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 su:
  - scanf (za unos podataka sa tastature) i
  - printf (za prikaz rezultata na monitoru)

  Da bi se koristile u programu, potrebno je uključiti programsku biblioteku stdio.h.

- 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.

Funkcija printf

- O pretprocesoru će biti riječi u nastavku kursa.
- Funkcija printf se može javiti u dva oblika. Prvi je jednostavniji:

```
printf("Tekst koji ce biti odstampan \n");
```

Znak za novi red se često koristi u printf.

- Drugi oblik je znatno značajniji:

```
printf("%d %d %e tekst %c\n", x, 60, y, z);
```

%slovo se naziva **specifikator formata**.

Funkcija printf u ovom obliku štampa string koji joj je prvi argument, dok se umjesto parova %slovo štampa vrijednost argumenata nakon stringa (ide se redom). 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 specifikatora formata.
- Argumenti printf-a: promjenljive, izrazi, konstante i pozivi funkcija.
- Specifikatori formata imaju sljedeće značenje:
  - %d i %i cijeli brojevi,
  - %bd cijeli broj sa b cifara,
  - %o oktalni zapis cijelog broja,
  - %x, %X, %#X heks. zapis cijelog broja (mala slova; velika slova; sa oznakom 0x ispred broja),
  - %f realni brojevi,
  - %.bf realni broj sa b decimalnih mjesta,
  - %a.bf realni broj sa ukupno a pozicija i b decimalnih mjesta, podrazumjeva se a>b (pozicije uključuju predznak, tačku i sve cifre).

Funkcija printf

- Nastavljamo sa specifikatorima formata:
  - %e eksponencijalni zapis realnih brojeva,
  - %g realni brojevi u kompaktnom zapisu (završne nule nisu uključene i ne prikazuje se decimalna tačka kod cijelih brojeva),
  - %s stringovi,
  - %p pokazivač.
- Rekli smo da argument funkcije može biti izraz. Na primjer:

```
printf("%d\n", j++); // štampa j, a zatim uvećava j za 1
```

- Argument printf-a može biti i funkcija. Na primjer:

```
printf("%d\n", zbir_cifara(N)); // funkcija zbir_cifara(N) vraća
                                   zbir cifara broja N
```

Funkcija scanf

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

```
scanf("%c%d", &kar, &cb);
```

Specifikatori formata za tipove promjenljivih koje se učitavaju, ovdje karakter i cijeli broj.

&kar i &cb predstavljaju adrese promjenljivih kar i cb, a ono što se unese sa tastature smješta se na te adrese.

Podrazumjeva se da je broj procenata jednak broju adresa i da su procenti na pravilan način upareni sa tipovima promjenljivih koji se učitavaju. Ako ova dva "podrazumjevanja" nijesu ispunjena, program i dalje može raditi, ali šta će tačno raditi i kakve će promjenljive učitati to nije sigurno. U svakom slučaju, bolje je 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 taster Enter. 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.
- Što se dešava ako unesete tri podatka, a traže se dva? Prva dva podatka se iskoriste, dok se treći čuva u memoriji koja se naziva **bafer** i taj podatak je spreman da bude preuzet narednom funkcijom za unos.

scanf – primjer

- scanf("%d%d",&a,&b); // ako unesemo 2 3 4 i pritisnemo Enter,
  // neke naredbe // 2 i 3 se dodjeljuju promjenljivim a i b

  scanf("%d",&c); // učiniće nam se da je ova naredba preskočena,
  // a zapravo je broj 4, preostao iz prethodnog unosa
  // i koji se nalazi u baferu, smješten u promjenljivu c

- Postoji način da se ovakav rad izbjegne, ali o tom potom.
- Prije funkcije scanf se obično postavlja printf koja objašnjava korisniku šta treba da unese:

```
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
  - definicija 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 obuhvata velikim zagradama **{}** i sastoji se od deklaracija promjenljivih koje se obično postavljaju na početku bloka i naredbi koje slijede. Blok naredbi u C-u je sekvenca naredbi ubuhvaćena sa **{}**.
- Prije tijela funkcije se postavlja zaglavlje koje sadrži tip rezultata, ime funkcije, 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 unutar velikih zagrada (tretira se kao jedna)
  {nar1; nar2; nar3; ... narN;}
- Prazna naredba ;
- Osam naredbi za kontrolu toka:
  - if
  - switch
  - while
  - do
  - for
  - break
  - continue
  - goto
- Naredba return za vraćanje rezultata funkcije.

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 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: Naredba 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 kod korišćenja tačke-zarez!

```
if(uslov) ;  // ispravna naredba, ali vjerovatno ne radi ono što ste zamislili
```

evo i razloga!

Varijante naredbe if

- 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 se N1 ako je ispunjen uslov1, a ako nije ispunjen uslov1, a jeste uslov2, izvršava se N2...
Ako nijedan od uslova nije ispunjen, izvršava se Nn.

Postoji bjelina

U C-u ne postoji elseif, već je if ugnježđeno u dijelu else.

Naredba switch-case

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

```
switch(cjel_izr)
{
      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 cjel_izr 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, cjel_izr mogu biti i karakteri!

Naredba switch-case

- 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 ponekad može biti korisno, ali obično nije ono što se traži i da bi se nakon izvršenja traženog case-a 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 (petlje) je sličan while ciklusu 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 ciklusu, izraz3 opisuje što se sa brojačem dešava u okviru ciklusa, tj. način na koji se on mijenja. naredba ili blok naredbi se izvršava sve dok je izraz2 ispunjen. izraz3 se izvršava nakon naredbe ili bloka naredbi.

for ciklus

- Primjer:

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

Prvi put se ciklus izvršava za vrijednost brojača i=0, nakon čega se brojač 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 obrnuto. Pravilo je: prvo se zatvara unutrašnja petlja, zatim spoljašnja.

for ciklus

- Primjer ugnježđenog ciklusa:

```
S=0;
for(i=0; i<6; i++)
        for(j=1; j<5; 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 ugnježđena, već jednostruka petlja.

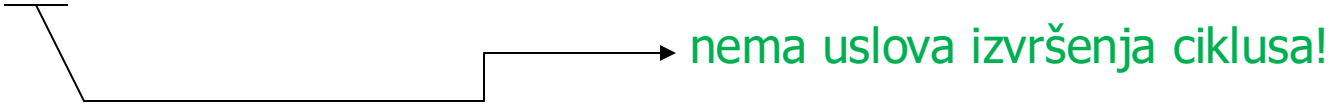
Beskonačni ciklusi

- Ciklusi tipa:

```
while(1)  {/*neke naredbe*/}
```

ili

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



nazivaju se **beskonačnim**.

- Sintaksno su ispravni (kompajler će ih prihvatiti), ali su često 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 continue, break i goto

- Naredba continue prekida tekuće izvršavanje ciklusa (tekuću iteraciju) i započinje novo izvršavanje. Naredna naredba nakon continue je provjera da li je ispunjen uslov while ili for ciklusa.
- Naredba break prekida izvršenje ciklusa u kom se nalazi i "skače" na prvu naredbu nakon ciklusa. Ako imamo slučaj ugnježdenih petlji i break se nalazi u unutrašnjoj, sa break se izlazi samo iz unutrašnje petlje, a ne i iz spoljašnje.
- Naredba break se smije naći samo unutar ciklusa i switch-case!
- Naredba goto ima sintaksu:
  goto labela; //labela je ispravno deklarisano ime u programu
- Naredba na koju se "skače" sa goto je označena sa
  labela: naredba;

Upotreba continue i goto

- Još 1966. Bohm i Jacopini su dokazali da se svi algoritamski rješivi problemi mogu riješiti pomoću sekvenci, ciklusa i selekcija.
- 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!

Bohm, C., and G. Jacopini, "Flow diagrams, Turing machines, and languages with only two formation rules," *Communications of the ACM*, Vol. 9, No. 5, May 1966, pp. 336–371.

Pokazivači

- Prvi složeni tip podataka kojeg uvodimo je pokazivač (eng. pointer). U pitanju je moćan, ali često i nerazumljiv koncept.
- Nemoguće je izbjeći rad pokazivačima u programskom jeziku C, kao i u aplikacijama koje rade sa hardverskim uređajima.
- Pokazivač se deklariše sa int *p;
- Zvjezdica ispred imena promjenljive p označava da je u pitanju pokazivač, u ovom slučaju pokazivač na cijeli broj.
- Pokazivač predstavlja **adresu** (prvog bajta) promjenljive u memoriji računara.
- Na primjer, nakon deklaracije
  int x, *p;
  naredbom p=&x; u p upisujemo adresu promjenljive x.
  Operacija uzimanja adrese se naziva referenciranje.

Pokazivači

- Preko pokazivača **p** se sada može pristupiti podatku koji se nalazi na adresi **p** (ova operacija se zove dereferenciranje):
  ***p = 20;** ⟶ uočite da je * sada unarni operator
  Upiši broj **20** na adresu **p**, u ovom slučaju u promjenljivu **x**.
- Koncept vam vjerovatno još uvijek djeluje apstraktno, ali sa njegovom neophodnošću ćemo se upoznati kasnije, posebno kad budemo radili nizove i funkcije.
- Tokom rada pokazivač može da pokaže na drugu, proizvoljnu, promjenljivu.
- Dozvoljene operacije sa pokazivačima su: poređenje pokazivača, sabiranje sa konstantom, inkrementiranje i dekrementiranje pokazivača.

Inicijalizacija pokazivača

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

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

- Drugi primjer je:

```
int *p = NULL;     // ili int *p=0
```

gdje je NULL simbolička konstanta definisana u biblioteci stdio.h, koja kaže da pokazivačka promjenljiva u datom trenutku ne pokazuje ni na jednu promjenljivu.

- Takođe, inicijalizacija se može vršiti i konkretnom adresom:

```
int *p = (int *) 0x000000006FA126BD;    // 64-bitna arhitek.
```

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 tokom izvršenja programa, vratiti ga na NULL.