", x, z, average(x, z));
    printf("Average of %g and %g: %g\n", y, z, average(y, z));
}
double average(double a, double b) {
    return (a+b)/2;
}
```

- Imena parametara se mogu izostaviti
- Navođenje tipa svakog parametra je obavezno

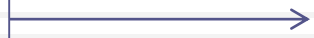
```
double average(double, double);
```

Argumenti funkcije

- Koja je razlika izmedju parametara i argumenata funkcije?
  - Parametri figurišu u definiciji funkcije
    - Parametri su fiktivna imena koja predstavljaju vrednosti koje će biti zadane kada se funkcija pozove
  - Argumenti su izrazi koji se zadaju u pozivu funkcije
    - U C jeziku argumenti se prenose po vrednosti
    - Kada se funkcija pozove izračunavaju se vrednosti argumenata i dodeljuju se odgovarajućim parametrima
  - Parameter sadrži kopiju vrednosti odgovarajućeg argumenta
    - Promena vrednosti parametra nema efekta na vrednost odgovarajućeg argumenta

Argumenti funkcije

```
int power(int x, int n) {
    int i, result=1;
    for(i=1; i<=n; i++) result*=x;
    return result;
}
```



```
int power(int x, int n) {
    int result=1;
    while(n-- >0) result*=x;
    return result;
}
```

- Parametar n će biti kopija originalnog argumenta
  - Može se bezbedno menjati u funkciji

```
void decompose(double x, long int_part, double frac_part) {
    int_part=(long)x;
    frac_part=x-int_part;
}
```

```
…
void main(void) {
    long i;
    double x, d;
     …
    decompose(x, i, d);
     …
}
```

- Funkcija ne može imati dve povratne vrednosti
- Na gornji način se ne može dobiti očekivani rezultat
  - Promene na parametrima int_part i frac_part nemaju efekta na promenljivim i odnosno d koje su argumenti poziva funkcije decompose

Niz kao argument funkcije

- U slučaju da je jednodimenzionalni niz parameter funkcije, maksimalna predvidjena dužina niza se može ispustiti (niz se ne kopira, funkciji je dostupan originalni niz)

```
int f(int a[MAXN]) {
    ...
}
```

←

```
int f(int a[MAXN]) {
    ...
}
```

- Kako u telu funkcije f izvući informaciju o stvarnoj dužini niza koji je zadat kao argument u pozivu funkcije?
  - Dužina niza koji je parametar funkcije se takođe može proslediti kao parameter funkcije

```
int sum_array(int a[], int n) {
    int i, sum=0;
    for(i=0; i<n; i++) sum+=a[i];
    return sum;
}
```

  - Prototip bi bio

```
int sum_array(int a[], int n);
```

odnosno

```
int sum_array(int [], int);
```

Niz kao argument funkcije

- U slučaju da je string (niz karaktera) parameter funkcije, dužina stringa se ne mora prenositi kao parameter (implicitno je sadržana u samom nizu, određena je pozicijom null-karaktera)

```
int strlen(char str[]) {
    int n=0;
    while(str[n] !='\0') n++;
    return n;
}
```

```
while(str[n++] !='\0') ;
```

```
while(str[n++]) ;
```

Niz kao argument funkcije

- Funkcija može menjati elemente niza koji je prosleđen kao argument

```
void store_zeros(int a[], int n) {
    int i;
    for(i=0; i<n; i++) a[i]=0;
}
```

U C jeziku se nizovi u funkciju prenose kao memorijske reference
(prenosi se adresa prvog elementa niza),
pa se elementi niza mogu menjati u funkciji

- Niz se može proglasiti konstantnim ako se u deklaraciji niza navede ključna reč const

```
double sum_1d_array(const double a[], int n) {
    int i, sum=0;
    for(i=0; i<n; i++) sum+=a[i];
    return sum;
}
```

Naredba return

- Ne void funkcija mora naredbom return zadati vrednost koju funkcija vraća

```
return expression;
```

- Ako je tip izraza u naredbi return različit od tipa funkcije (povratne vrednosti funkcije) izvršiće se odgovarajuća konverzija

```
return;
```

return naredba void funkcije

  - Korišćenje return naredbe na kraju tela void funkcije nije neophodno jer se izvršava automatski kada se završi telo void funkcije

```
void print_int(int n) {
    if(i<0) return;
    printf("%d\n", n);
}
```

- Ako se izvršavanjem ne-void funkcije dođe do kraja tela funkcije i pri tome se ne izvrši return naredba ponašanje je nedefinisano ako se na mestu poziva koristi povratna vrednost funkcije

Kraj programa

- I main funkcija je nekog tipa
  - Vrednost koju vraća main funkcija je obično status izvršavanja programa (status kod)
    - Koji se dalje može analizirati
    - Uobičajeno je da povratna vrednost main funkcije (status kod) bude
      - 0 u slučaju normalnog završetka programa
      - Vrednost različita od 0, u slučaju neželjenog završetka rada (kod greške)
- Ako se ipusti tip funkcije
  - Podrazumeva se da je int
  - U standardu C99 ispuštanje tipa funkcije nije dozvoljeno
- Ispuštanje ključne reči void u listi parametara funkcije kada je lista prazna je dozvoljena
  - main funkcija obično ima dva parametra sa imenima argc, argv

exit funkcija

- Izvršavanje return naredbe u main funkciji je jedan način završavanja programa
- exit funkcija iz stdlib biblioteke ima sličan efekat
  - exit funkcija ima jedan argument koji je celobrojni izraz

  `exit(izraz);` ⟷ `return izraz;`
  - normalan završetak: exit(0); odnosno exit(EXIT_SUCCESS);
  - nezeljeni zavrsetak: exit(EXIT_FAILURE);
  - EXIT_SUCCESS I EXIT_FAILURE: simboličke konstante definisane u stdlib.h
- Razlika izmedju exit funkcije i naredbe return
  - exit funkcija prekida izvršavanje programa bez obzira u kojoj funkciji je poziv izvršen
  - naredba return prekida izvršavanje programa samo ako se izvrši u main funkciji

Rekurzija

- Funkcija je rekurzivna ako poziva samu sebe

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

$$x^n = x \cdot x^{n-1}$$

```
int fact(int n) {
    if(n<=1) return 1;
    return n*fakt(n-1);
}
```

```
double power(double x, int n) {
    return n==0?1:x*power(x, n-1);
}
```