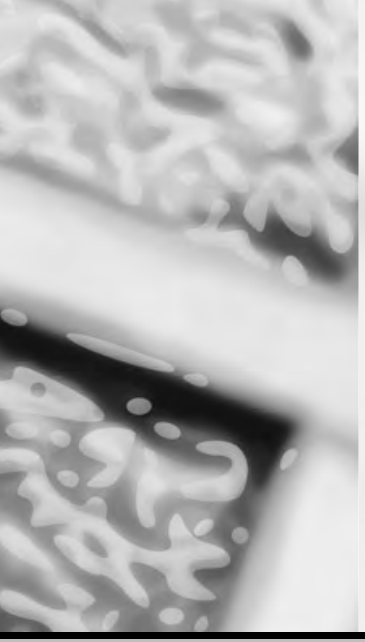

C i algoritmi

"Genius is one percent inspiration, ninety-nine percent perspiration."
Thomas Alva Edison

Prvi dio knjige bavi se programskim jezikom C. U ovom dijelu neće biti objašnjavani složeniji algoritmi. Izloženo gradivo predstavlja osnovu za daljnje izučavanje. Donekle jednostavno gradivo bit će popraćeno primjerima koji neće uvijek biti jednostavni.

1. Osnove C-a

1.1. Prvi program u C-u

Prije nego počnemo s programiranjem, trebamo znati kako unutar koda pisati tekst koji se ne izvršava, tj. zanemaruje se prilikom pokretanja. Takav tekst se zove **komentar**.

Postoje dvije vrste komentara:

```
// ovo je komentar koji se nalazi samo u jednoj liniji
```

ili

```
/* ovo je komentar koji se moze nalaziti u vise linija */
```

Treba primijetiti da komentar koji počinje s `//` mora završiti u istoj liniji, dok komentar koji počinje s `/*` može biti u više linija jer ima znak kraja komentara `*/`. Komentar koji počinje s `//` završava sa znakom za kraj reda (u C-u se taj znak označava s `\n`), dok komentar koji počinje s `/*` završava s `*/`, te se može nalaziti i usred linije koda. (Komentar koji počinje s `//` specifičan je za jezik C++, ali ga podržava većina C prevodioca.)

```
// ovo je komentar napisan
// u vise linija

/* ovo je takodjer komentar
napisan u vise linija
*/
```

Prvo smo naučili kako pisati komentare kako bismo mogli objašnjavati što koja linija u kodu radi.

Kod učenja novog programskog jezika postoji tradicija da se prvo nauči napraviti program koji ništa ne unosi, ništa ne računa i uvijek radi isto – ispisuje sljedeće:

```
Hello world!
```

Držat ćemo se te tradicije. Slijedi prvi kôd:

```
01. #include <stdio.h>
02.
03. int main() {
```

```
04.   printf("Hello world!\n");
05.   return 0;
06. }
07.
```

Ispred svake linije koda dodana je brojka kako bi se lakše opisao kôd (te brojke nisu sastavni dio koda, nego su u knjizi nadodane). Ukoliko imamo brojke ispred svake linije kôda, možemo reći: „u 4. liniji koda je funkcija za ispis“ i odmah je vidljivo koja je linija četvrta, bez brojenja linija. (Zamislite da u kodu od 200 linija trebate pronaći onu 183.) Kad pišemo kôd u tekstualnom editoru, naravno da ne pišemo brojke ispred linija koda. Stvarni kôd izgleda ovako:

```
#include <stdio.h>

int main() {
  printf("Hello world!\n");
  return 0;
}
```

Analizirajmo što taj kôd radi.

```
01. #include <stdio.h> /* ovo ćemo kasnije objasniti, za sada
    samo prepisujemo */
02.
03. int main() { /* ovdje počinje program, zasada prepisujemo */
04.   printf("Hello world!\n"); /* ovo je funkcija za ispis */
05.   return 0; /* ovo je također uvijek tu, zasada prepisujemo */
06. } /* ovdje završava program, isto prepisujemo */
07.
```

U analizi je rečeno da se zasad prepisuju sve linije osim četvrte. Ostalo će biti objašnjeno kasnije. Četvrta linija je:

```
printf("Hello world!\n");
```

Zasad se može reći da je `printf` funkcija za ispis. Svaka funkcija ima argumente koji se nalaze unutar zagrada, te ima oblik:

```
nazivFunkcije( nekiArgumenti );
```

> Treba primijetiti da svaka naredba na kraju ima znak `;` (točka-zarez). Taj znak označava kraj pojedine naredbe i ne smije se ispustiti.

Promotrimo sada argument funkcije `printf`. U kodu smo toj funkciji proslijedili:

```
"Hello world!\n"
```

To je niz znakova koji naredba ispisuje. Niz znakova se u C-u ograničava dvostrukim navodnicima, npr.: `"niz znakova"`, dok se jedan znak označava jednostrukim navodnicima, npr. `'a'`.

Prethodno se navedeni niz sastoji od 13 znakova. To su:

```
'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '!' i
'\n'
```

`'\n'` je posebna oznaka za znak koji dobijemo pritiskom tipke **enter**. `'\n'` predstavlja prelazak u novi red.

> Preporučam da prilikom pisanja kodova u C-u ne koristite hrvatska latinična slova s dijakritičkim znakovima č, ć, ž, š i đ. Program može raditi i s njima, ali ih radije izbjegavajte. Steći ćete lošu naviku koja kasnije može stvarati probleme.

Napravite za vježbu program koji na ekran ispisuje sljedeće:

```
Ovo je moj
prvi
program! :)
```

Ukoliko ne uspijete odmah, naučit ćete kasnije.

1.1.1. Oblikovanje izgleda kôda

Napisani kôd ne mora biti lijep kao onaj prethodno naveden. Pokušajte pokrenuti sljedeći kôd i vidjet ćete da je sve u redu.

```
#include <stdio.h>

int
main() { printf(
"Hello world!\n"
);return
0;}
```

Kao što možete primijetiti, radi se o istom prije navedenom kodu, ali neuredno napisanom. Kad bismo probali prekinuti kôd usred naziva funkcije ili niza znakova koje prosljeđujemo funkciji `printf`, došlo bi do pogreške prilikom prevođenja. Sljedeća dva koda javljaju pogrešku:

```
#include <stdio.h>
int
main() { pri
ntf(
"Hello world!\n"
);return
0;}
```

```
#include <stdio.h>
int
main() { printf(
"He
llo world!\n"
);return
0;}
```

Pogreška nastala zbog prekida niza znakova može se popraviti pišući znak `'\'` neposredno prije prelaska u novi red (znak `'\'` mora biti zadnji u redu, nakon njega ne smije biti razmak). Sljedeći kôd je ispravno napisan:

```
#include <stdio.h>

int main() {
  printf("He\
llo world!\n");
  return 0;
}
```

U C-u se velika i mala slova **razlikuju** i zato sljedeći kôd nije valjan:

```
01. #include <stdio.h>
02.
03. int main() {
04.   Printf("Hello world!\n");
05.   return 0;
06. }
07.
```

Prilikom prevođenja programa, prevodilac će (npr. gcc) javiti pogrešku u 4. liniji, tvrdeći da funkcija `Printf` ne postoji, što je i istina, jer postoji `printf`, a ne `Printf`. Pripazite na takve česte pogreške.

1.2. Osnovni tipovi podataka

Naučimo neke osnove o tipovima podataka. Kasnije će se detaljnije proučavati ova tema.

Postoje tri bitna tipa podataka koji se koriste u C-u za spremanje vrijednosti. To su `int`, `char` i `double`. Pomoću njih se stvaraju varijable u koje se pohranjuju vrijednosti. Promotrimo prvo `int`. U `int` se spremaju cjelobrojne vrijednosti, tj. pozitivni i negativni cijeli brojevi.

```
01. #include <stdio.h>
02.
03. int main() {
04.   int a;
05.   int b=6;
06.   int c=3,d;
07.   a=-b;
08.   d=a*b+c;
09.   printf("%d %d\n%d %d\n",a,b,c,d);
10.     // %d oznacava mjesto gdje se stavlja int
11.   return 0;
12. }
13.
```

Navedeni program stvara četiri varijable tipa `int`, namijenjene za spremanje cjelobrojnih vrijednosti. Varijable se stvaraju u 4., 5. i 6. liniji. U 4. liniji je stvorena varijabla tipa `int` naziva a. Primijetimo da na kraju svake naredbe, koja stvara varijable, imamo napisano `;`

jer time označavamo kraj naredbe. Ukoliko ispustimo `;` doći će do pogreške prilikom prevođenja programa. Varijabli `a` na početku nije pridružena vrijednost, te zbog toga ona na početku može sadržavati bilo koju vrijednost. Kada bismo probali računati nešto s varijablom kojoj nije pridružena vrijednost, rezultat bi bio slučajna vrijednost. U 5. liniji je stvorena varijabla s nazivom `b` i odmah pri stvaranju joj je pridružena vrijednost, broj 6. U 6. liniji su stvorene varijable `c` i `d`. Varijabli `c` je pridružena početna vrijednost 3, dok varijabla `d` sadrži neodređenu vrijednost.

Korištena je funkcija `printf` za ispis podataka. Funkcija `printf` bit će kasnije detaljno obrađena. Program izvršavanjem ispisuje sljedeće:

```
        -6 6
        3 -33
```

Program se može skratiti na sljedeći način:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int a,b=6,c=3,d;
05.   a=-b;
06.   d=a*b+c;
07.   printf("%d %d\n%d %d\n",a,b,c,d);
08.   return 0;
09. }
```

Sve su varijable stvorene u jednoj naredbi (naredba završava s `;`). Vjerojatno ste već primijetili da operator = upisuje vrijednost s desne strane tog znaka u varijablu s lijeve strane. Varijabla s lijeve strane mora biti promjenjiva (kasnije ćemo naučiti da postoje i „nepromjenjive varijable“ koje uvijek imaju istu vrijednost). Također, s lijeve strane ne smije biti izraz. `a+b=c+d;` je besmislena naredba.

Budući da se uvijek računaju iste vrijednosti (jer nema unosa podataka), program se može skratiti na sljedeći način:

```
01. #include <stdio.h>
02.
03. int main() {
04.   printf("-6 6\n3 -33\n");
05.   return 0;
06. }
```

U predzadnjem kodu su varijable tipa `int` nazvane `a`, `b`, `c` i `d`. Postoje pravila u vezi nazivanja varijabli:

- Naziv varijable smije početi slovom (velika i mala slova se razlikuju) ili znakom `'_'`.
- Naziv smije sadržavati slova, brojke i znak `'_'` (nema razmaka).
- Naziv ne smije biti ključna riječ programskog jezika.
- Naziv ne smije biti previše dugačak (varijable se trebaju razlikovati u prvih 31 znakova u nazivu, iako neki prevodioci nemaju ograničenje).

Znači, varijable nazvane `Marko`, `mojNazivVarijableZaBroj`, `moj_naziv`, `_broj`, `B_32d` i `Int` su ispravno nazvane, dok varijable nazvane `1nesto`, `varijabla od vise rijeci`, `int` ili `vise-manje` nisu ispravno nazvane.

Proučimo tip podataka `double`.
Tip podataka `double` služi za spremanje brojeva koji imaju u sebi decimalnu točku i konačno mnogo decimala. Možemo općenito reći da su to decimalni brojevi. To su npr. `1.25`, `5.5555555`, `-0.01`, ali također i brojevi `2.0` i `1.5e-6`.

Primjer:

```
01. #include <stdio.h>
02.
03. int main() {
04.   double a=0.02, b=2;
05.   double razlika;
06.   razlika = a / b + 100.5;
07.   printf("%lf\n",razlika); // %lf predstavlja double
08.   razlika = razlika - 2;
09.   printf("%lf\n",razlika);
10.   return 0;
11. }
12.
```

Rezultat izvršavanja:

```
        100.510000
        98.510000
```

Varijabla je nazvana `razlika` i sadržava razliku brojeva `a` i `b` uvećanu za `100.5`.

Tipovi podataka se mogu međusobno miješati pri računanju, ali pri dijeljenju nećete uvijek dobiti željeni rezultat ukoliko ni djeljenik ni djelitelj nisu tipa `double`; pojašnjeno je kasnije u knjizi.

> Nemojte koristiti `double` za varijable koje nikad ne trebaju sadržavati decimalni dio, radije koristite `int`.

Od tri bitnija tipa podataka, preostao je još tip podataka `char`.
Tip podataka `char` služi za spremanje znakova kao što su `'a'`, `'L'`, `'w'`, `'='`, `'.'`, `'+'`, `'0'`, `'_'`, `'4'`, `'?'`, `'\n'` i slično.

```
01. #include <stdio.h>
02.
03. int main() {
04.   char a='\n',b='A';
05.   printf("%c%c%c%c",b,a,b,a); // %c za char
06.   return 0;
07. }
08.
```

Rezultat izvršavanja navedenog koda:

```
        A
        A
```

1.3. Funkcije printf i scanf

Upoznajmo se ukratko s funkcijama `printf` i `scanf`. Kasnije ćemo ih detaljnije proučiti.

Funkcija `printf` služi za ispis podataka, pogledajmo primjer.
Dio koda:

```
    int x=2;
    printf("Ispisat cu broj dva : %d\n",x);
```

ispisuje na zaslon:

```
        Ispisat cu broj dva : 2
```

Funkcija `printf` prima različit broj argumenata. Prvi argument je niz znakova ograničen znakovima ", koji definira što će se ispisati, tj. oblik ispisa. Slijedi popis varijabli koje se koriste pri ispisu, onim redom kojim su najavljene u nizu znakova, odvojene zarezima. U navedenom primjeru u nizu znakova imamo napisanu oznaku (specifikator) %d. To znači da će na mjestu oznake %d biti ispisan broj tipa `int`. Taj broj je kasnije naveden kao dodatni argument (odvojen zarezom). Radi se o varijabli s nazivom `x`. Dakle, tip podataka `int` najavljujemo tako da napišemo %d. Slično najavljujemo i `double` pomoću `%lf`, te `char` pomoću %c.

Slijedi još nekoliko primjera.

Kôd1:

```
01.    int dva=2,tri=3;
02.    printf("Ispisa%dt cu b%d%droj%d dva :
    %d\n",dva,dva,dva,dva,tri);
03.
```

ispis1:

```
        Ispisa2t cu b22roj2 dva : 3
```

Kôd2:

```
    int a=1,b=3;
    printf("nekiTekst%d, %d,te%dkst\n",a,a+b,b*3-a);
```

ispis2:

```
        nekiTekst1, 4,te8kst
```

Kôd3:

```
01. #include <stdio.h>
02.
03. int main() {
04.
05.   double d=4;
06.   int i;
07.   char c='X';
08.   double d2=5.5;
09.   i=7;
10.   d=d/i;
11.   printf("char c je %c, double: %lf i \
```

```
12. %lf, int i = %d\n",c,d,d2,i);
13.
14.   return 0;
15. }
16.
```

ispis3:

```
        char c je X, double: 0.571429 i 5.500000, int i = 7
```

U zadnjem je kodu niz znakova prekinut znakom \ i nastavljen u 12. liniji.

Funkcija za unos podataka `scanf` je po građi vrlo slična funkciji `printf`. Proučimo funkciju `scanf` kroz nekoliko primjera.

Kôd:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int a;
05.   double b;
06.   char c;
07.   scanf("%d %lf %c",&a,&b,&c);
08.   printf("%d %lf %c",a,b,c);
09.   return 0;
10. }
11.
```

Unos:

```
        3 3.4 d
```

ispis:

```
        3 3.400000 d
```

Primijetimo da u 7. liniji ispred naziva varijabli postoji znak `'&'`. Znak `'&'` u funkciji `scanf` ne smijemo ispustiti; kasnije će biti objašnjeno zašto. Ispuštanje znaka & jedna je od čestih pogrešaka prilikom programiranja.

Kad tipkovnicom unosimo nešto pomoću `scanf`, odvajamo unesene vrijednosti razmakom ili **enter**-om (prelaskom u novi red), a nikako zarezom!

Funkcija `scanf` može imati i nekakav tekst unutar niza znakova, no uglavnom se tekst ne navodi jer time dobivamo ponašanje koje često nismo željeli:

```
#include <stdio.h>

 int main() {
   int a=-2;
   scanf("nesto%d",&a);
   printf("%d\n",a);
   return 0;
 }
```

Ako pokrenemo ovaj kôd (program) i unesemo:

```
43
```

ispisat će se:

```
-2
```

Ako unesemo:

```
nesto43
```

ispisat će se:

```
43
```

Ako unesemo:

```
neeeo43
```

ispisat će se:

```
-2
```

Razlog je u tome što `scanf` očekuje da upišemo doslovno ono što smo naveli u nizu znakova. `scanf` ne služi za ispis nego samo za unos, tako da linija:

```
scanf("Unesi broj: %d",&x);
```

vjerojatno ne radi ono što ste htjeli. Trebali ste napisati sljedeće:

```
printf("Unesi broj: ");
scanf("%d",&x);
```

To je jedna od čestih pogrešaka pri programiranju!

Ponekad je korisno navesti dodatne znakove u `scanf`-u, npr:

Zadatak:

Napravi program koji unosi trenutno vrijeme u obliku hh:mm:ss i ispisuje koliko je sekunda prošlo od ponoći.

Primjer unosa:

```
21:09:11
```

odgovarajući ispis:

```
proslo je 76151 sekunda od ponoci
```

Rješenje:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int sati, minute, sekunde;
05.   scanf("%d:%d:%d",&sati,&minute,&sekunde);
06.   sekunde = sekunde+minute*60+sati*60*60;
07.   printf("proslo je %d sekunda od ponoci\n",sekunde);
08.   return 0;
09. }
10.
```

Zadatak:

Napravi program koji unosi troznamenkasti broj i ispisuje ga obrnutim redoslijedom.

Primjer unosa:

```
        123
```

odgovarajući ispis:

```
        321
```

Rješenje:

```
01. #include <stdio.h>
02.
03. int main() {
04.   char a,b,c;
05.   scanf("%c%c%c",&a,&b,&c);
06.   printf("%c%c%c",c,b,a);
07.   return 0;
08. }
09.
```

Ne unosimo broj kao `int` nego kao tri znaka (budući da tekst zadatka tvrdi da se radi o troznamenkastom broju), a unesene znakove ispisujemo obrnutim redoslijedom.

Unos znakova (`char`) s `%c` često može stvoriti probleme jer `scanf` može učitati razmak koji ste slučajno unijeli. Za sada izbjegavajte koristiti `%c` u `scanf`-u; kasnije ćete naučiti više.

1.4. Osnovne naredbe

1.4.1. Blokovi

Ponekad je potrebno grupirati naredbe u jedan blok naredbi kako bi se znalo da određeni skup naredaba čini jednu cjelinu. To se čini pomoću vitičastih zagrada `{` i `}`. Blok naredbi se tretira kao jedna naredba. (Znamo od prije da pojedina naredba završava znakom `;`.) Prisjetimo se prvog napisanog koda:

```
01. #include <stdio.h>
02.
03. int main() {
04.   printf("Hello world!\n");
05.   return 0;
06. }
07.
```

Na kraju treće linije počinje blok naredaba glavne funkcije zvane `main`, dok se u šestoj liniji zatvara. Time naredbe u četvrtoj i petoj liniji čine cjelinu koja spada u glavnu funkciju `main`.

Na **početku** bloka se navode varijable koje postoje u tom bloku:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int a=1;
05.   printf("%d\n",a);
06.   {
07.     int a=3;
08.     printf("%d\n",a);
09.   }
10.   printf("%d\n",a);
11.   return 0;
12. }
13.
```

ispis:

```
        1
        3
        1
```

Varijabla a, koja je najavljena u 7. liniji, postoji samo od 6. do 9. linije i u tom dijelu koda se „vanjska" varijabla a s vrijednošću 1 zanemaruje, što vidimo iz ispisa u 8. liniji. Varijabla a s vrijednošću 1 postoji od 3. do 12. linije. Budući da u 8. liniji pri ispisu ne smije postojati sumnja u to koju varijablu treba ispisati, uvijek se misli na varijablu koja je definirana u kraćem (unutarnjem) dijelu koda. U 10. liniji varijabla a s vrijednošću 3 više ne postoji, pa se opet misli na varijablu a s vrijednošću 1 definiranu u 4. liniji.

Na početku bloka se najavljuju varijable koje postoje samo unutar tog bloka. Najšire područje gdje se varijabla može naći je **globalno područje** (izvan funkcije). Varijabla definirana u globalnom području postoji svugdje u kodu. Primjer:

```
01. #include <stdio.h>
02.
03. int x,y=1;
04.
05. int main() {
06.   printf("%d %d\n",x,y);
07.   return 0;
08. }
09.
```

Pokretanjem se ispisuje:

```
        0 1
```

> Za razliku od varijabli definiranih unutar nekog bloka, varijable definirane u globalnom području se uvijek postavljaju na 0 (ukoliko im se pri stvaranju ne pridruži neka druga vrijednost).

x će u gornjem primjeru imati vrijednost 0, a ne neodređenu vrijednost.

1.4.2. Naredbe grananja

Naredba `if` je naredba grananja.
Minimalni oblik naredbe je:

```
    if (nekiUvjet)
      naredbaIliBlok;
```

Ukoliko je `nekiUvjet` istinit, onda se izvršava `naredbaIliBlok`.
Krenimo s primjerom:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int x;
05.   scanf("%d",&x);
06.   if (x>99)
07.     printf("Uneseni broj ima barem tri znamenke\n");
08.   return 0;
09. }
10.
```

U 6. liniji se nalazi uvjet za grananje. Ukoliko je uvjet istinit, tj. ukoliko je x>99, izvršit će se sljedeća naredba (iza `if` naredbe), a to je ispis poruke. Zapamtimo da iza `if` naredbe ne ide `;` jer ukoliko ga stavimo, stvorili smo novu praznu naredbu koja će se „izvršiti". Dakle:

```
    if (nekiUvjet);
      nekaNaredba;
```

Ovo je besmislena sintaksa jer će se `nekaNaredba` uvijek izvršiti. `if` se odnosi na prvu sljedeću naredbu, tj. do znaka `;`, a budući da smo znak `;` stavili odmah iza `if` naredbe, stvorili smo praznu naredbu koja će se izvršiti ako je `if` istinit.

Želimo li baš to napraviti, bilo bi bolje napisati kôd na sljedeći način (kako bismo naglasili što želimo postići):

```
    if (nekiUvjet)
      ;
    nekaNaredba;
```

Ukoliko želimo da se `if` odnosi na više naredaba, tada je potrebno naredbe grupirati u blok. Slijedi primjer:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int x;
05.   scanf("%d",&x);
06.   if (x>99) {
07.     printf("Uneseni broj ima ");
08.     printf("barem tri znamenke\n");
09.   }
```

```
10.   return 0;
11. }
12.
```

U primjeru su 7. i 8. linija grupirane u blok i obje će se izvršiti ako je x>99. Vitičastim zagradama određujemo početak i kraj grupiranog sadržaja.

Primjer:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int x;
05.   scanf("%d",&x);
06.   if (x>99) {
07.     if (x<1000)
08.       printf("x je troznamenkast");
09.     if (x>999) {
10.       if (x<10000)
11.         printf("x je cetveroznamenkast");
12.       if (x>9999)
13.         printf("x je peteroznamenkast ili veci");
14.     }
15.   }
16.   return 0;
17. }
18.
```

Naredba `if` iz 6. linije ima blok sve do 15. linije, naredba `if` iz 7. linije se odnosi samo na naredbu u 8. liniji, dok se naredba `if` u 9. liniji odnosi na sve do 14. linije. Program ispisuje da je broj troznamenkast ako ima točno tri znamenke, da je četveroznamenkast ako ima točno četiri znamenke, da je peteroznamenkast ili veći ako ima više od 4 znamenke. Zbog jednostavnosti prikaza u programu zanemareni su negativni brojevi koji isto mogu biti troznamenkasti i sl.

Naredba `if` može imati „dodatak" koji se izvršava ukoliko uvjet u `if` naredbi nije istinit. To činimo pomoću `else` naredbe. Pogledajmo primjer:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int x;
05.   scanf("%d",&x);
06.   if (x>99)
07.     printf("x je troznamenkast ili veci");
08.   else
09.     printf("x je jednoznamenkast ili dvoznamenkast");
10.   return 0;
11. }
12.
```

Ako je uvjet x>99 lažan, onda će se izvršiti naredba u 9. liniji. `else` naredba se odnosi na `if` napisan prije nje. Ipak, to ponekad može biti zbunjujuće. Slijedi primjer:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int x;
05.   scanf("%d",&x);
06.   if (x>99)
07.   if (x>999)
08.   printf("x je veci od 999");
09.   else
10.   printf("ovo je else naredba");
11.   return 0;
12. }
13.
```

Linije namjerno nisu uvučene. Postavlja se pitanje odnosi li se `else` na `if` naredbu iz 6. ili iz 7. linije. Na takva pitanja ne treba odgovarati i ne treba se dovoditi u ovakvu nedoumicu, nego kôd treba napisati na jedan od sljedeća dva načina:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int x;
05.   scanf("%d",&x);
06.   if (x>99) {
07.     if (x>999)
08.       printf("x je veci od 999");
09.     else
10.       printf("ovo je else naredba");
11.   }
12.   return 0;
13. }
14.
```

```
01. #include <stdio.h>
02.
03. int main() {
04.   int x;
05.   scanf("%d",&x);
06.   if (x>99) {
07.     if (x>999)
08.       printf("x je veci od 999");
09.   }
10.   else
11.       printf("ovo je else naredba");
12.   return 0;
13. }
14.
```

Za rješavanje problema u kojima ne znate na što se `else` odnosi, kôd treba pomoću vitičastih zagrada rasporediti u blokove.

```
#include <stdio.h>

int main() {
  int x;
  scanf("%d",&x);
  if (x>5) { printf("x je veci od 5\n"); }
  if (x>5) printf("x je veci od 5\n");
  return 0;
}
```

Program dvaput ispisuje da je x veći od 5, ako on to zbilja jest. Obje `if` naredbe su jednako dobro napisane, s time da prvi `if` izvršava blok koji sadrži jedne naredbe, dok drugi `if` izvršava jednu naredbu.

`else` se često može izbjeći pisanjem nove `if` naredbe:

```
if (x>5) printf("x>5\n");
if (x<=5) printf("x<=5\n");
```

Umjesto `if (x<=5)` mogli smo napisati `else`. Međutim, nema potrebe izbjegavati `else`.

Kao kombinacija `if` i `else` naredaba može se pisati `else if` naredba. Proučimo primjer:

```
01. #include <stdio.h>
02.
03. int main() {
04.   int x;
05.   scanf("%d",&x);
06.
07.   if (x<=9)
08.     printf("x je 1-znamenkast\n");
09.   else if (x<=99)
10.     printf("x je 2-znamenkast\n");
11.   else if (x<=999)
12.     printf("x je 3-znamenkast\n");
13.   else if (x<=9999)
14.     printf("x je 4-znamenkast\n");
15.   else if (x<=99999)
16.     printf("x je 5-znamenkast\n");
17.   else
18.     printf("x je 6-znamenkast ili veci\n");
19.
20.   return 0;
21. }
22.
```

U navedenom nizu uvjeta izvršit će se prvi koji je istinit, dok se ostali uvjeti neće. Ako niti jedan uvjet nije istinit, izvršit će se `else`.

Unutar `if` naredbe koriste se različiti operatori. Slijede neki od njih:

Operator	Način uporabe	Opis
>	a > b	Vraća istinu ukoliko je a veći od b.
<	a < b	Vraća istinu ukoliko je a manji od b.
==	a == b	Vraća istinu ukoliko su a i b jednaki.
>=	a >= b	Vraća istinu ukoliko je a veći ili jednak b.
<=	a <= b	Vraća istinu ukoliko je a manji ili jednak b.
!=	a != b	Vraća istinu ukoliko su a i b različiti.
&&	a && b	Vraća istinu ukoliko je a istinit i b istinit.
||	a || b	Vraća istinu ukoliko su a ili b istiniti.
!	!a	Vraća istinu ukoliko je a laž.

Primijetimo da operator == nije isto što i operator =. Operator = pridružuje desnu stranu lijevoj, a operator == ne mijenja lijevu stranu nego samo provjerava je li jednaka desnoj. Zamjena == s = je česta pogreška u programiraju.

Primijetite da ne postoje operatori => i =<. Svejedno je pišemo li `(a > b)`, `(a >= b && a != b)` ili `(!(a <= b))`. Operator `ili`, tj. `||` se sastoji od dviju ravnih crta, a operator `i`, tj. `&&` se sastoji od dvaju znakova &. Postoji operator koji se sastoji od jedne ravne crte i operator koji se sastoji od jednog znaka &, ali njih ćemo učiti kasnije.

U izrazima poput:

```
    if (a>5 && b<100 || c!=54)
      nekaNaredba;
```

pomoću „prioriteta operatora", koje ćemo spominjati kasnije, odlučuje se koji će se operator prije izvoditi. Kakogod, poželjno je staviti zagrade kako bismo znali misli li se na:

```
    if ((a>5 && b<100) || c!=54)
      nekaNaredba;
```

ili na:

```
    if (a>5 && (b<100 || c!=54))
      nekaNaredba;
```

Slova koja se zapisuju u `char` popisana su po svojoj ASCII vrijednosti (brojčana vrijednost pojedinog slova) i **mogu se uspoređivati** (kasnije će biti detaljnije pojašnjeno).

Zadatak:

Napiši program koji provjerava je li uneseni znak brojka, veliko ili malo slovo.

Rješenje:

```
01. #include <stdio.h>
02.
03. int main() {
04.   char x;
05.   scanf(" %c",&x);
```

```
06.
07.   if (x>='0' && x<='9')
08.     printf("Unesena je znamenka!\n");
09.   else if (x>='a' && x<='z')
10.     printf("Uneseno je malo slovo!\n");
11.   else if (x>='A' && x<='Z')
12.     printf("Uneseno je veliko slovo!\n");
13.   else
14.     printf("Nije unesen alfanumericki znak!\n");
15.
16.   return 0;
17. }
18.
```

Naredba:

```
    if (uvjet1)
      if (uvjet2)
        naredba;
```

se može napisati ovako:

```
    if (uvjet1 && uvjet2)
      naredba;
```

Također postoji i „ljepši“ način pisanja puno `if-else` blokova pomoću naredbe `switch-case`. Ta naredba nije previše važna, štoviše, ne sjećam se kad sam je zadnji puta koristio. Ipak, radi potpunosti knjige, navedena je u primjeru.

Zadatak:

Napiši program koji ispisuje naziv dana u tjednu ovisno o unesenoj brojci.

Rješenje:

```
    #include <stdio.h>

    int main() {
      int x;
      scanf("%d",&x);
      switch (x) {
        case 1: // uvjet kada je x==1
          printf("ponedjeljak\n");
          break;
        case 2:
          printf("utorak\n");
          break;
        case 3:
          printf("srijeda\n");
          break;
        case 4:
          printf("cetvrtak\n");
          break;
```

```
        case 5:
          printf("petak\n");
          break;
        case 6:
          printf("subota\n");
          break;
        case 7:
          printf("nedjelja\n");
          break;
        default: // poput else
          printf("Unijeli ste krivi broj!\n");
      } // kraj switcha
      return 0;
    }
```

1.4.3. for petlja

Programska se petlja u programskom jeziku C zove `for` i ima sljedeću sintaksu:

```
    for (prviDio; drugiDio; treciDio)
      naredbaIliBlok;
```

`for` petlja služi za višestruko ponavljanje istog dijela koda `naredbaIliBlok`.

`prviDio` je naredba koja se odvija samo jednom prije početka iteriranja petlje. `drugiDio` je logički izraz koji predstavlja uvjet za prestanak iteriranja i provjerava se prije početka svake iteracije petlje. `treciDio` je naredba koja se izvrši nakon svake iteracije petlje. Često ta naredba povećava varijablu kojom se broje iteracije petlje (varijabla se zove brojač). Često `prviDio` postavlja brojač na početnu vrijednost (petlja se inicijalizira), `drugiDio` je uvjet za prekid, a `treciDio` je promjena brojača. Što se tiče naredbe ili bloka na koje se `for` petlja odnosi, vrijede ista pravila kao za `if` naredbu. Iza `for` petlje ne ide `;` (iz istih razloga kao kod `if` naredbe).

Zadatak:

Napiši program koji ispisuje brojeve od 1 od 10.

Rješenje:

```
    #include <stdio.h>

    int main() {
      int i;
      for (i=1;i<=10;i=i+1)
        printf("%d ",i);
      return 0;
    }
```

Rezultat izvršavanja je:

```
        1 2 3 4 5 6 7 8 9 10
```

U ovoj `for` petlji brojač je varijabla `i`, a inicijalizirana je naredbom `i=1` koja se izvršava samo jednom na početku izvođenja petlje. Nakon toga se, prije prve iteracije, provjerava uvjet koji glasi `i<=10`. Na početku je `i=1`, znači `1<=10` je istina i izvrši se prva iteracija petlje. U našem slučaju iteracija se sastoji od jedne `printf` naredbe koja ispiše brojač `i`. Nakon što se `printf` izvršio, iteracija je završena, izvršava se promjena brojača, tj. naredba `i=i+1`. Time `i` poprima vrijednost 2.

Prije početka sljedeće iteracije ponovno se provjerava je li zadovoljen uvjet `i<=10`. Sada `i` sadrži vrijednost 2, pa je `2<=10` istina i kreće druga iteracija petlje u kojoj se ponovno ispisuje vrijednost varijable (brojača) `i`, koja je sada 2. Nakon iteracije se ponovno izvršava `i=i+1`, pa `i` postaje 3.

Budući da je `3<=10` istina, počinje treća iteracija u kojoj se ispisuje `i`. Ovog puta je `i==3`, pa se ispisuje 3. Nakon toga se izvršava `i=i+1`, te `i` postaje 4.

I tako dalje...

Na kraju, kad se ispiše `10`, izvršava se `i=i+1`, te `i` postaje `11`, zatim se provjerava uvjet `i<=10`. Budući da je `11<=10` laž, petlja se prekida i program se nastavlja u prvoj liniji nakon `for` petlje i njenog bloka/naredbe.

Zadatak:

Ispiši prvih x potencija broja 2 počevši od nulte potencije. x se unosi.

Primjer unosa:

```
12
```

odgovarajući ispis:

```
1 2 4 8 16 32 64 128 256 512 1024 2048
```

Rješenje:

```
#include <stdio.h>

int main() {
  int x,i,potencija=1;
  scanf("%d",&x);
  for (i=0;i<x;i=i+1) {
    printf("%d ",potencija);
    potencija=potencija*2;
  }
  return 0;
}
```

Slijedi rješenje istog zadatka bez korištenja brojača `i`:

```
#include <stdio.h>

int main() {
  int x,potencija=1;
  scanf("%d",&x);
```

```
    for (;x>0;x=x-1) {
      printf("%d ",potencija);
      potencija=potencija*2;
    }
    return 0;
}
```

Treba primijetiti da je u ovom slučaju ispuštena inicijalizacija brojača, ali ne smijete ispustiti znakove `;` koji odjeljuju tri dijela `for` naredbe. Ukoliko vam ovaj ili prijašnji kôd nije jasan, potrudite se posvetiti mu više vremena i shvatiti kako funkcionira. Isplati se!

Zadatak:

Napravi program koji ispisuje sve potencije broja 2 manje od nekog unesenog broja.

Primjer unosa:

```
5000
```

odgovarajući ispis:

```
1 2 4 8 16 32 64 128 256 512 1024 2048 4096
```

Rješenje:

```
#include <stdio.h>

int main() {
  int x,i;
  scanf("%d",&x);
  for (i=1;i<x;i=i*2)
    printf("%d ",i);
  return 0;
}
```

Proučimo sljedeću `for` petlju:

```
  double d;
  for (d=30;d>-10;d=d-(d+10.5)/2)
    printf("%lf\n",d);
```

Navedeni isječak koda ispisuju:

```
30.000000
9.750000
-0.375000
-5.437500
-7.968750
-9.234375
-9.867188
```

Prijašnji kôd je mogao biti napisan i ovako:

```
  double d=30;
  for (;d>-10;) {
    printf("%lf\n",d);
```

```
    d=d-(d+10.5)/2;
  }
```

> Petlje koje broje određeni broj koraka uobičajeno je napisati tako da broje od `0`, a ne od `1`. Razlog tome bit će pojašnjen kasnije.

Petlja koja treba napraviti 5 iteracija treba izgledati ovako:

```
  int i;
  for (i=0;i<5;i=i+1)
    printf("%d ",i);
```

ispis:

```
    0 1 2 3 4
```

Da smo petlju promijenili na ovaj način:

```
  int i;
  for (i=0;i<5;i=i+1); // dodali smo ';'
    printf("%d ",i);
```

dobili bismo ispisano:

```
    5
```

zato što se u svakoj iteraciji petlje izvršava prazna naredba. Navedeni isječak koda bilo bi smislenije urediti ovako (ukoliko baš to želimo napraviti):

```
  int i;
  for (i=0;i<5;i=i+1)
    ;
  printf("%d ",i);
```

Primijetimo da nakon završetka izvršavanja `for` petlje varijabla `i` ima vrijednost 5, tj. vrijednost koja ne ispunjava uvjet `i<5`.

Sljedeći kôd ispisuje englesku abecedu:

```
  char c;
  for (c='A';c<='Z';c=c+1)
    printf("%c",c);
```

ispis:

```
    ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Ako za uvjet petlje napišemo izraz koji je uvijek istinit, petlja postaje beskonačna i program moramo prekinuti nasilno (npr. sa **ctrl+c**).

```
  for (;2<3;)
    printf("Ja sam beskonacan!\n");
```

Ispis nema kraja:

```
    Ja sam beskonacan!
    Ja sam beskonacan!
    Ja sam beskonacan!
    Ja sam beskonacan!
    ...
```