Uvod u računarstvo
10. predavanje

Saša Singer

singer@math.hr

web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

- Kontrola toka programa:
  - Izrazi i naredbe.
  - Uvjetne naredbe `if`, `if-else`, `switch`.
  - Petlje `while`, `for`, `do-while`.
  - Naredbe `break` i `continue`.
  - Naredba `goto`.
  - Operator zarez.
  - Petlja `do-while`.
  - Naredbe `break`, `continue` i `goto`.

Izrazi i naredbe

Izraz je svaka kombinacija operatora i operanada koju jezik dozvoljava. Svaki izraz ima svoju vrijednost koja se dobiva

- izvršavanjem svih operacija u izrazu,

redoslijedom prema prioritetu.

Primjer:

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

Poziv funkcije je, također, izraz — čak i kad "odbacujemo" povratnu vrijednost (ako je ima).

Izrazi i naredbe (nastavak)

Program se, općenito, sastoji od niza naredbi.

- Naredbe završavaju znakom točka–zarez `;`.

Svaki izraz iza kojeg slijedi točka–zarez postaje naredba (tzv. osnovna ili primitivna naredba).

Primjer:

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

Osim ovih, postoje još i složene naredbe, te posebne naredbe (s imenom) za kontrolu redoslijeda izvršavanja ostalih naredbi (tzv. naredbe za kontrolu postupaka).

Složena naredba

Složena naredba (blok, blok–naredba ili blok naredbi) je

- grupa deklaracija i naredbi, zatvorena u vitičaste zagrade `{` i `}`.

Primjer:

```
{x = 3;
 n++;
 printf(...);}
```

Uočiti da nema točka–zareza iza zatvorene zagrade `}`.

Složena naredba je sintaktički ekvivalentna jednoj naredbi, tj. može se pojaviti na istim mjestima gdje se može pojaviti i jednostavna naredba.

Uvjetno izvršavanje — `if` naredba

Najjednostavnija `if` naredba ima oblik:

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

gdje je `uvjet` aritmetički (ili pokazivački) izraz.

Redoslijed izvršavanja:

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

`if` naredba (nastavak)

Primjer:

```
int x;
...
if (x > 0) printf("\n x= %d \n", x);
++x;
```

ispisuje samo pozitivne vrijednosti varijable `x`.

Ovaj oblik `if` naredbe je:

- uvjetno izvršavanje jedne naredbe.

Alternative su (ovisno o uvjetu): izvrši ili ne.

`if` naredba (nastavak)

Primjer: želimo osigurati da je `i <= j`. Ako to nije, zamijenimo vrijednosti od `i` i `j`.

```
int i, j, temp;
...
if (i > j) {     /* zamjena vrijednosti */
    temp = i;
    i = j;
    j = temp;
}
```

Napomena: paziti na redoslijed pridruživanja (koga kamo kopiramo)!

`if-else` *naredba*

`if-else` naredba ima oblik:

```
if (uvjet)
    naredba1;
else
    naredba2;
```

Ako izraz `uvjet`

- ima vrijednost istine, onda se izvršava `naredba1`,
- a u suprotnom `naredba2`.

Ovo je:

- uvjetno izvršavanje jedne od dviju naredbi.

Alternative su (ovisno o uvjetu): izvrši jednu ili drugu.

Kojem `if` pripada `else`?

Problem: Kad imamo ugniježđene `if` i `if-else` naredbe, kojem `if` pripada `else` — prvom ili drugom?

Pravilo: Svaka `else` naredba pripada najbližoj (prethodnoj) `if` naredbi. (Razlog: kompajler u danom trenu uvijek "jede" najdulju moguću jezičku cjelinu.)

Primjer:

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

```
if (n > 0)
    if (a > b) z = a;
else  /* LOS STIL */
    z = b;
```

Obje varijante, naravno, rade isto.

Kojem `if` pripada `else`? (nastavak)

Pripadnost mijenjamo grupiranjem u složenu naredbu, tj. korištenjem vitičastih zagrada.

Primjer:

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

```
if (n > 0) {
    if (a > b)
        z = a;
}
    else  /* LOS STIL */
        z = b;
```

Obje varijante, opet, rade isto.

Funkcija `exit`

Funkcija:

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

deklarirana je u datoteci zaglavlja `<stdlib.h>`. Ona

- zaustavlja izvršavanje programa

i vrijednost `status` predaje operacijskom sustavu.

Standardno, `status = 0` znači da je program uspješno završen, a vrijednost različita od nule znači da se program zaustavio zbog greške.

Funkcija `exit` (nastavak)

Primjer:

```
#include <stdlib.h>
    ...
    if (!x) {
        printf("Djelitelj jednak nuli!\n");
        exit(-1);
    }
    else
        y /= x;
```

Uočiti: `!x` je istina ako i samo ako je `x == 0`.

`if` *naredba i uvjetni operator*

Sljedeće dvije naredbe su ekvivalentne:

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

i

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

Višestruki izbor `if-else` naredbama

Naredbe `if-else` mogu se ugnijezditi.

Primjer: dvije `if-else` naredbe, druga iza `else` od prve.

```
if (uvjet1)
    naredba1;
else if (uvjet2)
    naredba2;
else
    naredba3;
```

Primjer. Učitavaju se dva broja i jedan znak koji označava osnovnu računsku operaciju (`z`, `o`, `m`, `d`). U ovisnosti o učitanom znaku izvršava se jedna od četiri računske operacije (`+`, `-`, `*`, `/`) na učitanim brojevima.

Višestruki izbor `if-else` naredbama (nastavak)

```
#include <stdio.h>

int main(void)
{
    float a, b;
    char operacija;

    printf("Upisati prvi broj: ");
    scanf(" %f", &a);
    printf("Upisati drugi broj: ");
    scanf(" %f", &b);
    printf("Upisati operaciju: z, o, m, d:\n");
    scanf(" %c", &operacija);
```

Višestruki izbor `if-else` naredbama (nastavak)

```
    if (operacija == 'z')
        printf("%f\n", a + b);
    else if (operacija == 'o')
        printf("%f\n", a - b);
    else if (operacija == 'm')
        printf("%f\n", a * b);
    else if (operacija == 'd')
        printf("%f\n", a / b);
    else
        printf("Nedopustena operacija!\n");

    return 0;
}
```

Višestruki izbor — `switch` naredba

Naredba `switch` slična je nizu ugniježđenih `if-else` naredbi. Opći oblik je:

```
switch (izraz) {
    case konstanta_1: naredbe_1;
                      /* moze vise naredbi! */
    case konstanta_2: naredbe_2;
    ...
    case konstanta_n: naredbe_n;
    default:          naredbe;
}
```

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

`switch` *naredba (nastavak)*

Osnovna pravila:

- `izraz` u `switch` naredbi mora imati cjelobrojnu vrijednost (tipovi `char`, `int` ili `enum`).
- Nakon ključne riječi `case` pojavljuju se cjelobrojne konstante ili konstantni izrazi, iza koji mora biti znak `:` (dvotočka).

Redoslijed izvršavanja u `switch` naredbi:

- Prvo se računa vrijednost izraza `izraz`.
- Zatim se provjerava je li dobivena vrijednost jednaka jednoj od konstanti: `konstanta_1`, . . . , `konstanta_n`. Ove konstante moraju biti međusobno različite.

`switch` *naredba (nastavak)*

- Ako je `izraz = konstanta_i`
  - program nastavlja naredbama `naredbe_i` (može ih biti više, bez vitičastih zagrada),
  - i svim naredbama koje dolaze iza njih (u ostalim slučajevima ispod tog),

  sve do prve `break` naredbe (ako je ima) 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 `break` ili do kraja `switch` naredbe.

`switch` *naredba (nastavak)*

- Slučaj `default` ne mora nužno biti prisutan u `switch` naredbi. Ako nije i ako nema podudaranja izraza i konstanti,
  - program nastavlja prvom naredbom iza `switch` naredbe,

  tj. ne izvršava niti jednu naredbu iz `switch`.
- Slučajevi oblika `case konstanta_i` i slučaj `default` (ako ga ima) mogu biti napisani bilo kojim redom.
  - Na primjer, `default` može biti i prvi, na samom početku `switch` naredbe.

Primjer. Program s izborom aritmetičke operacije (od malo prije) sad realiziramo `switch` naredbom (preglednije).

switch *naredba (nastavak)*

```
#include <stdio.h>

int main(void)
{
    float a, b;
    char operacija;

    printf("Upisati prvi broj: ");
    scanf(" %f", &a);
    printf("Upisati drugi broj: ");
    scanf(" %f", &b);
    printf("Upisati operaciju: z, o, m, d:\n");
    scanf(" %c", &operacija);
```

switch *naredba (nastavak)*

```
    switch (operacija) {
        case 'z': printf("%f\n", a + b);
                  break;
        case 'o': printf("%f\n", a - b);
                  break;
        case 'm': printf("%f\n", a * b);
                  break;
        case 'd': printf("%f\n", a / b);
                  break;
      default: printf("Nedopustena operacija!\n");
    }
    return 0;
}
```

Ispuštanje `break` naredbe

Ispušteni `break` vodi na "propadanje kôda" u niži `case` blok.

Primjer: dio programa koji ispisuje korektne poruke!

```
int i;
...
switch (i) {
    case 1:
    case 2:
    case 3: printf("i < 4\n");
            break;
    case 4: printf("i = 4\n");
            break;
    default: printf("i > 4\n");
}
```

`while` *petlja*

`while` petlja ima oblik:

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

`naredba` se izvršava sve dok je `izraz` istinit (različit od 0).

Primjer: sljedeći dio programa ispisuje brojeve 0, 1, . . . , 9.

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

`while` petlja najčešće se koristi kad se broj ponavljanja ne zna unaprijed, već je pod kontrolom uvjeta `izraz`.

while *petlja (nastavak)*

Primjer. Program čita niz brojeva različitih od nule, sve dok se ne upiše nula, i računa srednju vrijednost tog niza (bez zadnje nule).

```
#include <stdio.h>
int main(void)
{
    int i = 0;
    double sum = 0.0, x;

    printf(" Upisite niz brojeva != 0,"
           " i nulu za kraj.\n");
    printf(" x[0]= ");
    scanf("%lf", &x);
```

while *petlja (nastavak)*

```
    while (x != 0.0) {
        sum += x;
        printf(" x[%d]= ", ++i);
        scanf("%lf", &x);
    }
    sum /= i;
    printf(" Srednja vrijednost = %f\n", sum);
    return 0;
}
```

Oprez! Što se događa ako odmah upišemo nulu kao prvi broj, tj. imamo "prazan" niz?

- Dijeljenje s nulom!

Popravak: treba dodati test `if (i > 0) ...`

`for` ***petlja***

`for` petlja ima oblik:

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

i ekvivalentna je s

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

Beskonačna petlja koja ne radi ništa (default `izraz2 = 1`):

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

`for` *petlja (nastavak)*

Primjer. Uočite razliku koju radi `;` na kraju retka s `for` naredbom.

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

Ovo će ispisati redom `brojac = 1` do `brojac = 4`.

Razlog: `printf` je unutar petlje.

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

Ovo će ispisati `brojac = 5`.

Razlog: `printf` je izvan, iza petlje.

Operator zarez ,

Operator zarez , separira dva izraza. Izrazi separirani zarezom izračunavaju se slijeva nadesno i rezultat čitavog izraza je vrijednost desnog izraza.

Primjer:

```
i = (i = 3, i + 4);
```

daje rezultat `i = 7`. Operator zarez uglavnom se koristi u `for` naredbi.

Prioritet operatora , je niži od operatora pridruživanja (tj. na dnu tablice prioriteta). Zato su nužne zagrade na desnoj strani operatora = u gornjem primjeru.

Asocijativnost je, naravno (po ideji), slijeva nadesno, L $\rightarrow$ D.

Operator zarez (nastavak)

Primjer. Funkcija `invertiraj` invertira niz znakova.

```
#include <string.h>

void invertiraj(char s[]) {
    int c, i, j;

    for (i = 0, j = strlen(s)-1; i < j; ++i, --j)
    {
        c = s[i]; s[i] = s[j]; s[j] = c;
    }
}
```

Funkcija `strlen` deklarirana je u datoteci zaglavlja `<string.h>` i daje duljinu znakovnog niza (bez nul-znaka).

`do-while` *petlja*

`do-while` petlja ima oblik:

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

`naredba` se 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 izraza

- računa i provjerava na "dnu" (kraju) prolaza kroz petlju, iza naredbe.

`do-while` *petlja (nastavak)*

Naredba u `do-while` petlji se stoga izvršava barem jednom.

Primjer: dio programa koji ispisuje brojeve 0, 1, . . . , 9.

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

Naredba `break`

Naredba **`break`** služi za:

- zaustavljanje ili prekidanje petlje
- i izlazak iz **`switch`** naredbe.

Može se koristiti unutar **`for`**, **`while`** i **`do-while`** petlji.

Pri nailasku na naredbu **`break`**,

- "kontrola" programa prenosi se na prvu naredbu iza petlje ili **`switch`** naredbe unutar koje se taj **`break`** nalazi.
- "Izlazak" se odnosi samo na najbližu okolnu petlju ili **`switch`**.

Naredba `break` (nastavak)

Primjer:

```
int i;
while (1) {
    scanf("%d", &i);
    if (i < 0) break;
    ...
}
```

`while (1)` je beskonačna petlja.

- Iz nje se izlazi ako se učita negativan broj.

Izvršavanje se nastavlja prvom naredbom iza ove `while` petlje.

Naredba `continue`

Naredba `continue` koristi se unutar `for`, `while` i `do-while` petlji za "skraćenje" pojedinog prolaza kroz petlju, preskakanjem dijela naredbi u petlji.

Nakon nailaska na `continue`,

- preostali dio tijela petlje (iza `continue`) se preskače i program nastavlja sa sljedećim prolazom kroz petlju.
- Preciznije, sljedeća naredba koja se izvršava je:
  - test uvjeta u `while` i `do-while`,
  - povećavanje brojača (`izraz3`) u `for`.

Uočite da naredba `continue` nema smisla u `switch` naredbi (za razliku od `break`).

Naredba `continue` *(nastavak)*

Primjer. Po ugledu na prethodni primjer, kôd koji preskače negativne vrijednosti (i ne obrađuje ih) mogao bi se izvesti naredbom `continue`:

```
int i;
while (1) {
    scanf("%d", &i);
    if (i < 0) continue;
    ...
}
```

Sada nam u dijelu kôda koji obrađuje nenegativne brojeve treba neki drugi način izlaza iz petlje (neka druga "oznaka" za kraj niza).

Naredbe `break` *i* `continue` *(primjer)*

Primjer. Što ispisuje sljedeći program s `break` naredbom, a što ako, umjesto `break`, piše `continue` naredba?

```
#include <stdio.h>
int main(void) {
  for (putchar('1'); putchar('2'); putchar('3')) {
      putchar('4');
      break;    /* continue; */
      putchar('5');
      }
  return 0;
}
```

Izlaz s `break` je: `124`.
Izlaz s `continue` je: `124324324...` (`324` do beskonačnosti).

Naredba `goto`

Naredba `goto` prekida sekvencijalno izvršavanje programa i

- nastavlja izvršavanje s naredbom koja je označena labelom koja se pojavljuje u `goto`.

Oblik joj je:

```
goto label;
```

gdje je `label` identifikator koji služi za označavanje naredbe kojom se nastavlja program. Sintaksa označavanja je:

```
label: naredba;
```

Labela na koju se vrši skok mora biti unutar iste funkcije kao i `goto` naredba, tj. pomoću `goto` se ne može izaći iz funkcije.

Naredba goto (nastavak)

Primjer. U pravilu, goto služi samo za reakcije na greške.

```
    scanf("%d", &i);
    while (i <= 100) {
        ...
        if (i < 0) goto error;
        ...
        scanf("%d", &i);
    }
    ...
error: { /* detekcija greske */
       printf("Greska : negativna vrijednost!\n");
       exit(-1);
       }
```

Naredba `goto` (nastavak)

Naredbe `break` i `continue` mogu se izvesti pomoću `goto` naredbe. Isto vrijedi i za sve naredbe za kontrolu toka.

- Prevoditelj ih zaista tako i prevodi, koristeći goto (odnosno, jump) instrukcije na nivou strojnog jezika ili Assemblera.

Primjer: kôd s `continue` naredbom u `for` petlji

```
for (...) {
    ...
    if (...) continue;
    ...
}
```

je ekvivalentan s

Naredba `goto` (nastavak)

```
for (...) {
    ...
    if (...) goto cont;
    ...
    cont: ;
}
```

Slično je i za `continue` unutar `while` ili `do-while` petlje.

Napomena: program koji ima puno `goto` naredbi

- bitno je teže razumjeti (pročitati)

od programa koji ne koristi `goto`. Stoga upotrebu `goto` naredbe treba izbjegavati.