Programiranje 1

Kontrola toka programa — naredbe za kontrolu postupaka

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

08. prosinca 2022.

Izrazi i naredbe

Izraz je svaka kombinacija **operatora** i **operanada** koju jezik dozvoljava. Svaki **izraz** ima svoju **vrijednost** (određenog **tipa**) koja se dobiva **izvršavanjem** svih **operacija** u izrazu, prema **prioritetu** i **asocijativnosti** operatora.

```
x = 3 ++n printf(...)
```

Poziv funkcije je također **izraz** (čak ako i ne koristimo povratnu vrijednost ukoliko funkcija vraća vrijednost, npr. `printf`).

Program se sastoji od niza **naredbi** koje završavaju znakom točka-zarez (;). Svaki **izraz** iza kojeg slijedi **točka–zarez** postaje **naredba**. To je tzv. **jednostavna**, **osnovna** ili **primitivna** naredba.

Izrazi i naredbe

```
1 int x;
2 x = 3;
3 ++n;
4 printf("%d",x);
```

Postoje i **složene** naredbe i **posebne** naredbe za kontrolu **redoslijeda** izvršavanja ostalih naredbi - naredbe za **kontrolu postupaka** ili **toka**.

Složene naredbe

Složena naredba (blok, blok-naredba ili blok naredbi) je **grupa deklaracija** i **naredbi okružena vitičastim zagradama** { i }.

Primjer složene naredbe.

```
1 {
2 x = 3;
3 ++n;
4 printf("%d",x);
5 }
```

Iza zatvorene zagrade bloka } nema znaka točka-zarez (;).

Složena naredba je sintaktički **ekvivalentna jednoj naredbi**, tj. može se pojaviti na **istim** mjestima gdje se može pojaviti i jednostavna (ili osnovna) naredba.

Uvjetno izvršavanje - `if` naredba

Najjednostavnija `if` naredba ima oblik:

```
1 if(uvjet) naredba;
```

`uvjet` je **logički** izraz.
Redoslijed **izvršavanja:**

- **Prvo** se računa **vrijednost** izraza **uvjet**.
- Ako je ta vrijednost **različita od nule - istina**, **izvršava** se **naredba**.
- Ako je ta vrijednost **jednaka nuli - laž naredba** se ne izvršava i program se nastavlja prvom sljedećom naredbom **iza** `if` naredbe.

`naredba` može biti **jednostavna** ili **složena** naredba.

Uvjetno izvršavanje - `if` naredba

Pravilo pisanja:

- izraz `uvjet` se piše u **okruglim** zagradama,
- odmah **iza ključne riječi** `if` koja označava **početak** naredbe.

Isto pravilo pisanja vrijedi za **kontrolne** dijelove u svim **uvjetnim** naredbama i **petljama**.

- pišu se u **okruglim** zagradama **iza ključne riječi**,
- na **početku** naredbe, na **kraju** samo kod `do-while`

Primjer if naredbe.

```
1 int x;
2 ...
3 if (x>0) printf("x␣=␣%d\n",x);
4 ++x;
```

Uvjetno izvršavanje - `if` naredba

Gornji primjer radi sljedeće:

- **ako i samo ako** je vrijednost varijable `x` **pozitivna**, **ispiši** tu vrijednost. **Inače** ne radi ništa.
- Povećaj (inkrementiraj) vrijednost od x za 1 (**neovisno** o uvjetu u `if`, tj. neovisno o **vrijednosti** x).

Primjer: Želimo **osigurati** da je $i \leq j$. Ako to nije, **zamijenimo** vrijednosti od i i j.

```
int i,j, tmp;
...
if (i>j){ //zamjena vrijednosti
    tmp = i;
    i = j;
    j = tmp;
} //sigurno i<=j
```

Paziti na redoslijed **pridruživanja**!

if-else naredba

if-else naredba.

```
1 if (uvjet)
2     naredba_1;
3 else
4     naredba_2;
```

Događa se **uvjetno** izvršavanje **jedne** od **dviju** naredbi.

Ako izraz **uvjet** ima vrijednost **istine**, onda se **izvršava** `naredba_1`, inače (**uvjet** ima vrijednost **laž**) se **izvršava** `naredba_2`.

Svaki `else` pripada **najbližem** (prethodnom) `if`-u (bitno kod ugniježdenih `if` i `if-else` naredbi). Prevoditelj uvijek **obrađuje najdulju** smislenu jezičnu cjelinu (`if-else` je dulji od `if`).

if-else naredba

Ugniježdena if i if-else naredba.

```
1 if (n>0)
2     if(a>b)
3         z = a;
4     else z = b;
```

```
if(n>0)
    if(a>b) z = a;
    else
        z=b;
```

Obje varijante rade **identično**.

Pripadnost `else`-a naredbi `if`, **mijenjamo grupiranjem** u složenu naredbu, korištenjem **vitičastih zagrada**.

Ugniježdena if i if-else naredba s grupiranjem.

```
1 if (n>0){
2     if(a>b)
3         z = a; }
4 else
5     z = b;
```

```
if(n>0){
  if(a>b)
    z = a; }
  else
    z = b;
```

Obje varijante rade **identično**.

Izlaz iz programa, funkcija `exit`

U datoteci zaglavlja `<stdlib.h>` deklarirana je funkcija

```
void exit(int status)
```

Poziv funkcije `exit(status);` **zaustavlja** izvršavanje programa i vrijednost **status** predaje **operacijskom sustavu**.
Istu stvar radi i poziv `return status;` unutar funkcije `main`.
Glavna razlika je što poziv `exit(status);` u **proizvoljnoj** funkciji **zaustavlja** izvršavanje programa.

Zaglavlje `<stdlib.h>` sadrži definiciju dvije **standardne** vrijednosti za status: `EXIT_SUCCESS` i `EXIT_FAILURE`.

- 0 i `EXIT_SUCCESS` označavaju **uspješno** izvođenje programa.
- `EXIT_FAILURE` signalizira **neuspješno** izvođenje (negdje unutar programa je došlo do nedozvoljenih, krivih vrijednosti, nedostajućih resursa i slično).

Izlaz iz programa, funkcija exit

Korištenje exit kao reakciju na nedozvoljene vrijednosti.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int x,y;
4  scanf("%d␣%d",&x,&y);
5  //ucitamo vrijednosti u varijable x i y
6  //zelimo izracunati y = y/x;
7  if(!x){ //nedopustena vrijednost, x je 0
8      printf("Greska:␣djelitelj␣jednak␣nuli!\n");
9      exit(EXIT_FAILURE);
10 }
11 else
12     y/=x;
```

if naredba i uvjetni operator

Sljedeće dvije **naredbe** su **ekvivalentne**.

```
max = a>=b ? a:b;
```

```
if(a>=b)
    max = a;
else
    max = b;
```

Obje postavljaju `max` na **maksimum** vrijednosti varijabli a i b.

Zadatak: Napišite analogne naredbe koje postavljaju `min` na **minimum** vrijednosti varijabli a i b. **Uputa:** Koristite $<=$ umjesto $>=$.

Višestruki izbor `if-else` naredbama

Naredbe `if-else` mogu se **ugnijezditi**.

Dvije if-else naredbe. Druga počinje iza else dijela prve.

```
if(uvjet_1)
      naredba_1;
else if(uvjet_2)
      naredba_2;
else
     naredba_3;
```

Primjer: Učitavaju se **dva broja** (tipa `double`) i **jedan znak** koji **označava** osnovnu računsku operaciju (+,-,*,/). U ovisnosti o učitanom znaku, **izvršava** se **jedna** od te **četiri** operacije na učitanim **brojevima** (*jednostavni kalkulator*).

Jednostavni kalkulator if-else naredbama

```
1  #include <stdio.h>
2
3  int main(void){
4
5  double a,b;
6  char operacija;
7
8  printf("Upisati␣prvi␣broj:␣");
9  scanf("␣%lf", &a);
10 printf("Upisati␣drugi␣broj:␣");
11 scanf("␣%lf", &b);
12 printf("Upisati␣operaciju␣(+,-,*,/):␣");
13 scanf("␣%c",&operacija);
14
15 if(operacija == '+')
16     printf("%f\n",a+b);
```

Jednostavni kalkulator `if-else` naredbama

```
17 else if(operacija == '-')
18    printf("%f\n",a-b);
19 else if(operacija == '*')
20    printf("%f\n",a*b);
21 else if(operacija == '/')
22    printf("%f\n",a/b);
23 else
24    prinf("Nedopustena␣operacija!\n");
25
26    return 0;
27 }
```

Jednostavni kalkulator `if-else` naredbama

Primjer izvršavanja programa:

```
Upisati prvi broj: 21\n
Upisati drugi broj: 13\n
Upisati operaciju (+, -, *, /): /\n
1.615385
```

Program *jednostavni kalkulator* prima ulaz u **postfix obliku** (prvi operand, drugi operand, operacija).

Zadatak: Preuredite **ulaz** tako da izraz pišemo u uobičajenom **infix** obliku (prvi operand, operacija, drugi operand). Omogućite da izraz možemo pisati i u jednom redu (npr. `10*15\n`).

Napomene vezane uz čitanje i pisanje:

- Kod čitanja prvo preskačemo bjeline ispred broja (ako ih ima). Bjeline su praznina, tabulatori (\t, \v) i znakovi \n, \r, \f.

Jednostavni kalkulator `if-else` naredbama

- Oznaka konverzije `%c` služi za čitanje i pisanje jednog znaka (objekt tipa `char`). Učitava se prvi sljedeći znak na ulazu **bez preskakanja bjelina**.
- Ukoliko želimo preskočiti bjeline pri učitavanju znaka, format glasi `" %c"` (imamo prazninu na početku).
- Početne praznine ispred `%lf` nisu potrebne pošto se **praznine automatski zanemaruju** pri učitavanju cijelih i realnih brojeva.

Zadatak: ispitajte što se događa ukoliko ispustimo vodeću prazninu u formatu za čitanje znaka, tj. koristimo format `"%c"`.

Što se nalazi iza **drugog** broja na ulazu? (\n)

Višestruki izbor - switch naredba

Naredba switch slična je nizu **ugniježdenih** if-else naredbi. Opći oblik naredbe je:

```
1 switch (izraz) {
2       case konstanta_1: naredbe_1;
3       /* moze vise naredbi! */
4       case konstanta_2: naredbe_2;
5       ...
6       case konstanta_n: naredbe_n;
7       default: naredbe;
8 }
```

Vrijednost izraza **određuje** ili **selektira** odgovarajući **slučaj** (**case**) i eventualno slučajeve **ispod** njega.

Višestruki izbor - `switch` naredba

Osnovna pravila kod pisanja `switch` naredbe:

- `izraz` u `switch` naredbi mora imati `cjelobrojnu` vrijednost (`char`, `int` ili `enum`).
- Nakon svake ključne riječi `case` pojavljuje se **cjelobrojna konstanta** ili **konstantni izraz**, a iza toga mora biti znak `:` (dvotočka). Ovi izrazi se računaju prilikom prevođenja. Napomena. **Ne smije** biti **varijabla**, čak i kad ima `const`.

Redoslijed izvršavanja u `switch` naredbi:

- **Prvo** se računa **vrijednost** izraza **izraz**.
- **Zatim** se **provjerava** je li dobivena **vrijednost jednaka** jednoj od **međusobno različitih** konstanti `konstanta_1`, ..., `konstanta_n`.

Višestruki izbor - `switch` naredba

- Ako je `izraz = konstanta_i` onda,
  - program **izvodi** naredbe `naredbe_i` (može ih biti više, bez vitičastih zagrada)
  - zatim **izvodi sve naredbe koje dolaze iza, u ostalim slučajevima** do prve naredbe `break` (ako postoji) ili do kraja `switch` naredbe. Nakon toga, program nastavlja **prvom naredbom** iza `switch` naredbe.
- Ako **izraz** nije jednak niti jednoj navedenoj konstanti, program **izvršava naredbe** iza **ključne** riječi `default` (ako postoji) i **sve naredbe iza njih** do prve `break` naredbe ili do kraja `switch` naredbe.

Naredba `break` **prekida** izvršavanje naredbi **slučajeva** u kodu iza nje.

Višestruki izbor - `switch` naredba

- Slučaj `default` **ne mora** nužno biti prisutan u `switch` naredbi. Ako **nije** i ako **nema** podudaranja izraza i neke od navedenih konstanti, program **izvršava prvu** naredbu **iza** `switch` (ne izvršava niti jednu naredbu iz `switch`).
- Slučajevi oblika `case konstanta_i` i slučaj `default` (ako ga ima) mogu biti napisani **bilo kojim redom** (npr. `default` može biti na **prvoj** poziciji, odmah iza `switch`).

Primjer: realizirajmo program s **izborom** aritmetičke operacije (*jednostavni kalkulator*) korištenjem `switch` naredbe.

Jednostavni kalkulator switch naredbom

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5
6      double a, b;
7      char operacija;
8
9      printf("Upisati␣prvi␣broj:␣");
10     scanf("␣%lf", &a);
11     printf("Upisati␣drugi␣broj:␣");
12     scanf("␣%lf", &b);
13     printf("Upisati␣operaciju␣(+,␣-,␣*,␣/):␣");
14     scanf("␣%c", &operacija);
```

Jednostavni kalkulator switch naredbom

```
15    switch (operacija) {
16       case '+': printf("%f\n", a + b);
17                break;
18       case '-': printf("%f\n", a - b);
19                break;
20       case '*': printf("%f\n", a * b);
21                break;
22       case '/': printf("%f\n", a / b);
23                break;
24       default: printf("Nedopustena␣operacija!\n");
25    }
26  return 0;
27 }
```

Ispuštanje break naredbe

Ukoliko neki case ne sadrži break naredbu, izvršavaju se naredbe case-a ispod. Pošto case 0 i case 1 ne sadrže naredbu break, za $i = 0$ će se izvršiti sve naredbe unutar case 0, case 1 i case 2.

```
unsigned int i;
...
switch (i) {
    case 0:
    case 1:
    case 2: printf("i␣<␣3\n");
            break;
    case 3: printf("i␣=␣3\n");
            break;
    default: printf("i␣>␣3\n");
}
```

while petlja

while petlja ima oblik:

```
while (izraz) naredba;
```

Sve dok je izraz istinit (različit od 0), izvršava se naredba.

Ispisivanje brojeva 0..9 korištenjem while petlje.

```
1    i = 0;
2    while(i<10){
3        printf("%d\n",i);
4        ++i;
5    }
```

while petlja najčešće se koristi kad se **broj** ponavljanja **ne zna** unaprijed, već ovisi o uvjetu **izraz**.

Primjer: program čita **niz** realnih brojeva **različitih** od **nule**, sve dok se ne ispiše **nula**, i računa **srednju vrijednost** tog niza (bez zadnje nule, nula je samo oznaka za kraj niza).

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int n = 0;
6      double sum = 0.0, x;
7      printf("␣Upisite␣niz␣brojeva␣!=␣0,"
8      "␣i␣nulu␣za␣kraj.\n");
9      printf("␣x[0]␣=␣");
10     scanf("%lf", &x);
```

while petlja

```
11      while (x != 0.0) {
12          sum += x;
13          printf("␣x[%d]␣=␣", ++n);
14          scanf("%lf", &x);
15      }
16    if(n>0) sum /= n;
17    printf("␣Srednja␣vrijednost␣=␣%f\n", sum);
18    return 0;
19  }
```

Uvjet `if(`$n > 0$`)` je **važan**. Bez njega imamo dijeljenje s nulom, ukoliko je prvi učitani broj odmah 0.

`for` petlja

`for` **petlja** ima oblik:

```
for (izraz_1; izraz_2; izraz_3) naredba;
```

ekvivalentna je s:

```
izraz_1;
while (izraz_2) {
      naredba;
      izraz_3;
}
```

do na eventualno korištenje naredbe `continue` (o kojoj ćemo govoriti kasnije).
`izraz_2` se interpretira kao **logički izraz** a ostala dva mogu biti proizvoljni izrazi (pretvaraju se u naredbe).

`for` petlja

`for` petlja se najčešće koristi za ponavljanje pod kontrolom **brojača** (određeni, poznati broj puta).

Ukoliko imamo `for` petlju s brojačem:

```
for (izraz_1; izraz_2; izraz3) naredba;
```

- **izraz_1** - predstavlja **inicijalizaciju** brojača na početku petlje.
- **izraz_2** - predstavlja **uvjet** izvršavanja petlje.
- **izraz_3** - predstavlja **pomak** brojača na kraju svakog prolaza kroz petlju.

for petlja

Primjer: standardni pomak brojača za 1 **unaprijed**.

```
for(brojac = 1; brojac < 5; ++brojac) ...
```

Primjer: brojač možemo mijenjati i drugačije:

```
for(brojac = 1; brojac < 5; brojac+=2) ...
```

Svaki od izraza unutar `for` petlje **smijemo** i **ispustiti**.
Beskonačna `for` petlja koja ništa ne radi se zapisuje kao:

```
for(;;);
```

Primjer: Važno je razlikovati nalazi li se točka-zarez (`;`) direktno nakon `for` naredbe ili ne (drugačije izvođenje programa).

for petlja

```
for (brojac = 1; brojac < 5; ++brojac)
  printf("brojac = %d\n", brojac); //unutar for petlje

Ispis:
brojac = 1
brojac = 2
brojac = 3
brojac = 4

for (brojac = 1; brojac < 5; ++brojac); //prazna naredba
  printf("brojac = %d\n", brojac); //sada je brojac = 5

Ispis:
brojac = 5
```

do-while petlja

do-while petlja ima oblik:

```
do
   naredba;
while (izraz);
```

naredba se ponavlja (izvršava) sve dok izraz ima vrijednost **istine**, tj. sve dok je **različit** od **nule**.

Za razliku od while petlje gdje se vrijednost izraza **računa** i **provjerava** na **vrhu** petlje, **prije** naredbe, u do-while petlji se vrijednost **računa** i **provjerava** na kraju prolaza kroz petlju (**iza** naredbe).

Naredba se u do-while petlji izvršava **barem jednom** (**prije prve** provjere izraza).

do-while petlja

Ispis brojeva 0-9 korištenjem do-while petlje.

```
1   i = 0;
2   do {
3         printf("%d\n", i);
4         ++i;
5 } while (i < 10);
```

do-while se obično koristi kada trebamo obraditi **niz** podataka u kojem **barem jedan** podatak treba bezuvjetno obraditi.

Naredba break

Naredba `break` služi za:

- izlazak iz `switch` naredbe
- **zaustavljanje** ili **prekidanje** petlje. Može se koristiti unutar `for`, `while` i `do-while` petlji.

Nakon izvršavanja naredbe `break`, izvršavanje programa se prenosi na **prvu naredbu** iza `switch` naredbe ili **petlje unutar** koje se taj `break` nalazi. Prekida se izvršavanje samo najbliže okolne `switch` naredbe ili petlje.

Obrada niza brojeva s oznakom za kraj niza.

```
1   int i;
2   while(1) {   /*beskonacna petlja*/
3         scanf("%d\n", &i);   /*ucitavanje broja.*/
4         if(i<0) break;       /*test kraja.*/
5         ...                  /*obrada broja.*/
6 }/*nakon break, izvodi se naredba iza petlje*/
```

Naredba `continue`

Naredba `continue` može se koristiti unutar `for`, `while` i `do-while` za **skraćivanje** izvođenja tijela petlje **preskakanjem** preostalih naredbi u petlji.

Nakon izvršavanja naredbe `continue`:

- **preostali** dio tijela petlje (**iza continue**) se **preskače** i program **nastavlja** sa sljedećim prolazom (iteracijom) kroz petlju.
- **sljedeća** naredba koja se **izvršava** je:
  - **test uvjeta** u `while` i `do-while` petlji,
  - **pomak brojača** (**izraz_3**) u `for` petlji.

  Tu je **razlika** između `while` i pripadne `for` petlje.

Korištenje naredbe `continue` **nema smisla** unutar `switch` naredbe.

Naredba continue

Kod koji preskače (ne obrađuje) negativne vrijednosti.

```
1   int i;
2   while(1) {  /*beskonacna petlja*/
3       scanf("%d\n", &i);  /*ucitavanje broja.*/
4       if(i<0) continue; /preskakanje negativnog.*/
5       ...                 /*obrada broja.*/
6 }/*nakon break, izvodi se naredba iza petlje*/
```

Trebamo neku drugu **oznaku kraja niza** u dijelu koda koji obrađuje **nenegativne brojeve**. Što treba dodati u gornji program ukoliko je ta oznaka **nula**?

Naredba goto

Naredba `goto` **prekida sekvencijalno** izvršavanje programa i **nastavlja** izvršavanje s **naredbom** koja je **označena labelom** navedenom u **toj** `goto` naredbi (tzv. skok).

```
goto label;
```

gdje je **label identifikator** koji služi za **označavanje naredbe** kojom se **nastavlja** izvršavanje programa. Označavanje se vrši kao:

```
label: naredba;
```

Labela na koju se vrši skok **mora biti** unutar **iste funkcije** kao i `goto` naredba.

U pravilu se `goto` koristi **samo** za **obradu grešaka**.

Naredba goto

```
1   double x, s = 0.0;
2   while (1) {
3       scanf("%lg", &x);
4       if (x < 0.0) goto error;
5       if (x = 0.0) break;
6       s += sqrt(x); /* zbraja korijene. */
7     }
8     ... /* i normalni zavrsetak posla. */
9   error:
10      /* reakcija na gresku. */
11    printf("Greska: negativan broj!\n");
12    exit(EXIT_FAILURE);
```

Naredba goto

Naredbe `break`, `continue` i naredbe za **kontrolu toka** mogu se **izvesti** pomoću naredbe `goto`. Prevoditelj ih prevodi koristeći `jump` instrukcije strojnog jezika (slična funkcionalnost kao `goto`).

```
1   for(...){
2     ...
3     if(...) continue;
4     ...
5   }
6   //ekvivalentno kao
7
8   for (...) {
9     ...
10   if (...) goto cont;
11    ...
12    cont: ; // prazna naredba na dnu petlje
13  }
```

Naredba goto

Slično se pomoću `goto` može izvesti `continue` unutar `while` i `do-while` petlje.

Zadatak: Napravite slične transformacije za `break` naredbu u petljama i `switch`.

Program koji ima **puno** `goto` naredbi je **teže razumjeti** od programa koji ne koriste `goto`. Upotrebu `goto` **treba izbjegavati**. **Zabranjeno korištenje na kolokvijima!**

`goto` naredba služi samo za reakciju na **specijalne** slučajeve i **greške**. Koristi se i kada veće dijelove programa treba preskočiti skokom **unaprijed**, te da se izbjegne dodatno uvlačenje dijelova koda `if-else` naredbama koje bi **smanjilo** preglednost.

Korištenje skokova **unatrag**, odnosno implementacija petlji koristeći `goto` može dovesti do raznih **teško uočljivih** grešaka.