

# *Programiranje 1*

## *8. predavanje*

Saša Singer

`singer@math.hr`

`web.math.pmf.unizg.hr/~singer`

PMF – Matematički odsjek, Zagreb

# Sadržaj predavanja

- Naredbe — kontrola toka programa:
  - Izrazi i naredbe.
  - Uvjetne naredbe if, if-else, switch.
  - Petlje while, for, do-while.
  - Naredbe break i continue.
  - Naredba goto.

# Informacije

## Anketa:

- Koliko vas je instaliralo Code::Blocks
- i probalo izvršiti neki program?

# Kontrola toka programa

# Sadržaj

- Naredbe — kontrola toka programa:
  - Izrazi i naredbe.
  - Uvjetne naredbe if, if-else, switch.
  - Petlje while, for, do-while.
  - Naredbe break i continue.
  - Naredba goto.

# 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, redosljedom prema **prioritetu** i **asocijativnosti** operacija.

**Primjer.**

---

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

---

**Poziv funkcije** je, također, **izraz** — čak i kad “odbacujemo” **povratnu** vrijednost funkcije (ako funkcija vraća neku vrijednost, poput funkcije **printf**).

# Izrazi i naredbe (nastavak)

Općenito, “radni” dio programa sastoji se od niza naredbi.

- Naredbe završavaju znakom točka–zarez ; .

Svaki izraz iza kojeg slijedi točka–zarez postaje naredba. To je tzv. jednostavna, osnovna ili primitivna naredba.

Primjer.

---

```
x = 3;           // Smije, ali besmisleno:  
++n;           17;  
printf(...);   x+y;
```

---

Osim ovih, postoje još i složene naredbe, te posebne naredbe s “imenom”, za kontrolu redoslijeda izvršavanja ostalih naredbi — tzv. naredbe za kontrolu postupaka ili toka. Idemo redom.

# 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 (ili osnovna) naredba.



# Uvjetno izvršavanje — if naredba

Najjednostavnija `if` naredba ima oblik:

---

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

---

gdje je `uvjet` aritmetički (logički) 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 sljedećom naredbom **iza if** naredbe.

Napomena: `naredba;`, naravno, može biti i **složena** `{...}`.

## if naredba (nastavak)

Ovaj oblik `if` naredbe je:

- **uvjetno** izvršavanje **jedne** (može i složene) naredbe.

Alternative su, ovisno o uvjetu: **izvrši** ili **ne**.

**Pravilo** pisanja:

- “**kontrolni**” izraz **uvjet** piše se u **okruglim** zagradama,
- odmah **iza** **ključne riječi** (ovdje `if`) koja ovdje 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, ili na **kraju** (samo u **do-while**).

## if naredba (nastavak)

Primjer. Odsječak programa

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

radi sljedeće:

- ako (i samo ako) je vrijednost varijable **x** pozitivna, onda ispisuje tu vrijednost,
- a u protivnom — ne radi ništa.

Zatim, povećava (inkrementira) vrijednost od **x** za 1, i to

- neovisno o uvjetu u **if**, tj. neovisno o vrijednosti **x**.

## if naredba (nastavak)

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

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

Paziti na redoslijed pridruživanja — kad koga kamo kopiramo!

- 🕒 Pomoćna varijabla `temp` je prva lijeva strana.
- 🕒 Nadalje, prethodna desna strana je sljedeća lijeva strana.

## if-else naredba

if-else naredba ima oblik:

```
if (uvjet)
    naredba_1;
else
    naredba_2;
```

Ako izraz **uvjet**

- ima vrijednost **istine**, onda se izvršava **naredba\_1**,
- a u **suprotnom** (**laž**) se izvršava **naredba\_2**.

Ovo je:

- uvjetno** izvršavanje **jedne** od **dviju** naredbi.

Alternative su, ovisno o uvjetu: izvrši **jednu** ili **drugú**.

## Kojem `if` pripada `else`?

**Problem.** Kad imamo ugniježdene `if` i `if-else` naredbe, kojem `if` pripada `else` — prvom ili drugom?

**Pravilo.** Svaka `else` naredba pripada **najbližoj** (prethodnoj) `if` naredbi. (Razlog: prevoditelj u danom trenu uvijek “guta” **najdulju** moguću jezičku cjelinu, a `if-else` je **dulji** od `if`.)

**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.

# Izlaz iz programa, funkcija `exit`

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

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

Poziv te funkcije `exit(status);`

- **zaustavlja** izvršavanje programa i vrijednost `status` predaje **operacijskom sustavu**.

To radi **isto** što i `return status;` u funkciji `main`, s tim da se funkcija `exit` može pozvati i u **bilo kojoj** drugoj funkciji.

U `<stdlib.h>` definirane su i dvije **standardne** vrijednosti za status: `EXIT_SUCCESS` i `EXIT_FAILURE`.

- `0` i `EXIT_SUCCESS` znače da je program **uspješno** završio,
- `EXIT_FAILURE` signalizira **neuspješni** završetak programa.



## Funkcija `exit` (nastavak)

Primjer. Podijeli `y` s `x`, reakcija na grešku `x == 0`.

---

```
#include <stdlib.h>

...
int x, y;
...
if (!x) {
    printf("Greska: djelitelj jednak nuli!\n");
    exit(EXIT_FAILURE);
}
else
    y /= x;
```

---

Uočiti: `!x` je istina ako i samo ako je `x == 0`. U ovom primjeru, čitljivija je druga = numerička varijanta.

## *if naredba i uvjetni operator*

Sljedeće dvije naredbe su ekvivalentne:

---

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

---

i

---

```
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**. (Staviti **<=**, umjesto **>=**.)

## Višestruki izbor if-else naredbama

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

**Primjer.** Dvije `if-else` naredbe, druga je **iza else** od 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

```
#include <stdio.h>

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

    printf("Upisati prvi broj: ");
    scanf(" %lf", &a);
    printf("Upisati drugi broj: ");
    scanf(" %lf", &b);
    printf("Upisati operaciju (+, -, *, /): ");
    scanf(" %c", &operacija);
```

## Jednostavni kalkulator — if-else (nastavak)

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

return 0;
}
```

---

# Jednostavni kalkulator — primjer i zadatak

Primjer izvršavanja programa — poruke, ulaz i rezultat:

---

Upisati prvi broj: 21\n

Upisati drugi broj: 13\n

Upisati operaciju (+, -, \*, /): /\n

1.615385

---

Zadatak. Naš program čita jednostavni izraz u tzv.

• postfiks obliku — prvi operand, drugi operand, operacija.

Preuredite ulaz tako da izraz pišemo u uobičajenom

• infiks obliku — prvi operand, operacija, drugi operand,

s tim da izraz smije biti napisan u jednom redu, poput

---

21/13\n

---

## Jednostavni kalkulator — ulaz/izlaz

Par napomena vezanih za **formatirano čitanje** i **pisanje** u ovom primjeru.

Oznaka konverzije **%lf** (slično kao i **%lg**) služi za:

- **čitanje** realnih brojeva tipa **double**.

Za **čitanje** realnih brojeva tipa **float** treba koristiti **%f** (slično kao i **%g**). Oznaka konverzije **%f** (ili **%g**) služi i za:

- **pisanje** realnih brojeva tipa **float** i **double**, s tim da se vrijednost tipa **float** prvo **pretvara** u **double**.

Kod **čitanja**,

- **prvo** se **preskaču** bjeline ispred broja, ako ih ima.

Bjeline su: praznina, tabulatori **\t**, **\v**, i znakovi **\n**, **\r**, **\f**.

## Jednostavni kalkulator — čitanje znaka

Oznaka konverzije `%c` služi za čitanje i pisanje jednog znaka (objekta tipa `char`). Kod čitanja, učitava se

- prvi sljedeći znak na ulazu — bez preskakanja bjelina!

Zato format glasi "`%c`" — s prazninom na početku,

- da se prvo preskoče eventualne bjeline na ulazu!

Pitanje. Zašto to treba napraviti?

Uputa. Što je iza drugog broja na ulazu? (`\n`).

Zadatak. Probajte što se događa ako ispustimo prazninu u formatu za čitanje znaka, tj. format napišemo kao "`%c`".

Usput, vodeća praznina u "`%lf`" nije bitna i može se ispustiti (probajte). Detaljno objašnjenje — kod opisa funkcije `scanf`.



## Višestruki izbor — switch naredba

Naredba `switch` slična je nizu ugniježđenih `if-else` naredbi. Opći oblik te naredbe 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 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 konstanti: **konstanta\_1**, ..., **konstanta\_n**. Ove konstante **moraju** biti međusobno **različite**.

## switch naredba (nastavak)

- Ako je `izraz = konstanta_i`, onda
  - 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 prve **break** naredbe ili do kraja **switch** naredbe.

Naredba **break prekida** “propadanje” kroz **switch** (v. kasnije).

## switch naredba (nastavak)

- Slučaj `default` **ne mora** nužno biti prisutan u `switch` naredbi. Ako **nije** i ako **nema** podudaranja izraza i neke od navedenih 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).

# *Jednostavni kalkulator* switch naredbom

```
#include <stdio.h>

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

    printf("Upisati prvi broj: ");
    scanf(" %lf", &a);
    printf("Upisati drugi broj: ");
    scanf(" %lf", &b);
    printf("Upisati operaciju (+, -, *, /): ");
    scanf(" %c", &operacija);
```

## Jednostavni kalkulator — switch (*nastavak*)

```
switch (operacija) {
    case '+': printf("%f\n", a + b);
              break;
    case '-': printf("%f\n", a - b);
              break;
    case '*': printf("%f\n", a * b);
              break;
    case '/': 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!

```
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), `naredba` se *ponavlja*.

*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* — *primjer*

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

---

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

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

## while *petlja* — *primjer (nastavak)*

```
while (x != 0.0) {
    sum += x;
    printf(" x[%d] = ", ++n);
    scanf("%lf", &x);
}
sum /= n;
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 (n > 0) sum /= n;`

# 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;  
}
```

(*osim* ponašanja *continue* naredbe unutar petlje, v. kasnije).

*Srednji* izraz *izraz\_2* interpretira se kao *logički* izraz, a ostala *dva* izraza mogu biti *bilo što* (pretvaraju se u naredbe).

## for *petlja* (*nastavak*)

`for` petlja najčešće se koristi za ponavljanje pod kontrolom nekog “brojača”.

Kad imamo “brojač” za kontrolu ponavljanja u `for` petlji

---

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

---

standardna značenja pojedinih izraza su:

- 🔴 `izraz_1` — inicijalizacija brojača na početku petlje,
- 🔴 `izraz_2` — provjera završne vrijednosti brojača za “nastavak” ponavljanja u petlji,
- 🔴 `izraz_3` — “pomak” brojača na kraju svakog prolaza kroz petlju.

## for *petlja (nastavak)*

**Primjer.** Standardni “pomak” brojača za 1 “**unaprijed**”:

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

**Primjer.** Brojač možemo “pomicati” i drugačije.

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

Ovdje brojač **povećavamo** za 2.

Pojedini izrazi u **for** petlji smiju biti “**prazni**”.

**Beskonačna** petlja koja ne radi ništa (default **izraz\_2 = 1**):

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

## for *petlja* (nastavak)

**Primjer.** Uočite razliku koju radi `;` na kraju retka s `for` naredbom. Prvo **bez** točka-zareza.

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

Ovo će ispisati redom: `brojac = 1` do `brojac = 4`.  
Razlog: `printf` je naredba **unutar** petlje.

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

Ovo će ispisati samo: `brojac = 5`.  
Razlog: `printf` je **izvan**, **iza** petlje. Petlja “vrti” **praznu** naredbu “`;`”!

# 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 izraza
- računa i provjerava na “*dnu*” (*kraju*) prolaza kroz petlju, *iza* naredbe.

## do-while *petlja (nastavak)*

Zato se naredba u `do-while` petlji izvršava **barem jednom**,

• **prije** prve provjere izraza.

**Primjer.** Dio programa koji ispisuje brojeve 0, 1, ..., 9.

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

Ovo **nije** pravi primjer za korištenje `do-while` petlje.

Puno **bolji** primjer je obrada **niza** podataka, u kojem **barem jedan** podatak treba obraditi, pa makar to bio i jedini.



# Naredba `break`

Naredba `break` služi za:

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

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

Pri nailasku na naredbu `break`,

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

## Naredba break (*nastavak*)

Primjer. Obrada **niza** brojeva s “oznakom” za **kraj** niza.

---

```
int i;
while (1) {
    scanf("%d", &i);      /* citanje broja. */
    if (i < 0) break;    /* test kraja. */
    ...                  /* obrada broja. */
}
```

---

Petlja `while (1)` je **beskonačna** petlja.

- 🔴 Služi za **čitanje** i **obradu** pojedinih brojeva.
- 🔴 Iz nje se **izlazi** ako se učitava **negativan** broj (**kraj** niza).

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

## Naredba `continue`

Naredba `continue` može se koristiti unutar `for`, `while` i `do-while` petlji za “skraćenje” pojedinog prolaza kroz petlju, preskakanjem preostalih 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`,
  - pomak brojača (`izraz_3`) u `for`.

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

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

## Naredba continue (*nastavak*)

**Primjer.** Po ugledu na prethodni primjer obrade **niza** brojeva, kôd koji **preskače negativne** vrijednosti (i **ne obrađuje ih**) može se napraviti naredbom **continue**:

```
int i;
while (1) {
    scanf("%d", &i);        /* citanje broja. */
    if (i < 0) continue;   /* preskakanje neg. */
    ...                    /* obrada neneg. */
}
```

U dijelu kôda koji **obrađuje nenegativne** brojeve, trebamo neki drugi način **izlaza** iz petlje, tj. neku drugu “oznaku” za **kraj niza**. Ako je to **nula**, što treba “**dodati**” u ovaj dio programa?

# Naredba goto

Naredba `goto` prekida sekvencijalno izvršavanje programa i

- nastavlja izvršavanje s naredbom koja je označena labelom navedenom u toj `goto` naredbi (tzv. “skok”).

Pravilo pisanja `goto` naredbe 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.

---

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

---

## 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: ;    // prazna naredba na dnu petlje  
}
```

Kad ovo “prebacimo” u pripadnu `while` petlju, dobijemo **korektno** ponašanje — sljedeća operacija je pomak brojača!

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

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



# Naredba goto — savjet

Napomena. Program koji ima puno goto naredbi

- bitno je teže pročitati i razumjeti, od programa koji ne koristi goto.

Zato savjet: upotrebu goto naredbe treba izbjegavati.

Gruba uputa: goto naredba služi samo za reakciju na

- specijalne slučajeve i greške,
- kad “velike” dijelove programa treba preskočiti, i to skokom unaprijed,

zato da se izbjegne dodatno “uvlačanje” dijelova kôda if-else naredbama, koje bi smanjilo preglednost!

“Strogo zabranjeno”: skokovi unatrag = petlje putem goto!