Funkcije i makroi

Studijski program matematika

Programiranje 1

Pregled

Pregled

Funkcije

- Svaki C program sačinjen je od funkcija.
- Funkcija `main` mora da postoji i, pojednostavljeno rečeno, izvršavanje programa uvek počinje izvršavanjem ove funkcije.
- Iz funkcije `main` (ali i drugih) pozivaju se druge funkcije, bilo bibliotečke (poput funkcije `printf`), bilo korisnički definisane .
- Funkcije se koriste se da bi kôd bio kraći, čitljiviji, modularniji, šire upotrebljiv, itd.
- U funkciju se obično izdvaja neko izračunavanje, neka obrada koja predstavlja celinu za sebe i koristi se više puta u programu.

Zašto pišemo funkcije?

- Ponovno korišćenje koda.
- Mogućnost upotrebe čak i kada se ne razume implementacija.
- Čitljivost koda.
- Lakše održavanje koda.
- Lakše testiranje i debagovanje koda.
- Korišćenje istih imena promenljivih.

Kako pišemo funkcije?

- Uočiti logičke celine.
- Uočiti parametre i povratnu vrednost.
- Dati ime koje odgovara implementaciji.
- Implementirati funkciju.

Primer

```
#include <stdio.h>
int kvadrat(int n);

int main() {
    printf("Kvadrat broja %i je %i\n", 5, kvadrat(5));
    printf("Kvadrat broja %i je %i\n", 9, kvadrat(9));
    return 0;
}

int kvadrat(int n) {
    return n*n;
}
```

Primer

```
#include <stdio.h>
int max = 10; /*Globalna promenljiva*/
float stepen(float x, unsigned n);
int main() {
  unsigned i; /*Lokalna promenljiva*/
  for(i = 0; i < max; i++)
     printf("%d %f %f\n", i, stepen(2.0f,i), stepen(3.0f,i));
   return 0;
}
float stepen(float x, unsigned n) {
   unsigned i;     /*Lokalna promenljiva*/
   float s = 1.0f; /*Lokalna promenljiva*/
   for(i = 1; i<=n; i++)
     s *= x;
   return s;
}
```

Deklaracija i definicija funkcije

- *Deklaracija* (ili *prototip*) funkcije ima sledeći opšti oblik:

```
tip ime_funkcije(niz_deklaracija_parametara);
```

- *Definicija* funkcije ima sledeći opšti oblik:

```
tip ime_funkcije(niz_deklaracija_parametara) {
   deklaracije
   naredbe
}
```

Definicija i deklaracija

- Definicija funkcija mora da bude u skladu sa navedenim prototipom, tj. moraju da se podudaraju tipovi povratne vrednosti i tipovi parametara.
- Deklaracija ukazuje prevodiocu da će u programu biti korišćena funkcija sa određenim tipom povratne vrednosti i parametrima određenog tipa.
- Zahvaljujući tome, kada prevodilac (na primer, u okviru funkcije `main`), naiđe na poziv funkcije `kvadrat`, može da proveri da li je njen poziv ispravan (čak iako je definicija funkcije nepoznata u trenutku te provere).

Deklaracija

- Pošto prototip služi samo za proveravanje tipova u pozivima, nije neophodno navoditi imena parametara, već je dovoljno navesti njihove tipove. Na primer:

```
int kvadrat(int);
```

- Nije neophodno za svaku funkciju navoditi najpre njen prototip, pa onda definiciju. Ukoliko je navedena definicija funkcije, onda se ona može koristiti u nastavku programa i bez navođenja prototipa

Parametri funkcije

- Funkcija može imati parametre koje obrađuje i oni se navode u okviru definicije, iza imena funkcije i između zagrada.
- Termini *parametar funkcije* i *argument funkcije* se ponekad koriste kao sinonimi.
- `n` je parametar funkcije `int kvadrat(int n);`, a `5` i `9` su njeni argumenti u pozivima `kvadrat(5)` i `kvadrat(9)`.
- Parametri funkcije mogu se u telu funkcije koristiti kao lokalne promenljive te funkcije a koje imaju početnu vrednost određenu vrednostima argumenata u pozivu funkcije.

Parametri funkcije

- Kao i imena promenljivih, imena parametara treba da oslikavaju njihovo značenje i ulogu u programu.
- Pre imena svakog parametra neophodno je navesti njegov tip.
- Ukoliko funkcija nema parametara, onda se između zagrada ne navodi ništa ili se navodi ključna reč `void`.
- Promenljive koje su deklarisane kao parametri funkcije lokalne su za tu funkciju i njih ne mogu da koriste druge funkcije.
- Bilo koja druga funkcija može da koristi isto ime za neki svoj parametar ili za neku svoju lokalnu promenljivu.

Prenos po vrednosti

- Na mestu u programu gde se poziva neka funkcija kao njen argument se može navesti promenljiva, ali i bilo koji izraz istog tipa (ili izraz čija vrednost može da se konvertuje u taj tip).
- Na primer, funkcija `kvadrat` može biti pozvana sa `kvadrat(5)`, ali i sa `kvadrat(2+3)`.
- Argumenti funkcija se uvek prenose *po vrednosti*.
- To znači da se vrednost koja se koristi kao argument funkcije *kopira* kada počne izvršavanje funkcije i onda funkcija radi samo sa tom kopijom, ne menjajući original.

Primer

```
#include <stdio.h>

void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 3, y = 5;
    swap(x, y);
    printf("x = %d, y = %d\n", x, y);
}
```

Poklapanje imena

Moguće je i da se ime parametra funkcije poklapa sa imenom promenljive koja je prosleđena kao stvarni argument. Na primer:

```
#include <stdio.h>
void f(int a) {
  a = 3;
  printf("f: a = %d\n", a);
}
int main() {
   int a = 5;
   f(a);
   printf("main: a = %d\n", a);
}
```

Konverzije tipova

Prilikom poziva funkcije, ukoliko je poznata njena deklaracija, vrši se implicitna konverzija tipova argumenata u tipove parametara (ako se oni razlikuju).

```
#include <stdio.h>

void f(int a) {
  printf("%d\n", a);
}

int main() {
  f(3.5);
}
```

Povratna vrednost funkcije

- Funkcija može da vraća rezultat i `tip` označava tip vrednosti koja se vraća kao rezultat.
- Funkcija rezultat vraća naredbom `return r;` gde je `r` izraz zadatog tipa ili tipa koji se može konvertovati u taj tip.
- Naredba `return r;` ne samo da vraća vrednost `r` kao rezultat rada funkcije, nego i prekida njeno izvršavanje.

Povratna vrednost funkcije

- Ako funkcija ne treba da vraća rezultat, onda se kao tip povratne vrednosti navodi specijalan tip `void` i tada naredba `return` nema argumenata (tj. navodi se `return;`).
- Funkcija koja je pozvala neku drugu funkciju može da ignoriše, tj. da ne koristi vrednost koju je ova vratila.

Rekurzija

- Funkcije mogu da pozivaju druge funkcije.
- Funkcija može da pozove i sama sebe

```
double stepen(double x, unsigned n) {
  if (n == 0)
    return 1;
  else
    return x*stepen(x, n-1);
}
```

Rekurzija

- Funkcije mogu da pozivaju druge funkcije.
- Funkcija može da pozove i sama sebe

```
double stepen(double x, unsigned n) {
  if (n == 0)
    return 1;
  else
    return x*stepen(x, n-1);
}
```

izračunava vrednost x^n. Na primer, `stepen(2.0, 3)` = `2.0*stepen(2.0, 2)` = `4.0*stepen(2.0, 1)` = `8.0*stepen(2.0, 0)` = `8.0*1` = `8.0`.

Funkcije sa promenljivim brojem argumenata

- Ponekad je korisno definisati funkcije koje u pozivu mogu da imaju promenljiv broj argumenata.
- Primetimo da su takve funkcije `printf` i `scanf` a takve mogu biti i korisnički definisane funkcije.
- U programskom jeziku C postoji podrška za pisanje funkcija sa promenljivim brojem argumenata

Pregled

Makroi

Dve najčešće korišćenje pretprocesorske direktive su:

- `include` koja uključuje sadržaj neke druge datoteke;
- `define` koja zamenjuje neki tekst, **makro**, drugim tekstom.

Opšti oblik:

```
#define originalni_tekst novi_tekst
```

Najjednostaniji primer:

```
#define MAX_LEN 80
```

- `MAX_LEN` je samo simboličko ime i nikako ga ne treba mešati sa promenljivom (čak ni sa promenljivom koja je const).
- Za ime `MAX_LEN` u memoriji se ne rezerviše prostor tokom izvršavanja programa.

Makroi

- Makro može biti bilo koji identifikator, pa čak i ključna reč jezika C.
- Zamene se ne vrše u konstantnim niskama niti u okviru drugih simboličkih imena.
- Na primer, direktiva `#define MAX_LEN 80` neće uticati na naredbu `printf( "MAX_LEN is 80");` niti na simboličko ime `MAX_LEN_VAL`.
- Pretprocesor omogućava i definisanje makroa sa argumentima, kao i uslovno prevodenje (odredeni delovi izvornog programa se prevode samo ukoliko su ispunjeni zadati uslovi).

Makroi sa argumentima

- Moguće je defisati i pravila zamene sa argumentima od kojih zavisi tekst zamene.

```
#define max(A, B) ((A) > (B)? (A) : (B))
```

- Ukoliko je u nastavku datoteke, u nekoj naredbi navedeno `max(2, 3)`, to će, pre prevodenja, biti zamenjeno sa `((2) > (3) ? (2) : (3))`.
- Tekst `max(x+2, 3*y)` biće zamenjen tekstom `((x+2) > (3*y) ? (x+2) : (3*y))`.
- Tekst `max(2, 3)` i slični ne predstavljaju poziv funkcije i nema nikakvog prenosa argumenata kao kod pozivanja funkcija.

Makroi sa argumentima

- Ukoliko je negde u programu navedeno `max(a++, b++)`, na osnovu date definicije, ovaj tekst biće zamenjen tekstom `((a++) > (b++) ? (a++) : (b++))`, što će dovesti do toga da se veća od vrednosti a i b inkrementira dva puta.

Makroi sa argumentima

- Ukoliko se definicija
  `#define kvadrat(x) x*x`
  primeni na `kvadrat(a+2)`, tekst zamene će biti `a+2*a+2`, a ne `(a+2)*(a+2)`.
- Zbog toga bi trebalo koristiti:
  `#define kvadrat(x) (x)*(x)`
  kada se makro primeni na izraz `a/kvadrat(b)`, on će biti zamenjen izrazom `a/(b)*(b)`, što je ekvivalentno sa `(a/b)*b` (a ne sa `a/(b*b)`).
- Zbog toga je bolja definicija:
  `#define kvadrat(x) ((x)*(x))`

Literatura

Slajdovi su pripremljeni na osnovu materijala iz osmog i devetog poglavlja knjige:
Predrag Janičić, Filip Marić: Programiranje 1.
Nastali su dopunom slajdova prof. dr Milene Vujošević Janičić.

Za pripremu ispita nisu dovoljni slajdovi, potrebno je koristiti knjigu!