Programiranje I

Funkcije printf i scanf

Kontrola toka programa

Pokazivači - osnovno

Funkcije printf i scanf

- Dvije funkcije bez kojih je teško zamisliti početak rada u programskom jeziku C:
  - scanf (za unos podataka sa tastature) i
  - printf (za prikaz rezultata na monitoru)

  nijesu dio osnovne definicije jezika C. Da bi se sa njima radilo mora se uključiti programska biblioteka stdio.h **(može se dogoditi da kompajler radi i ako ova biblioteka nije uključena).**

- Biblioteka se uključuje pretprocesorskom naredbom:

  #include <stdio.h>

  Kompajler izvršava pretprocesorske naredbe prije bilo koje druge naredbe u programu.

  Sve pretprocesorske naredbe počinju ovim simbolom, pa se ponekad nazivaju tarabama. U posljednje vrijeme ovaj simbol se i kod nas naziva šarp simbol.

Funkcija printf

- O pretprocesoru ćemo kasnije reći znatno više.
- Funkcija printf se može javiti u dva oblika. Prvi je jednostavniji:
  - printf("neki tekst u navodnicima sa specijalnim karakterima \n");

Poruka bi bila odštampana na ekranu u datom obliku.

Znak za novi red je čest element funkcije printf.

- Drugi oblik je znatno značajniji:
  - printf("%d %d %e tekst %c\n", x, 60, y, z);

Funkcija printf u ovom obliku štampa string koji joj je prvi argument, ali ono što je iza procenata mijenja se redom sa argumentima koji slijede. Tako se umjesto %d štampa promjenljiva x i to kao cjelobrojna (%d znači štampaj kao cjelobrojnu promjenljivu).

Funkcija printf

- Očekuje se da broj argumenta nakon stringa u funkciji printf bude jednak broju "procenata" unutar stringa.
- Ti argumenti mogu biti promjenljive, izrazi, konstante i pozivi funkcija.
- Procentualne oznake imaju sljedeće značenje (navodimo samo neke važnije):
  - %d i %i cijeli brojevi,
  - %f realni brojevi,
  - %s stringovi,
  - %bd cijeli broj sa b cifara,
  - %a.bf realni broj sa a pozicija i b decimalnih mjesta, podrazumjeva se a>b (pozicije uključuju predznak, tačku i sve cifre).

Funkcija printf

- Nastavljamo sa značenjem procenata:
  - %e eksponencijalni zapis realnih brojeva
  - %g realni brojevi u kompaktnom zapisu (završne nule nisu uključene i ne prikazuje se decimalna tačka kod cijelih brojeva)
  - %x heksadecimalni zapis cijelog broja
  - %o oktalni zapis cijelog broja
  - %p pokazivač
- Rekli smo da argument funkcije može biti izraz. Na primjer:
  printf("%d\n", j++); /*štampa j, a zatim uvećava j za 1*/
- Prilikom upotrebe izraza kao argumenata oprez! Na primjer:
  j=1;
  printf("%d %d\n", j, j++); /*štampa 2 pa 1*/

Po pravilu, prvo se predaje drugi argument (j++), kod njega dođe do uvećavanja j i tek se onda preda prvi argument.

Funkcija scanf

- Funkcija scanf, koja se koristi za unos podataka sa tastature, ima sljedeću sintaksu:

```
scanf("%c%d", &ch, &cj);
```

Specijalni simboli koji označavaju kojeg su tipa promjenljive koje se učitavaju. U ovom slučaju: karakter i cijeli broj.

&ch i &cj označavaju adrese promjenljivih ch i cj, a ono što se unese sa tastature smjesti na adresu ovih promjenljivih.

Podrazumjeva se da je broj procenata jednak broju adresa. Podrazumjeva se da su procenti na pravilan način upareni sa tipovima promjenljivih koji se učitavaju. Ako ova dva "podrazumjevanja" nijesu ispunjena program vam i dalje može raditi, ali šta će zapravo raditi i kakve će promjenljive učitati to nije sigurno. U svakom slučaju je bolje da vam program "pukne" na početku izvršavanja nego da završi sa izvršavanjem i produkuje besmislene rezultate.

Funkcija scanf i unos podataka

- Funkcija scanf čeka da unesete potreban broj podataka.
- Ova funkcija zanemaruje bjeline i učitava samo podatke do unosa potrebnog broja podataka i pritiska na Enter taster. Na primjer, ako se očekuje unos 2 podatka, a vi unesete 1 i pritisnete Enter, neće se nastaviti dalje izvršavanje programa, već će se i dalje čekati na dodatni podatak.
- Šta se dešava ako unesete tri podatka, a traže se dva? Prva dva podatka se iskoriste, dok se treći čuva u memoriji koja se zove **bafer** i taj podatak je spreman da bude iskorišćen za naredni unos.

Scanf - primjer

- scanf("%d %d",&a,&b); /*ako unesete 2 3 4*/
  ... /*2 i 3 se dodjeljuju promjenljivim a i b*/
- scanf("%d",&c); /*sada će nam se učiniti da je ova naredba*/
  /*preskočena, a zapravo je zaostalo 4*/
  /*iskorišćeno i smješteno u c*/
- Postoji način da se ovakav rad izbjegne, ali o tom potom.
- Prije scanf funkcije se često postavlja printf koja objašnjava korisniku koji se podatak od njega očekuje:

```
printf("Unesite cijeli broj N:\n");
scanf("%d",&N);
```

Struktura programa

- Program u C-u se sastoji od:
  - pretprocesora (taraba),
  - deklaracija promjenljivih i funkcija, i
  - funkcija.
- Pretprocesorske naredbe počinju sa **#** i svaka se nalazi u posebnom redu. Ove naredbe (direktive) važe do kraja teksta programa i kompajler ih izvršava prije bilo koje druge naredbe.
- Izvršavanje programa startuje od glavnog programa (funkcije **main**).

Oblik funkcije u C-u

- Tijelo funkcije se oivičava velikim zagradama **{}** i sastoji se od deklaracija promjenljivih koje se obično postavljaju na početku bloka i naredbi koje slijede. Blok naredbi u programskom jeziku C je sekvenca naredbi oivičena sa **{}**.
- Prije tijela funkcije se postavlja zaglavlje koje sadrži ime funkcije, tip rezultata, kao i tip i imena pojedinih argumenata.

```
zaglavlje() {
        deklaracija1;
        deklaracija2;
        ...
        naredba1;
        naredba2;
        ...
}
```

Struktura funkcije

- Za sada je glavni program jedina funkcija koju ćemo koristiti.

Pregled naredbi u C-u

Postoji samo 13 osnovnih naredbi, što se smatra prednošću C-a.

- Naredba pridruživanja =
- Poziv funkcije koji se obavlja u obliku ime_funkcije(argumenti);
- Složena ili blok naredba koja je zapravo skup naredbi oivičen velikim zagradama (tretira se kao jedna) {nar1; nar2; nar3; ... narN;}
- Prazna naredba ;
- Osam naredbi za kontrolu toka:
  - if
  - switch
  - while
  - do
  - for
  - break
  - continue
  - goto
- Vraćanje rezultata funkcije naredbom return.

Pridruživanje

- Pridruživanje se u C-u obavlja preko operatora =.
- Sa lijeve strane mora biti dozvoljena lijeva vrijednost (lvalue) kojoj se može pridružiti vrijednost izraza sa desne strane, dok sa desne može biti izraz, poziv funkcije, promjenljiva ili konstanta.
- C jezik dozvoljava pridruživanje tipa:
  a=b=c;
- Sada promjenljive a i b moraju biti dozvoljene lijeve strane izraza. Ovakav način pridruživanja drugi programski jezici rijetko podržavaju.

Uslovno izvršavanje if

- Naredba uslovnog izvršavanja ima oblik:
  if(uslov) naredba ili blok naredbi;
- Ako je uslov logički tačan (različit od nule, podsjetite se smisla logičke tačnosti u C-u) izvršava se naredba ili blok naredbi. Ako se izvršava jedna naredba ne moraju se koristiti vitičaste zagrade, a ako se izvršava više naredbi postavljaju se vitičaste zagrade oko njih (blok naredbi).
- if naredba, sa onim što se izvršava ako je uslov zadovoljen, tretira se kao jedna naredba.
- OPREZ!
  if(uslov) ; /*ispravna naredba, ali vjerovatno ne radi ono što ste zamislili*/

  → a evo i razloga!

if varijante

- Varijanta naredbe if u obliku:
  if(uslov) naredba1 ili blok naredbi1;
  else naredba2 ili blok naredbi2;
- Izvršava naredbu1 ili blok naredbi1 ako je ispunjen uslov, a ako nije naredbu2 ili blok naredbi 2.
- Varijanta:
  if(uslov1) N1;
  else if(uslov2) N2;
  else if(uslov3) N3;
  ...
  else Nn;

izvršava N1 ako je ispunjen uslov1, a ako nije ispunjen uslov1, a ispunjen je uslov2 izvršava N2 itd, a ako nijedan od uslova nije ispunjen izvršava Nn.

Obratite pažnju da C ne posjeduje elseif naredbu, već da je if ugnježdeno u else dijelu.

Postoji bjelina!

switch-case selekcija

- switch-case naredba ima sličnu funkciju kao if, pri čemu se navode sve moguće vrijednosti kontrolnog izraza (case-ovi).

```
switch(cjelIzr)
{
      case v1: nar. ili blok 1; break;
      case v2: nar. ili blok 2; break;
      ...
      case vN: nar. ili blok N; break;
      default: nar. ili blok Q;
}
```

Ako je cjelobrojni izraz cjelIzr jednak v1 izvršava se nar. ili blok 1, ako je jednak v2 izvršava se nar. ili blok 2 itd. Ako nije jednaka nijednoj konstanti vi, i=1,...,N, izvršava se default dio.

Pored cjelobrojnih izraza i cjelobrojnih promjenljivih, cjelIzr mogu biti i karakteri!

switch-case selekcija

- default dio se može izostaviti. Ako bi se izostavio, a da ne postoji nijedan case dio koji odgovara vrijednosti cjelobrojnog izraza, blok bi bio preskočen.
- Ako se na kraju naredbe ili bloka naredbi preskoči ključna riječ break izvršio bi se blok naredbi u narednom case dijelu bez provjere uslova**!!!** Ovakav način izvršavanja ovog bloka naziva se propadanje.
- Ovo može ponekad valjati, ali obično nije ono što se traži i da bi se nakon izvršenja traženog bloka naredbi izašlo iz switch-case bloka treba obavezno postaviti break naredbu.
- default dio ne mora biti na kraju bloka (može čak i na početku), ali ga onda mora pratiti break ako se želi izbjeći propadanje.

while ciklus

- Oblik while ciklusa je veoma sličan while ciklusu kojeg smo vidjeli u MATLAB-u.

  while (uslov)
  naredbe ili blok naredbi;

  Ako je logički uslov tačan (različit od nule) izvršava se naredba ili blok naredbi.

- Oblik

  while(uslov) ;

  je sintaksno ispravan, ali logički vjerovatno nije, jer ako je uslov tačan izvršiće se prazna naredba, odnosno neće se dogoditi ništa, što vjerovatno nije bio cilj programera.

do-while ciklus

- Drugi tip ciklusa je do-while koji ima sintaksu:
  do
  naredba ili blok naredbi;
  while (uslov);
- naredba ili blok naredbi se izvršavaju dok je logički uslov ispunjen (različit od nule).
- Osnovna razlika u odnosu na while ciklus je ta što se ovaj ciklus izvršava barem jednom, tj. tek nakon prvog prolaska kroz tijelo ciklusa provjerava se ispravnost logičkog uslova.
- Obratite pažnju da se do-while blok tretira kao jedna naredba, pa se mora završiti sa **;** jer se ne završava sa naredbom iz bloka (koja se uvijek završava sa **;**) ili sa vitičastom zagradom.

for ciklus

- for ciklus se znatno razlikuje od onog u MATLAB-u, iako ima istu namjenu – izvršavanje bloka naredbi tačno određen broj puta.
- Oblik for ciklusa je:

```
for(izraz1; izraz2; izraz3)
      naredba ili blok naredbi;
```

izraz1 služi za postavljanje početne vrijednosti brojača u petlji, izraz3 opisuje što se sa brojačem dešava u okviru petlje, tj. način na koji se on mijenja. Naredba ili blok naredbi se izvršava sve dok je izraz2 ispunjen.

for ciklus

- Primjer:

```
for(i=0;i<5;i++)
      naredba ili blok naredbi;
```

  Prvi put se ciklus izvršava za vrijednost brojača i=0, nakon te prve upotrebe brojač se inkrementira i ciklus se izvršava za i=1, i=2, i=3 i i=4. Kada brojač dostigne vrijednost i=5 logički uslov nije ispunjen i izvršavanje ciklusa se prekida. Dakle, ovaj ciklus se izvršava 5 puta za i=0, i=1, i=2, i=3 i i=4.

- Ciklusi se mogu ugnježdavati, sadržati druge cikluse i obratno. Pravila kao u MATLAB-u: prvo se zatvara unutrašnja petlja, zatim spoljašnja!

for ciklus

- Primjer ugnježdenog ciklusa:

```
S=0;
for(i=0;i<6;i++)
        for(j=1;j<=4;j+=2)
                S+=i+j;
```

Koliko iznosi S nakon ovih naredbi?

Pogledajmo sljedeći oblik ciklusa:

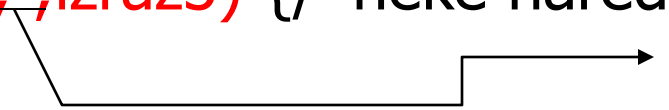
```
for(i=0,j=0; i<10; i++,j++)
```

Na zgodan način smo inicijalizovali dva brojača i svaki od njih inkrementiramo u prolazu petlje. Zapamtite da ovo nije ugnježdena, već jednostruka petlja.

Beskonačni ciklusi

- Ciklusi tipa:

  `while(1)  {/*neke naredbe*/}`
  ili
  `for(izraz1; ;izraz3) {/*neke naredbe*/}`

  nema uslova koji provjerava ispravnost

  nazivaju se **beskonačnim**.
- Sintaksno su ispravni (kompajler će ih prihvatiti), ali su najčešće logički neispravni.
- Postoje, ipak, situacije kada se ovakvi ciklusi koriste, a najčešće je to u radu sa hardverom, a posebno perifernim uređajima, što izlazi van okvira kursa.

Naredbe break , continue i goto

- Naredba break prekida tekući ciklus i naredna naredba je prva nakon tekućeg ciklusa pri čemu, ukoliko imamo slučaj ugnježdenih petlji i break se nalazi u unutrašnjoj, sa break se izlazi samo iz unutrašnje petlje, a ne i iz spoljašnje.
- Naredba continue prekida tekuće izvršavanje u ciklusu **(ponekad se kaže tekuću iteraciju)** i počinje novo izvršavanje (naredna naredba nakon continue je, recimo, provjera da li je zadovoljen logički uslov u while ili for dijelu).
- Naredba goto ima sintaksu:
  goto labela; //labela je ispravno deklarisano ime u programu
- Naredna naredba koja se izvršava nakon goto je označena kao:
  labela: naredba;

Upotreba continue i goto

- Već smo vidjeli da se break sintaksno mora koristiti u C-u kod switch-case kombinacije.
- Još 1966 Bohm i Jacopini su dokazali da se svi algoritamski rješivi problemi mogu riješiti pomoću jedne sekvence, ciklusa i selekcije.
- To je postao standard naredbi za kontrolu toka programa.
- Programerska praksa pokazuje da su programi koji se pišu sa continue, break i goto veoma teški za održavanje (kasnije korišćenje i prepravljanje).
- Stoga se preporučuje izbjegavanje ovih naredbi kad god je to moguće!
- Posebno se ne preporučuje korišćenje naredbe goto!!!

Pokazivači

- Prvi složeni (izvedeni) tip podataka kojeg uvodimo je pokazivač (pointer).
- Ovo je veoma moćan, ali često i veoma nerazumljiv koncept.
- Moramo pokušati da ga bolje objasnimo, a vi morate uložiti nemali trud da ga shvatite :-)
- Komplikacije sa pokazivačima će se nastaviti kad god, od sada pa do kraja kursa, uvedemo bilo koji novi pojam.
- Trend u programiranju je da se pokazivači izbjegnu kad god je to moguće, ali to nije moguće u programskom jeziku C, niti u aplikacijama koje rade sa hardverskim uređajima.

Pokazivači

- Pokazivač se deklariše kao:
  int *y;
- Zvjezdica ispred imena promjenljive **y** je oznaka da je u pitanju pokazivač, u ovom slučaju **pokazivač na cijeli broj**.
- Pokazivač je zapravo **adresa** **(prvog bajta)** neke (u ovom slučaju int) promjenljive u memoriji računara. Na primjer:
  int x, *y;
- Sada možemo postaviti **y=&x**, gdje je **&** već uvedeni operator **adresa od**:
- Znači, u promjenljivoj **y** se sada čuva adresa promjenljive **x**.

Pokazivači

- Preko pokazivača **y** se sada može pristupiti podatku koji se nalazi na adresi **y**:
  ***y=20;**
  uočite da je sada u pitanju unarni operator
  Upiši broj **20** u podatak koji se nalazi na memorijskoj adresi **y**, odnosno u konkretnom slučaju u promjenljivu **x**.
- Koncept vam vjerovatno još uvijek djeluje veoma apstraktno, ali sa njegovom neophodnošću ćemo se upoznati kasnije, posebno kad budemo radili funkcije i nizove.
- Tokom rada pokazivač može da pokaže na drugu, proizvoljnu, promjenljivu.
- "Dozvoljene" operacije sa pokazivačima su: poređenje, sabiranje sa konstantom, inkrementiranje i dekrementiranje. Značaj ovih operacija ćemo vidjeti kada budemo učili nizove.

Inicijalizacija pokazivača

- Pokazivačke promjenljive se mogu inicijalizovati i deklarisati baš kao i ostale promjenljive. Primjer:

```
int x;
int *y=&x;          // moglo je i int x,*y=&x;
```

- Drugi primjer je:

```
int *y=NULL;
```

gdje je **NULL** simbolička konstanta definisana u zaglavlju (biblioteci) stdio.h, koja kaže da pokazivačka promjenljiva u datom trenutku ne pokazuje ni na jednu promjenljivu (alternativno se može pisati int *y=0;).

Inicijalizacija pokazivača - Značaj

- Ako se pokazivačka promjenljiva ne inicijalizuje, ona pokazuje na bilo koju adresu u memoriji, a to može biti i adresa podataka potrebnih za rad operativnog sistema i sistemskih programa, proizvoljnih promjenljivih našeg programa, pa, recimo, čak i adresa preko kojih se komunicira sa periferijama.
- U velikim programima možete incidentno koristiti ovu promjenljivu što može dovesti do "pucanja" programa.
- Stoga je značaj inicijalizacije pokazivača veći nego inicijalizacije ostalih promjenljivih.
- **Grubo pravilo**: ako pokazivač ne pokazuje na željenu promjenljivu u bilo kom trenutku programa treba ga vratiti na NULL.