

# **Programiranje i programsko inženjerstvo**

Predavanja  
2014. / 2015.

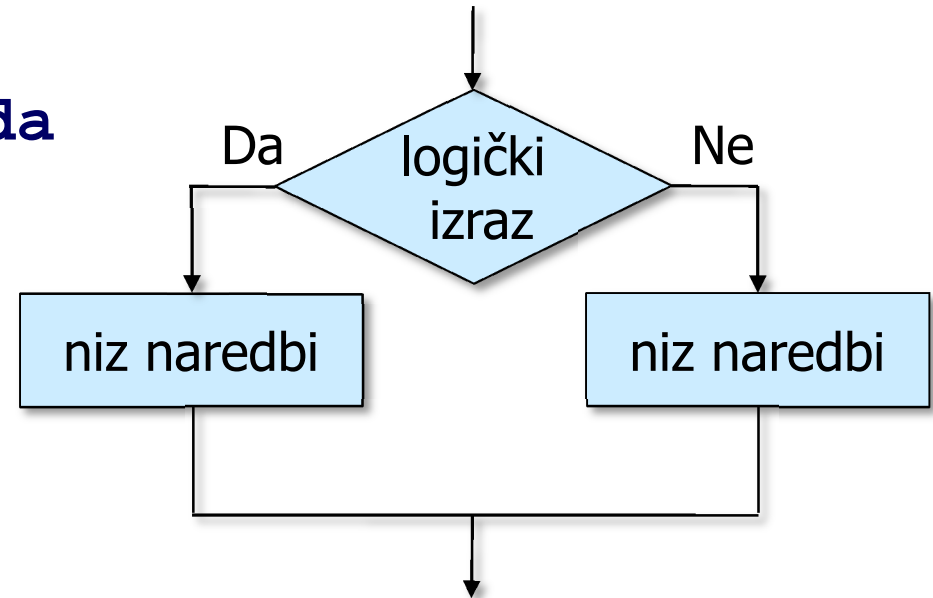
## **5. Kontrolne naredbe**

# Kontrolna naredba `if`

## - dvostrana selekcija - ponavljanje

### ■ Pseudokôd

```
ako je logički_izraz tada  
| niz_naredbi_1  
inače  
| niz_naredbi_2
```



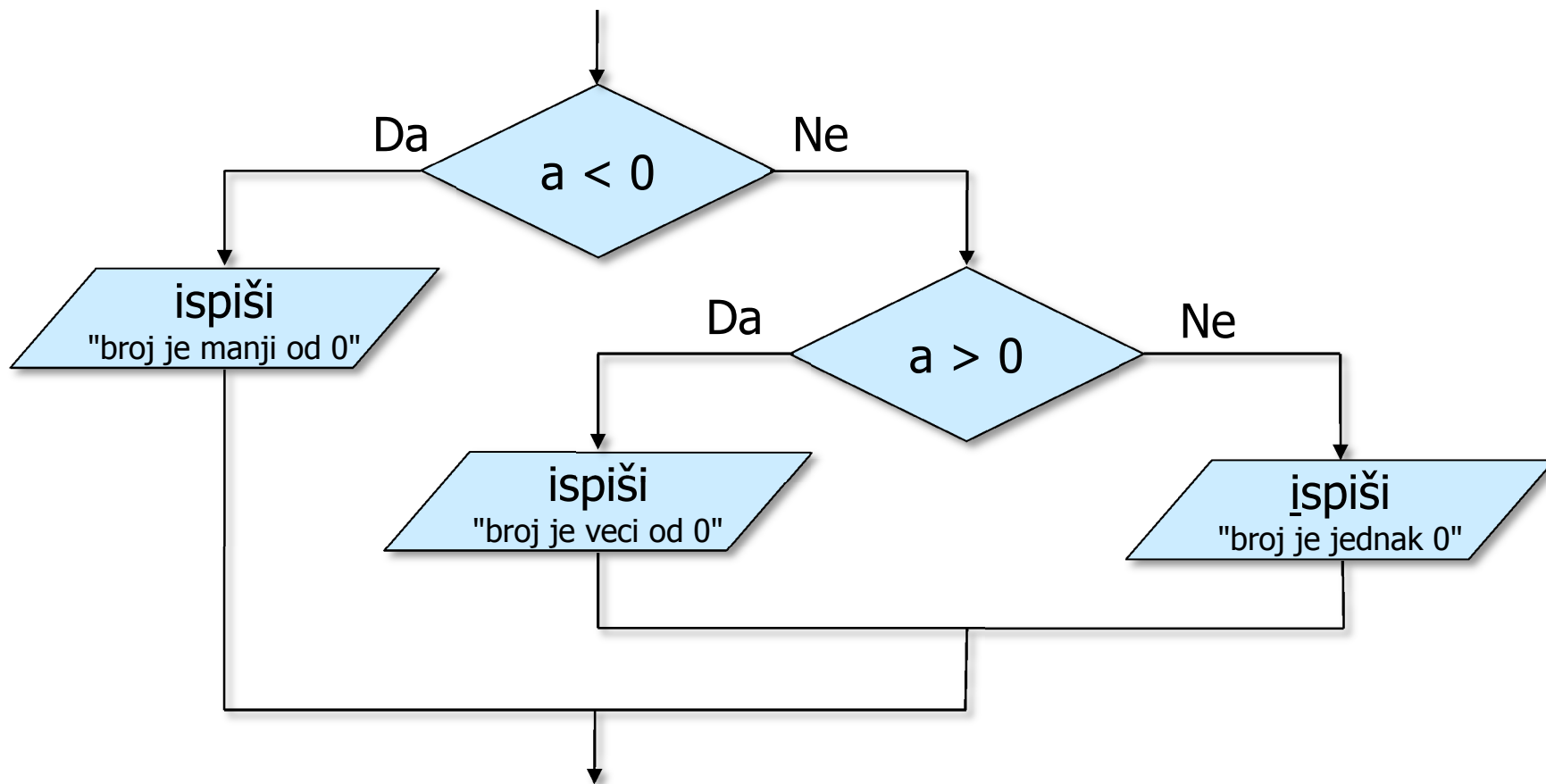
### ■ U C-u

```
if (logički_izraz) {  
    niz_naredbi_1  
} else {  
    niz_naredbi_2  
}
```

- ako niz\_naredbi obuhvaća samo jednu naredbu, pripadne vitičaste zagrade smiju se ispustiti

# Primjer za dvostranu selekciju - ponavljanje

- Zadatak: učitati cijeli broj. Ovisno o učitanoj broju, ispisati "broj je veci od 0", "broj je jednak 0" ili "broj je manji od 0".



# Rješenje primjera za dvostranu selekciju - ponavljanje

---

```
#include <stdio.h>
int main(void) {
    int a;
    printf("Upisite cijeli broj:");
    scanf("%d", &a);
    if (a < 0) {
        printf("Broj je manji od 0\n");
    } else
        if (a > 0) {
            printf("Broj je veci od 0\n");
        } else {
            printf("Broj je jednak 0\n");
        }
    return 0;
}
```

# Višestrana selekcija s pomoću naredbi `if - else if - else`

---

```
if (logički_izraz_1) {  
    niz_naredbi_1  
}  
else if (logički_izraz_2) {  
    niz_naredbi_2  
}  
else if (logički_izraz_3) {  
    niz_naredbi_3  
    ...  
}  
else {  
    niz_naredbi_0  
}
```

dio **else** višestrane selekcije  
može se (eventualno) ispustiti

# Rješenje primjera za dvostranu selekciju napisano malo drugačije

---

```
#include <stdio.h>
int main(void) {
    int a;
    printf("Upisite cijeli broj:");
    scanf("%d", &a);
    if (a < 0) {
        printf("Broj je manji od 0\n");
    } else if (a > 0) {
        printf("Broj je veci od 0\n");
    } else {
        printf("Broj je jednak 0\n");
    }

    return 0;
}
```

# Rješenje primjera za dvostranu selekciju napisano još malo drugačije

---

U konkretnom primjeru vitičaste zagrade nisu bile nužne:

```
#include <stdio.h>
int main(void) {
    int a;
    printf("Upisite cijeli broj:");
    scanf("%d", &a);
    if (a < 0)
        printf("Broj je manji od 0\n");
    else if (a > 0)
        printf("Broj je veci od 0\n");
    else
        printf("Broj je jednak 0\n");
    return 0;
}
```

# Primjer za višestranu selekciju

---

- Učitati dva cijela broja i jedan znak. Ako je učitani znak '+' ispisati zbroj učitanih brojeva, a ako je učitani znak '\*' ispisati njihov umnožak. Ako je učitani neki treći znak, ispisati poruku **Neispravna operacija**.

```
pročitaj (m, n, znak)
ako je znak = '+' tada
    ispiši(m + n)
inače ako je znak = '*' tada
    ispiši(m * n)
inače
    ispiši("Neispravna operacija")
kraj
```



# Rješenje

---

```
#include <stdio.h>
int main(void) {
    int m, n;
    char znak;
    scanf("%d %d %c", &m, &n, &znak);
    if ( znak == '+' ) {
        printf("%d\n", m + n);
    } else if ( znak == '*' ) {
        printf("%d\n", m * n);
    } else {
        printf("Neispravna operacija\n");
    }
    return 0;
}
```

# Primjer za višestranu selekciju

---

- Kotao ima radnu temperaturu u intervalu  $[50^{\circ}\text{C}, 80^{\circ}\text{C}]$ . Napisati program koji provjerava je li s tipkovnice učitana vrijednost temperature kotla unutar dopuštenog intervala te na zaslon ispisuje jednu od sljedećih poruka:

Temperatura kotla je ispod dopustene!

Temperatura kotla je iznad dopustene!

Temperatura kotla je OK.

# Rješenje

---

```
#include <stdio.h>
#define DONJA 50.0f
#define GORNJA 80.0f
int main(void) {
    float temp;
    printf("\n Unesite temperaturu kotla: ");
    scanf("%f",&temp);
    if (temp < DONJA) {
        printf("Temperatura kotla je ispod dopustene!\n");
    } else if (temp > GORNJA) {
        printf("Temperatura kotla je iznad dopustene!\n");
    } else {
        printf("Temperatura kotla je OK.\n");
    }
    return 0;
}
```

# Moguće dileme u korištenju `else` dijela naredbe `if`

---

Pogrešno "uvučeno"

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

Ispravno "uvučeno"

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

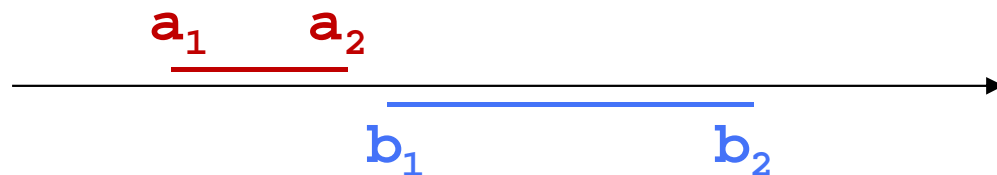
Ispravno "uvučeno", ali to nije isti odsječak kao gore

```
if(uvjet1) {
    if (uvjet2)
        naredba2;
}
else
    naredba3;
```

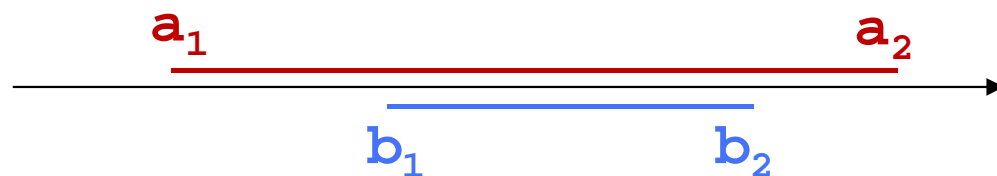
**Pravilo:** *else* pripada "najbližem" *if-u* koji "nema svoj *else*"

# Primjer

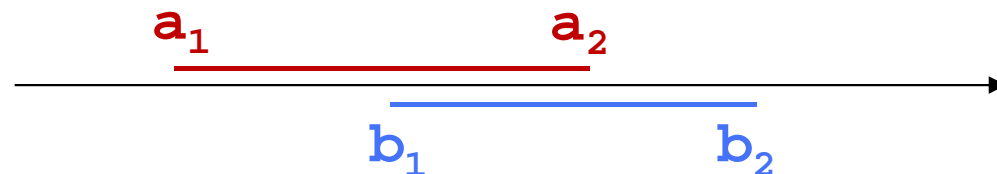
- Učitati granice zatvorenih intervala realnih brojeva  $[a_1, a_2]$  i  $[b_1, b_2]$ , pri čemu  $a_1$  mora biti manji od  $b_1$ . Nije potrebno provjeravati ispravnost unesenih granica intervala. Ako je presjek intervala neprazan skup, ispisati granice intervala  $[r_1, r_2] = [a_1, a_2] \cap [b_1, b_2]$ .
- Ako je  $a_1 < b_1$ , tada su mogući samo prikazani odnosi među intervalima:



$a_2 < b_1$ :  
intervali se ne sijeku



$a_2 > b_2$ :  
 $r_1 = b_1, r_2 = b_2$



$a_2 \leq b_2$ :  
 $r_1 = b_1, r_2 = a_2$

# Rješenje

```
#include <stdio.h>
int main(void) {
    float a1, a2, b1, b2;
    float r1, r2;
    scanf("%f %f %f %f", &a1, &a2, &b1, &b2);
    if (a2 >= b1) {          /* presjek intervala nije prazan */
        r1 = b1;
        if( a2 > b2 )
            r2 = b2;
        else
            r2 = a2;
        printf("r1=%f, r2=%f", r1, r2);
    }
    return 0;
}
```

- Obratiti pažnju na to da `else` pripada drugoj (unutarnjoj) naredbi `if`, a ne prvoj (vanjskoj).

Kontrolne naredbe  
- programske petlje -

# Programske petlje

---

- Programske petlje služe za obavljanje određenog programskog odsječka (tijelo petlje) više puta.
  
- Dijelimo ih na
  - programske petlje s ispitivanjem uvjeta na početku
    - ovisno o početnim uvjetima može se dogoditi da se tijelo petlje uopće neće izvršiti
  - programske petlje s ispitivanjem uvjeta na kraju
    - tijelo petlje će se izvršiti barem jednom
  - programske petlje s poznatim brojem ponavljanja
    - broj ponavljanja može se unaprijed izračunati i ne ovisi o izvršavanju tijela petlje



# Programska petlja s ispitivanjem uvjeta ponavljanja na početku

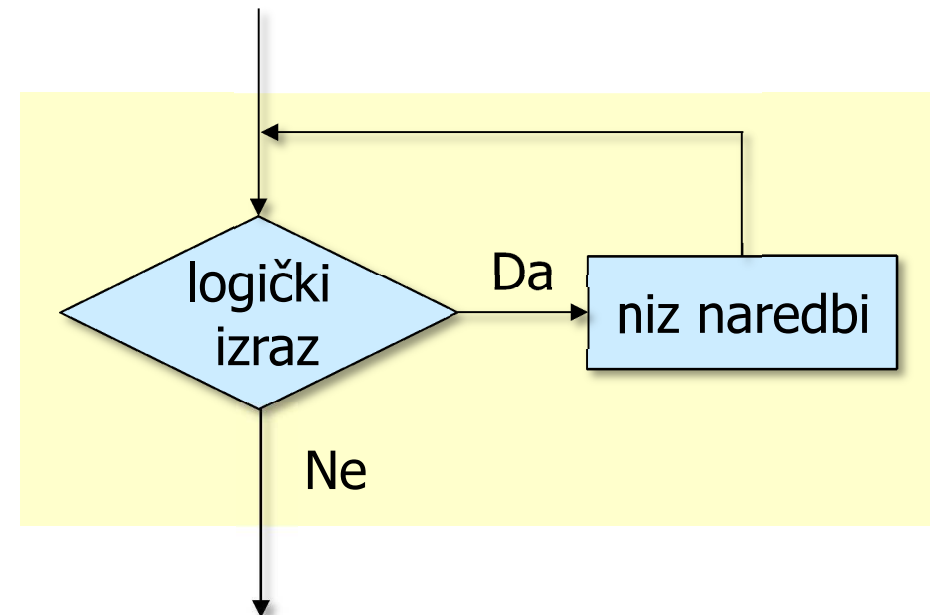
## ■ Pseudokod

```
dok je (logički_izraz)
    | niz_naredbi
```

## ■ C

```
while (logički_izraz) naredba
ili
while (logički_izraz) {
    niz_naredbi
}
```

## ■ Dijagram toka



# Primjer

- Učitati nenegativni cijeli broj. Nije potrebno provjeravati ispravnost unesenog broja. Ispisivati ostatke uzastopnog dijeljenja učitano broj s 2, a postupak prekinuti kad se dijeljenjem dođe do 0.
  - učitani broj može biti 0. Može se dogoditi da se neće ispisati niti jedan ostatak dijeljenja, tj. da se tijelo petlje neće izvršiti niti jednom.

```
Upisite nenegativan cijeli broj: 11
Upisali ste: 11
Ostatak je: 1
Ostatak je: 1
Ostatak je: 0
Ostatak je: 1
```

```
Upisite nenegativan cijeli broj: 0
Upisali ste: 0
```

# Rješenje

---

```
#include <stdio.h>
int main(void) {
    int broj, ostatak;
    printf("Upisite nenegativan cijeli broj: ");
    scanf("%d", &broj);
    printf("Upisali ste: %d\n", broj);
    while (broj != 0) {
        ostatak = broj % 2;
        printf("Ostatak je: %d\n", ostatak);
        broj = broj / 2;
    }
    return 0;
}
```

# Primjer

---

- Učitavati i sumirati cijele brojeve dok se ne upiše cijeli broj 0. Ispisati sumu učitanih brojeva.
- Trebat će barem jednom učitati cijeli broj. Petlja s ispitivanjem uvjeta na početku nije naročito prikladna za rješavanje ovog zadatka.

```
#include <stdio.h>
int main(void) {
    int broj, suma = 0;
    printf("Upisite broj: ");
    scanf("%d", &broj);
    while (broj != 0) {
        suma = suma + broj;
        printf("Upisite broj: ");
        scanf("%d", &broj);
    }
    printf ("Suma=%d.\n", suma);
    return 0;
}
```

# Programska petlja s ispitivanjem uvjeta ponavljanja na kraju

---

- Pseudokod
- U nekim programskim jezicima (npr. Fortran, Pascal) postoji oblik petlje *repeat ... until*:

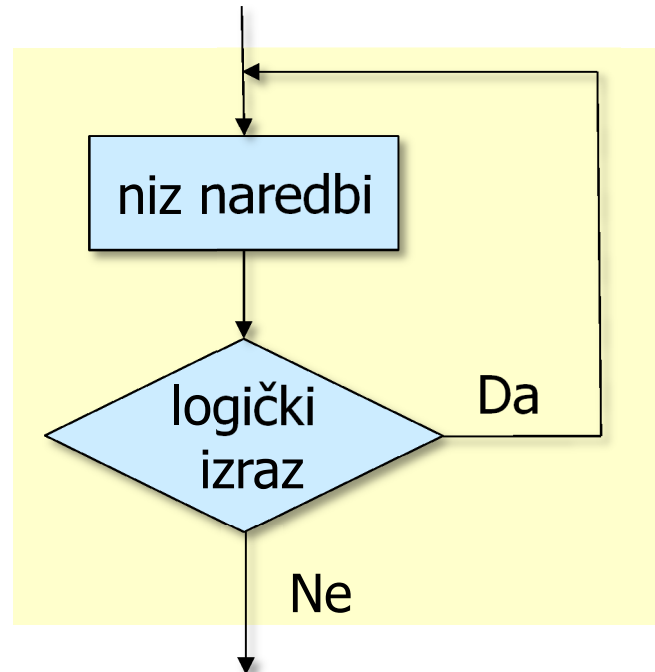
```
ponavlja  
  | niz_naredbi  
dok ne bude (logički_izraz)
```

- U programskom jeziku C postoji oblik petlje *do ... while*:

```
ponavlja  
  | niz_naredbi  
dok je (logički_izraz)
```

# Programska petlja s ispitivanjem uvjeta ponavljanja na kraju

- Dijagram toka



- C

`do naredba while (logički_izraz);`

`ili`

`do {`

`niz_naredbi`

`} while (logički_izraz);`

# Primjer

---

- Učitavati i sumirati cijele brojeve dok se ne upiše cijeli broj 0. Ispisati sumu učitanih brojeva.

```
#include <stdio.h>
int main(void) {
    int broj, suma = 0;
    do {
        printf("Upisite broj: ");
        scanf("%d", &broj);
        suma = suma + broj;
    } while (broj != 0);
    printf("Suma=%d.\n", suma);
    return 0;
}
```

# Primjer

---

- Učitati pozitivni cijeli broj koji određuje gornju granicu sume brojeva (ne treba provjeravati ispravnost učitano broj). Zatim učitavati i sumirati cijele brojeve sve dok njihova suma ne prekorači zadanu gornju granicu sume. Nakon toga izračunati i ispisati aritmetičku sredinu učitanih brojeva.



# Rješenje

- Pseudokod

```
učitaj (gg)
```

```
brojac := 0
```

```
suma := 0
```

```
ponavljaj
```

```
    učitaj (broj)
```

```
    suma := suma + broj
```

```
    brojac := brojac + 1
```

```
dok ne bude suma > gg
```

```
as := suma / brojac
```

```
ispiši( suma, brojac, as )
```

Mora se promijeniti u:

dok je suma <= gg

# Rješenje zadatka u C-u

```
#include <stdio.h>
int main(void) {
    int brojac = 0, suma = 0, broj, gg;
    float as;
    printf("Upisite gornju granicu: ");
    scanf("%d", &gg);

    do {
        scanf("%d", &broj);
        suma = suma + broj;
        brojac = brojac + 1;
    } while (suma <= gg);

    as = (float) suma / brojac;
    printf("%d %d %f\n", suma, brojac, as);
    return 0;
}
```

naredbe iz tijela petlje obaviti će se barem jednom jer se koristi petlja s ispitivanjem uvjeta ponavljanja na kraju

# Rješenje istog zadatka

(s pomoću petlje s ispitivanjem uvjeta ponavljanja na početku)

```
#include <stdio.h>
int main(void) {
    int brojac = 0, suma = 0, broj, gg;
    float as;
    printf("Upisite gornju granicu: ");
    scanf("%d", &gg);

    while (suma <= gg) {
        scanf("%d", &broj);
        suma = suma + broj;
        brojac = brojac + 1;
    }

    as = (float) suma / brojac;
    printf("%d %d %f\n", suma, brojac, as);
    return 0;
}
```

naredbe iz tijela petlje obaviti će se barem jednom jer je u ovom slučaju na početku sigurno zadovoljen uvjet za obavljanje tijela petlje

# Primjer

---

- Učitavati cijele brojeve iz intervala  $[-100, 100]$ . Učitavanje brojeva prekinuti kada se učitava broj izvan intervala  $[-100, 100]$ . Ispisati broj učitanih pozitivnih brojeva, broj učitanih negativnih brojeva i broj učitanih nula. U obzir uzeti samo brojeve iz intervala  $[-100, 100]$ .

# Rješenje

---

```
#include <stdio.h>
int main(void) {
    int broj;
    int brojPozitivnih = 0, brojNegativnih = 0, brojNula = 0;
    printf("Unesite brojeve: ");
    do {
        scanf("%d", &broj);
        if (broj >= -100 && broj <= 100) {
            if (broj == 0) ++brojNula;
            else if (broj > 0) ++brojPozitivnih;
            else ++brojNegativnih;
        }
    } while (broj >= -100 && broj <= 100);

    printf("Pozitivnih je %d, negativnih je %d, nula je %d\n"
           , brojPozitivnih, brojNegativnih, brojNula);
    return 0;
}
```

# Primjer

---

- Učitavati cijele brojeve (napomena: učitani brojevi mogu biti i negativni). Učitavanje prekinuti kada se unese broj nula. Ako je unesen barem jedan pozitivan broj, odrediti i ispisati najveći od unesenih pozitivnih brojeva. Ako nije unesen niti jedan pozitivan broj, ispisati poruku: `Nije unesen niti jedan pozitivan broj`.

# Rješenje

- Kod traženja najvećeg (slično i kod traženja najmanjeg) člana niza koristimo sljedeći algoritam:
  - prvi član niza proglasimo najvećim i njegovu vrijednost pohranimo u pomoćnu varijablu koja predstavlja trenutni maksimum
  - redom ispitujuemo preostale članove niza i ako je neki od njih veći od trenutnog maksimuma, ažuriramo trenutni maksimum na tu vrijednost
- Nakon što smo ispitali sve članove niza, u pomoćnoj varijabli se nalazi maksimalni element niza.
- Primjer: niz  $\rightarrow$  5, 6, 3, 7, 4, 7, 9, 5.

5	6	3	7	4	7	9	5
maks=5	6>5 ?	3>6?	7>6?	4>7?	7>7?	9>7?	5>9?
	maks=6		maks=7			maks=9	

# Rješenje

- U ovom zadatku možemo kao pretpostavljenu maksimalnu vrijednost postaviti vrijednost nula.
- Primjer: niz  $\rightarrow$  5, 6, -3, 7, 4, 9, 5, 0.

	5	6	-3	7	4	9	5	0
maks=0	5>0 ?	6>5 ?	zanemari	7>6?	4>7?	9>7?	5>9?	prekid učitavanja
	maks=5	maks=6		maks=7		maks=9		

- Primjer: niz  $\rightarrow$  -2, -7, -9, 0.

	-2	-7	-9	0
maks=0	zanemari	zanemari	zanemari	prekid učitavanja



# Rješenje zadatka u C-u

---

```
...
int maks = 0;
int broj;
do {
    printf("Unesite broj : ");
    scanf("%d", &broj);
    if (broj > maks)
        maks = broj;
} while (broj != 0);
if (maks > 0)
    printf("Najveci uneseni broj je %d\n", maks);
else
    printf("Nije unesen niti jedan pozitivan broj\n");
...
```

# Programska petlja s poznatim brojem ponavljanja

---

- Pseudokod

```
za i = pocetak do kraj (korak k)
    niz_naredbi
```

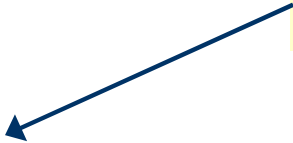
- C

```
for (i = pocetak; i <= kraj; i = i + k) naredba;
    ili
for (i = pocetak; i <= kraj; i = i + k) {
    niz_naredbi
}
```

# Primjer

- Ispisati neparne brojeve iz intervala [1, 10], redom od najmanjeg prema najvećem, svaki broj u svom retku.

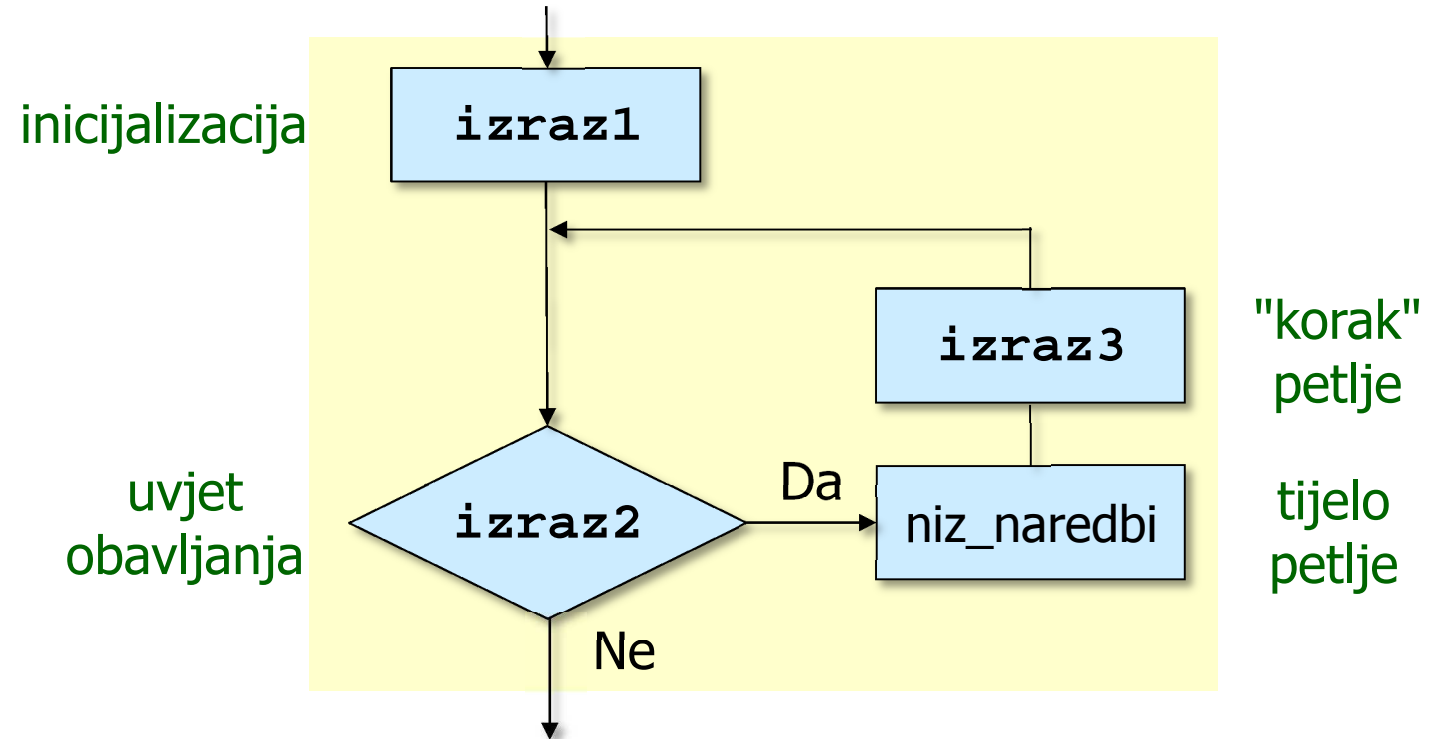
```
#include <stdio.h>
int main(void) {
    int brojac;
    for (brojac = 1; brojac <= 10; brojac = brojac + 2) {
        printf("%d\n", brojac);
    }
    return 0;
}
```



Može se staviti 9

# Programska petlja s poznatim brojem ponavljanja

- Dijagram toka



- C

```
for (izraz1; izraz2; izraz3) naredba;
```

ili

```
for (izraz1; izraz2; izraz3) {  
    niz_naredbi  
}
```

# Programska petlja s poznatim brojem ponavljanja

---

```
for (izraz1; izraz2; izraz3) {  
    ...  
}
```

- **izraz1** je izraz koji će se izvršiti samo jednom, prije ulaska u prvu iteraciju. Najčešće se koristi za inicijalizaciju brojača.
- **izraz2** se izračunava kao logički izraz (0 - false, !=0 - true), te se tijelo petlje izvršava ako je **izraz2** zadovoljen (istinit). Ako nije, petlja se prekida (nastavlja se s prvom naredbom iza petlje).
- **izraz3** se obavlja nakon svakog prolaska kroz tijelo petlje. Najčešće se koristi za povećavanje/smanjenje brojača. Nakon obavljanja izraza **izraz3**, ponovo se testira uvjet u **izraz2**.
- Bilo koji od izraza (**izraz1**, **izraz2**, **izraz3**) se može izostaviti. Ako je izostavljen **izraz2**, petlja se izvodi kao da je logička vrijednost za **izraz2** istinita (true).

# Odvajanje naredbi zarezom u for petlji

- Zarez se kao operator koristi za odvajanje naredbi (izraza) obično tamo gdje je dopuštena samo jedna naredba (izraz). Ovaj se operator najčešće koristi upravo u petljama s poznatim brojem ponavljanja. Na primjer:

```
for ( i = 0, j = 60; i < j; ++i, --j ) {  
    . . .  
}
```

inicijalizacija dvaju brojača

promjena vrijednosti dvaju  
brojača

# Primjeri petlji s poznatim brojem ponavljanja

---

- Primjer `for` petlje koja uzlazno mijenja kontrolnu varijablu:

```
for (i = poc; i <= kraj; i = i + korak) {  
    ...  
}
```

- Primjer `for` petlje koja silazno mijenja kontrolnu varijablu :

```
for (cv = 10; cv > 0; cv = cv - 1) {  
    ...  
}
```

- Primjer `for` petlje s dva brojača (jedan se smanjuje, drugi povećava):

```
for (si = 10, uz = 0; si >= uz; si = si-1, uz = uz+1) {  
    ...  
}
```

# Primjer

---

- Učitati pozitivan cijeli broj  $n$  (nije potrebno provjeravati je li učitani ispravan broj). Zatim učitati  $n$  cijelih brojeva, izračunati i na zaslon ispisati njihovu aritmetičku sredinu.
- Koja je vrsta petlje najprikladnija za rješavanje ovog zadatka?



# Rješenje

---

```
#include <stdio.h>
int main(void) {
    int i, n, suma = 0, x;
    float arit_sred;
    printf("Za koliko brojeva zelite izracunati"
           " aritmeticku sredinu : ");
    scanf("%d", &n);
    for (i = 1; i <= n; i = i + 1) {
        printf("Unesite %d. broj : ", i);
        scanf("%d", &x);
        suma = suma + x;
    }
    arit_sred = (float) suma / n;
    printf("Aritmeticka sredina ucitanih brojeva"
           " je %f\n", arit_sred);
    return 0;
}
```

# Primjer

---

- Učitati pozitivan cijeli broj  $n$  (nije potrebno provjeravati je li učitani ispravan broj). Zatim ispisati realne brojeve od 0 do  $n$  s korakom od 0.1. Za ispis realnih brojeva koristiti formatsku specifikaciju `%12.9f`.

```
#include <stdio.h>
int main(void) {
    int n;
    float x;
    printf("Do kojeg broja zelite ispis: ");
    scanf("%d", &n );
    for ( x = 0.f; x <= n; x = x + 0.1f ) {
        printf("%12.9f\n", x);
    }
    return 0;
}
```

# Rezultat izvođenja programa

Do kojeg broja zelite ispis: 2

0.000000000

0.100000001

0.200000003

0.300000012

0.400000006

0.500000000

0.600000024

0.700000048

0.800000072

0.900000095

1.000000119

1.100000143

1.200000167

1.300000191

1.400000215

1.500000238

1.600000262

1.700000286

1.800000310

1.900000334

Prikazano rješenje očito nije ispravno. Kako ga poboljšati?

# Primjer

- Učitati pozitivan cijeli broj  $n$  (nije potrebno provjeravati je li učitani ispravan broj). Zatim od većih prema manjim, ispisati sve prirodne brojeve djeljive sa 7, 13 ili 19, koji su manji od broja  $n$ .

```
#include <stdio.h>
int main(void) {
    int i, n;
    printf("Upisite broj n: ");
    scanf("%d", &n );
    for ( i = n - 1; i > 0; i = i -1 ) {
        if (( i % 7 == 0 ) ||
            ( i % 13 == 0 ) ||
            ( i % 19 == 0 )) printf("%d\n", i);
    }
    return 0;
}
```

# Česte pogreške

Programski kod	Rezultat
<pre>for (i=1; i=10; i=i+1) {     printf("%d\n",i); }</pre>	neprekidno ispisuje broj 10 (beskonačna petlja)
<pre>for (i=1; i==10; i=i+1) {     printf("%d\n",i); }</pre>	neće ništa ispisati (naredba u tijelu petlje se neće niti jednom obaviti)
<pre>for (i=1; i&lt;=10; i=i+1); {     printf("%d\n",i); }</pre>	jednom ispisuje broj 11

# Primjer

- Ispisati tablicu potencija  $2^n$  i  $2^{-n}$  za brojeve od 0 do 16

```
#include <stdio.h>
int main(void) {
    int brojac;
    int x = 1;
    double y = 1.0;
    printf ( "%2d %12d %18.16f\n", 0, x, y);
    for (brojac = 1; brojac <= 16; ++brojac) {
        x = x * 2;
        y = y / 2;
        printf ( "%2d %12d %18.16f\n", brojac, x, y);
    }
    return 0;
}
```

# Rezultat izvođenja programa

0	1	1.000000000000000000
1	2	0.500000000000000000
2	4	0.250000000000000000
3	8	0.125000000000000000
4	16	0.062500000000000000
5	32	0.031250000000000000
6	64	0.015625000000000000
7	128	0.007812500000000000
8	256	0.003906250000000000
9	512	0.001953125000000000
10	1024	0.000976562500000000
11	2048	0.000488281250000000
12	4096	0.000244140625000000
13	8192	0.000122070312500000
14	16384	0.000061035156250000
15	32768	0.000030517578125000
16	65536	0.000015258789062500

# Primjer

- Uzastopno učitavati nenegativne cijele brojeve. Učitavanje prekinuti kada se učitava broj 0. Svaki učitani broj ispisati u obliku binarnog broja. Za izdvajanje pojedinačnih bitova koristiti operatore  $\gg$  i  $\&$
- Ideja:

```
1010...01011001 >> 31 & 0x1 → 0000...00000001
1010...01011001 >> 30 & 0x1 → 0000...00000000
1010...01011001 >> 29 & 0x1 → 0000...00000001
...
1010...01011001 >> 2 & 0x1 → 0000...00000000
1010...01011001 >> 1 & 0x1 → 0000...00000000
1010...01011001 >> 0 & 0x1 → 0000...00000001
```



# Rješenje

---

...

```
unsigned int broj;
```

```
int i;
```

```
do {
```

```
    printf("Upisite nenegativni cijeli broj: ");
```

```
    scanf ("%u", &broj);
```

```
    for (i = 31; i >= 0; i = i - 1) {
```

```
        printf("%d", broj >> i & 0x1);
```

```
    }
```

```
    printf ("\n");
```

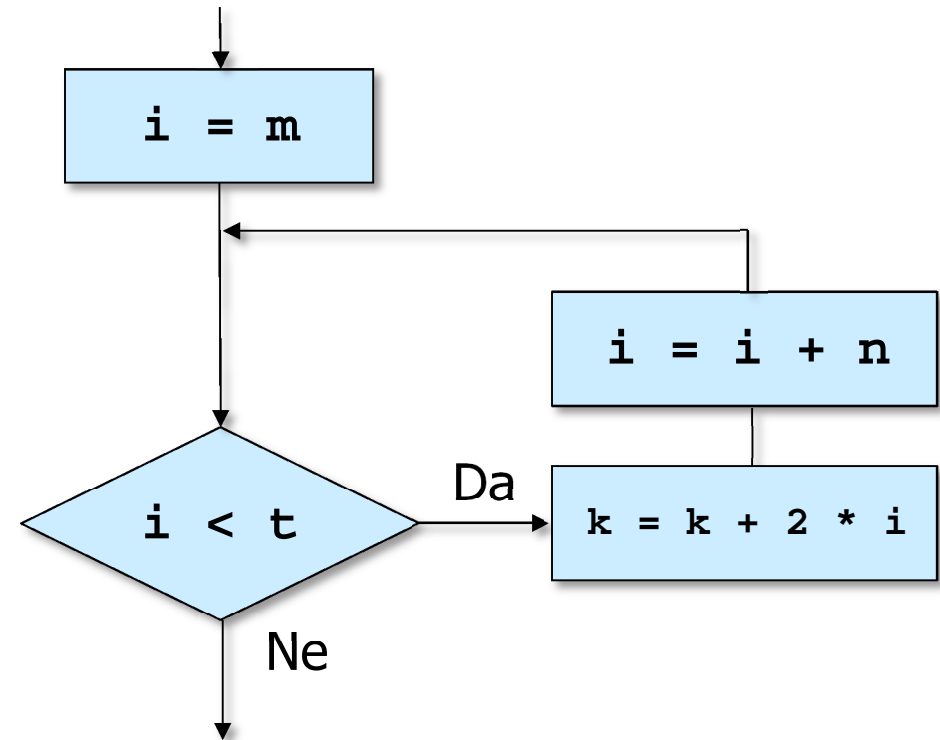
```
} while (broj != 0);
```

...

# Primjer realizacije istog algoritma raznim vrstama programskih petlji (1)

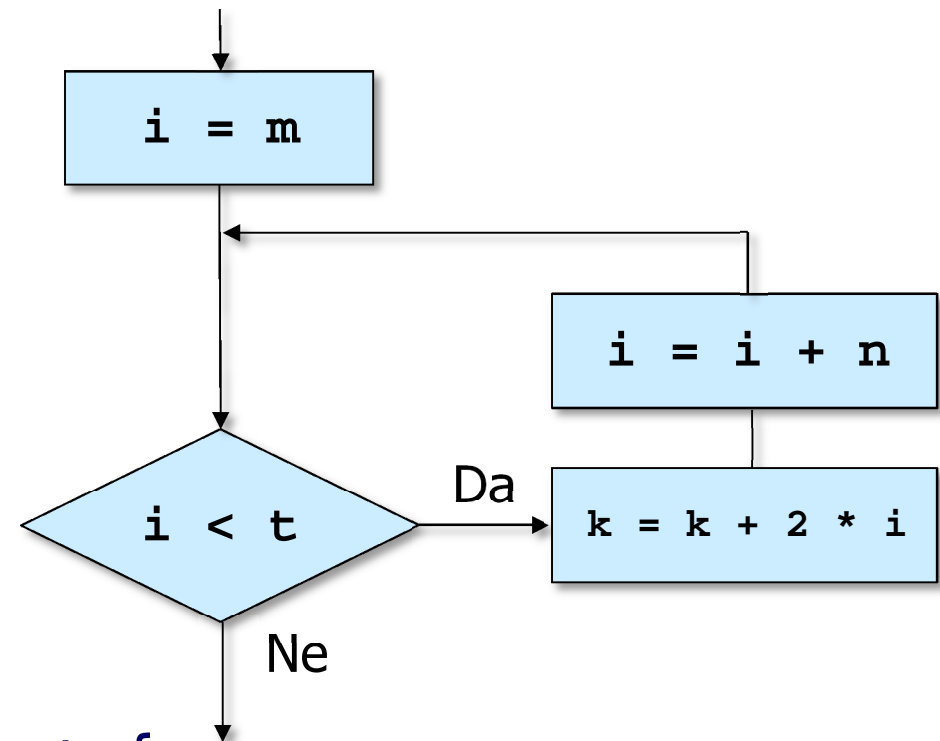
- Programski odsječak prikazan dijagramom toka treba realizirati petljom s ispitivanjem uvjeta ponavljanja na početku

```
i = m;  
while (i < t) {  
    k = k + 2 * i;  
    i = i + n;  
}
```



# Primjer realizacije istog algoritma raznim vrstama programskih petlji (2)

- Programski odsječak prikazan dijagramom toka treba realizirati petljom s poznatim brojem ponavljanja



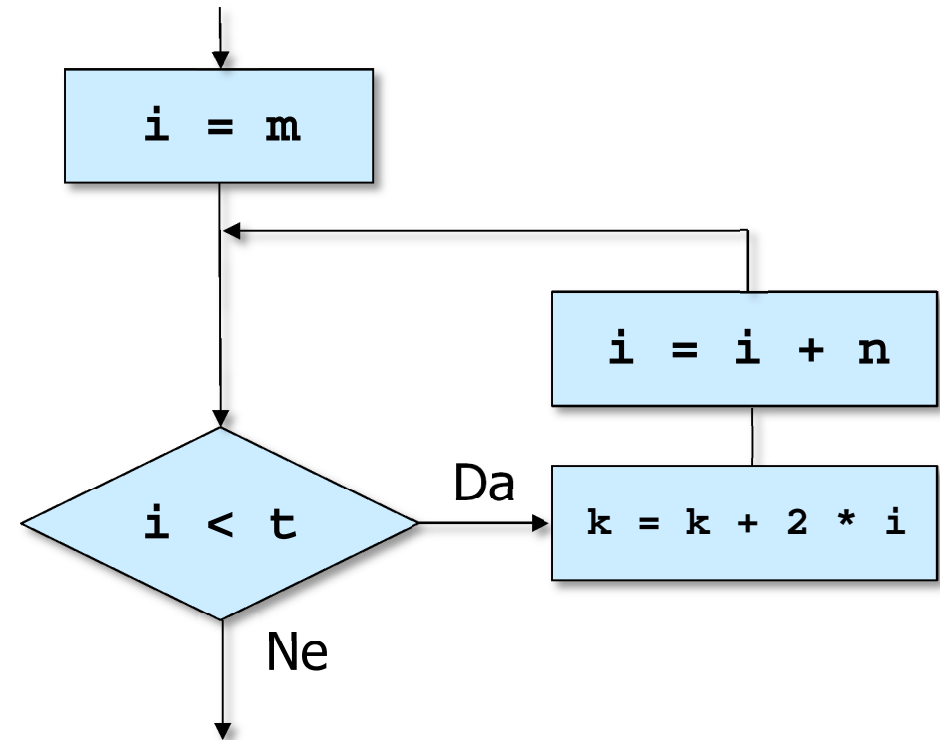
```
for (i = m; i < t; i = i + n) {  
    k = k + 2 * i;  
}
```

Prednost `for` petlje u ovom slučaju je u tome što se početna inicijalizacija brojača, ispitivanje uvjeta i korak brojača nalaze na jednom mjestu u kodu.

# Primjer realizacije istog algoritma raznim vrstama programskih petlji (3)

- Programski odsječak prikazan dijagramom toka treba realizirati petljom s ispitivanjem uvjeta ponavljanja na kraju

```
i = m;  
if (i < t) {  
    do {  
        k = k + 2 * i;  
        i = i + n;  
    } while (i < t)  
}
```



Nedostatak `do-while` petlje u ovom slučaju je u tome što je nužna dodatna provjera treba li obaviti prvi prolaz kroz tijelo petlje.

# Primjer

- Učitati nenegativan cijeli broj  $n$  (nije potrebno provjeravati je li učitani ispravan broj). Izračunati i ispisati  $n$  *faktorijela* u obliku:

$n! = \text{xxxxxx}$

- Zadatak riješiti na tri načina:
  - petljom s ispitivanjem uvjeta na početku
  - petljom s ispitivanjem uvjeta na kraju
  - petljom s poznatim brojem ponavljanja
- Za ispis  $n$  *faktorijela* koristiti formatsku specifikaciju %g

%g - formatska specifikacija za ispis realnog broja u "znanstvenoj" ili "standardnoj" notaciji.

broj		ispis
0.0000002	→	2e-007
0.25	→	0.25
128.1	→	128.1
128000000.0	→	1.28e+008

# Rješenje (1)

- S programskom petljom s ispitivanjem uvjeta na početku

```
#include <stdio.h>
int main(void) {
    int n, i;
    double fakt;
    scanf ("%d", &n);
    fakt = 1.;
    i = 1;
    while (i <= n) {
        fakt = fakt * i;
        i = i + 1;
    }
    printf ("%d! = %g\n", n, fakt);
    return 0;
}
```

# Rješenje (2)

- S programskom petljom s ispitivanjem uvjeta na kraju

```
#include <stdio.h>
int main(void) {
    int n, i;
    double fakt;
    scanf ("%d", &n);
    fakt = 1.;
    i = 1;
    do {
        fakt = fakt * i;
        i = i + 1;
    } while (i <= n);
    printf ("%d! = %g\n", n, fakt);
    return 0;
}
```

# Rješenje (3)

- S programskom petljom s poznatim brojem ponavljanja

```
#include <stdio.h>
int main(void) {
    int n, i;
    double fakt;
    scanf ("%d", &n);
    fakt = 1.;
    for (i = 1; i <= n; i = i + 1) {
        fakt = fakt * i;
    }
    printf ("%d! = %g\n", n, fakt);
    return 0;
}
```



# Rezultati testiranja

---

Ulaz:

Ispis na zaslon

0	$0! = 1$
1	$1! = 1$
5	$5! = 120$
10	$10! = 3.6288e+006$
100	$100! = 9.33262e+157$
150	$150! = 5.71338e+262$
170	$170! = 7.25742e+306$
171	$171! = \text{inf}$

# Komentar

---

```
/* ne pisati ovako! */  
i=1; fakt=1.; do fakt *= i++; while (i <= n);
```

- Prihvatljiva rješenja:

```
    i = 1;  
    fakt = 1.0;  
    do {  
        fakt *= i++;  
    } while (i <= n);
```

```
    i = 1;  
    fakt = 1.0;  
    do {  
        fakt *= i;  
    } while (++i <= n);
```

# Primjer

---

- Učitati nenegativan cijeli broj  $n$  (nije potrebno provjeravati je li učitani ispravan broj). Ispisati prvih  $n$  Fibonaccijevih brojeva:  
1, 1, 2, 3, 5, 8, 13, 21, ...
- Algoritam za računanje Fibonaccijevih brojeva:  
 $\text{Fibonacci}(0) = \text{Fibonacci}(1) = 1$   
 $\text{Fibonacci}(n) = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$ , za  $n > 1$

# Rješenje

---

```
#include <stdio.h>
int main(void) {
    int n, i, f0 = 1, f1 = 1, f = 1;
    printf ("Upisite broj Fibonaccijevih brojeva: ");
    scanf ("%d", &n);
    for (i = 0; i < n; ++i) {
        if (i > 1) {
            f = f1 + f0;
            f0 = f1;
            f1 = f;
        }
        printf ("Fibonnaci (%d) = %d \n", i , f);
    }
    return 0;
}
```

# Primjer

- Ispisati tablicu množenja do 100 (10 redaka i stupaca).

```
#include <stdio.h>
int main(void) {
    int i, j;
    for (i = 1; i <= 10; ++i) {
        for (j = 1; j <= 10; ++j) {
            printf("%4d", i*j);
        }
        printf("\n");
    }
    return 0;
}
```

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

# Beskonačne petlje

---

## ■ Beskonačna petlja

- Petlja čije bi se tijelo izvodilo beskonačno mnogo puta kada ne bi sadržavala naredbu za izlazak iz petlje (**break**), naredbu za povratak iz funkcije (**return**), poziv funkcije za završetak programa (**exit**) ili **goto** naredbu.

## ■ Primjeri:

```
while(1 == 1) { ... }  
while(1) { ... }  
do { ... } while(1 == 1);  
...
```

## ■ Primjer (loš):

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

# Naredbe `break` i `continue`

---

- U C-u postoje dvije naredbe za kontrolu toka petlje

**`break`**        prekida izvođenje najbliže vanjske programske petlje

**`continue`**    unutar petlji `while` i `do-while` usmjerava izvođenje programa na ispitivanje uvjeta. Unutar petlje `for` usmjerava izvođenje programa na korak petlje `for` ("izraz3") i potom na ispitivanje uvjeta ("izraz2"). Također se odnosi na najbližu vanjsku petlju.

# Primjer

---

- Napisati program koji će učitavati cijele brojeve s tipkovnice i postupati prema sljedećem pravilu: ako je učitani broj manji od nule, treba ispisati poruku `Nedopustena vrijednost` i prestati s učitavanjem brojeva. Ako je učitani broj veći od 100, treba ga zanemariti, ispisati poruku `zanemarujem vrijednost` i nastaviti s učitavanjem, a inače treba ispisati učitani broj. Osim u slučaju pogreške s učitavanjem brojeva prestati kada se učitava (i ispiše) broj nula.



# Rješenje

---

```
...
int x;
do {
    printf ("Upisite broj : \n");
    scanf ("%d", &x );
    if (x < 0) {
        printf ("Nedopustena vrijednost\n");
        break;          /* Izlazak iz petlje */
    }
    if (x > 100) {
        printf ("Zanemarujem vrijednost\n");
        continue;       /* Skok na novu iteraciju */
    }
    printf ("Upisani broj je : %d\n", x);
} while (x != 0);
```

...

Za vježbu riješiti bez `break` i `continue`. Dobit će se jednostavnije i znatno bolje rješenje!

# Primjer

- Što će se ispisati sljedećim programskim odsječkom?

```
i = 1;
while (i < 5) {
    if (i == 3) {
        printf ("\n Hello world %d.x!", i);
        continue;
    } else if (i == 4) {
        printf ("Goodbye %d.x!", i);
        continue;
    }
    ++i;
}
```

## Rješenje:

Hello world 3.x!

Hello world 3.x!

... i tako beskonačno puta. Kada *i* dostigne vrijednost 3 izvršava se blok naredbi pod "*i*==3". Zbog `continue` se *i* ne povećava nego se program grana na uvjetni izraz (*i* < 5).

# Primjer

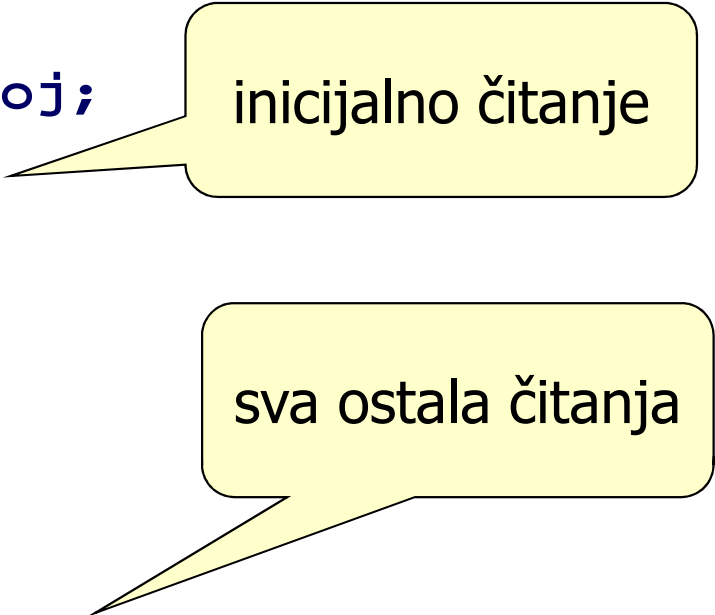
---

- Učitavati cijele brojeve dok se ne unese broj nula. Izračunati aritmetičku sredinu pozitivnih učitanih brojeva. Učitani negativni brojevi i nula ne ulaze u prosjek.

# Rješenje (1)

- Petlja s ispitivanjem uvjeta ponavljanja na početku:

```
#include <stdio.h>
int main(void) {
    int suma = 0, n = 0, broj;
    scanf ("%d", &broj);
    while (broj != 0) {
        if (broj > 0) {
            suma += broj;
            ++n;
        }
        scanf ("%d", &broj);
    }
    if (n > 0) printf("Prosjek =%f\n", (float) suma/n);
    return 0;
}
```




inicijalno čitanje

sva ostala čitanja

# Rješenje (2)

- Beskonačna petlja + break + continue:

```
#include <stdio.h>
int main(void) {
    int suma = 0, n = 0, broj;
    while (1 == 1) {
        scanf ("%d", &broj);
        if (broj == 0) break;
        if (broj < 0) continue;
        suma += broj;
        ++n;
    }
    if (n > 0) printf("Prosjek =%f\n", (float)suma/n);
    return 0;
}
```



beskonačna petlja

# Rješenje (3)

- Petlja s ispitivanjem uvjeta ponavljanja na kraju:

```
#include <stdio.h>
int main(void) {
    int suma = 0, n = 0, broj;
    do {
        scanf ("%d", &broj);
        if (broj > 0) {
            suma += broj;
            ++n;
        }
    } while (broj != 0);
    if (n > 0) printf("Prosjek =%f\n", (float)suma/n);
    return 0;
}
```

najbolje rješenje

# Primjer

---

- Učitati prirodni broj (nije potrebno provjeravati je li učitani ispravan broj). Na zaslon ispisati jednu od sljedećih poruka:

*`n jest prost broj !`*

*`n nije prost broj !`*

# Rješenje

---

```
#include <stdio.h>
int main(void) {
    int i, n, prost = 1;          /* prost = true */
    printf("Upisite prirodan broj: ");
    scanf("%d", &n);
    for (i = 2; i <= n - 1; ++i) {
        if (n % i == 0) {
            prost = 0;            /* prost = false */
            break;
        }
    }
    if (prost)
        printf("%d jest prost broj !\n", n);
    else
        printf("%d nije prost broj !\n", n);
    return 0;
}
```



# Moguća poboljšanja prethodnog rješenja

---

- Moguća poboljšanja algoritma (smanjivanje broja iteracija petlje):
  - Dovoljno je s petljom ići samo do  $\sqrt{n}$  (C funkcija sqrt(n))
  - Ispitati djeljivost broja s 2, te ako nije djeljiv s 2, unutar petlje ispitivati djeljivost samo s neparnim brojevima većim od 2
- 
- načiniti za vježbu!

Kontrolne naredbe  
- naredba goto -

# Naredba goto

---

Opći oblik naredbe:

```
goto oznaka_naredbe_1;  
    . . .  
    . . .  
oznaka_naredbe_1:  
    programski odsječak
```

# Primjer

- Napisati programski odsječak koji će učitavati pozitivne brojeve dok su manji ili jednaki 100. Ako se unese negativni broj "poduzeti odgovarajuće korake".

```
...
/* programska petlja za citanje pozitivnih brojeva */
scanf("%d", &x);
while (x <= 100) {
    if (x < 0) goto pogreska;
    ...
    scanf("%d", &x);
}
...
/* odsjecak u kojem se reagira na pogresku */
pogreska:
    printf("POGRESKA - NEGATIVAN BROJ");
    ...
```

# Strukturiranim programiranjem protiv neprihvatljive uporabe goto

```
for (i = 0; i < 10; ++i) {  
    programski odsječak;  
}
```

najbolje rješenje



```
i = 0;  
while (i < 10) {  
    programski odsječak;  
    ++i;  
}
```

prihvatljivo rješenje



```
i = 0;  
opet:  
    if (i >= 10) goto dalje;  
    programski odsječak;  
    ++i;  
    goto opet;  
dalje:  
...
```

neprihvatljivo rješenje

# Jedan od rijetkih primjera prikladnih za uporabu naredbe goto

---

```
for (...; uvjet1; ...) {  
    while (uvjet2) {  
        do {  
            ...  
            if (uvjet) prekinuti sve tri petlje;  
            ...  
        } while (uvjet3);  
        ...  
    }  
    ...  
}  
  
nastavak;
```

# Rijetki primjer dopuštene uporabe naredbe goto

---

```
for (...; uvjet1; ...) {  
    while (uvjet2) {  
        do {  
            ...  
            if (uvjet) goto van;  
            ...  
        } while (uvjet3);  
        ...  
    }  
    ...  
}
```

```
van:  
    nastavak;
```

# Strukturirano programiranje

## Rješenje istog problema bez goto

---

```
int gotovo = 0;
for (...; uvjet1; ...) {
    while (uvjet2) {
        do {
            ...
            if (uvjet) {    /* prekinuti sve tri petlje */
                gotovo = 1;
                break;
            }
            ...
        } while (uvjet3);
        if (gotovo) break;
        ...
    }
    if (gotovo) break;
    ...
}
```

*nastavak;*



# Kontrolne naredbe

## - skretnica -

# Naredba `switch` - skretnica

---

- može se upotrijebiti umjesto višestране `if` selekcije
- sintaksa:

```
switch( cjelobrojni_izraz ) {  
    case const_izraz1: naredbe1;  
    [case const_izraz2: naredbe2;]  
    ...  
    [default : naredbeN;]  
}
```

- izraz unutar zagrada iza switcha **mora** biti cjelobrojan
- izrazi iza `case` moraju biti cjelobrojni i sadrže samo konstante (npr. `3*7+'a'` je O.K., `3*i` ne valja)
- obratiti pažnju: ako se unutar bloka `case` ne navede ključna riječ `break`; nastavak programa je sljedeći blok `case` u listi!

# Primjer

```
#include <stdio.h>
int main(void) {
    char c;
    scanf ("%c", &c);
    switch (c) {
    case 'A':
        printf ("c = 'A'\n");
    case 'B':
        printf ("c = 'B'\n");
    default:
        printf ("Pogreska\n");
    }
    return 0;
}
```

Ulazni podatak: A

Rezultat izvođenja:

```
c = 'A'
c = 'B'
Pogreska
```

Ulazni podatak: B

Rezultat izvođenja:

```
c = 'B'
Pogreska
```

# Rješenje sa switch i break

---

```
...  
char c;  
scanf ("%c", &c);  
switch (c) {  
    case 'A':  
        printf ("c = A'\n");  
        break;  
    case 'B':  
        printf ("c = 'B'\n");  
        break;  
    default:  
        printf ("Pogreska\n");  
        break;  
}  
...
```

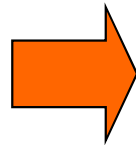
Ulazni podatak: B  
Rezultat izvođenja:

```
c = 'B'
```

# Realizacija skretnice koja sadrži **break**, korištenjem naredbe **if - else if - else**

---

```
switch (vr) {  
    case C1:  
        naredbe_1;  
        break;  
    case C2:  
        naredbe_2;  
        break;  
    case Cn:  
        naredbe_n;  
        break;  
    default:  
        naredbe_n_plus1;  
}
```

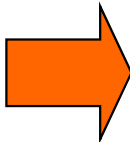


```
if (vr == C1) {  
    naredbe_1;  
} else if (vr == C2) {  
    naredbe_2;  
} else if ...  
    ...  
} else if (vr == Cn) {  
    naredbe_n;  
} else {  
    naredbe_n_plus1;  
}
```

# Realizacija skretnice koja ne sadrži **break**, korištenjem naredbe **if**

---

```
switch (vr) {  
    case C1:  
        naredbe_1;  
    case C2:  
        naredbe_2;  
    case Cn:  
        naredbe_n;  
    default:  
        naredbe_n_plus1;  
}
```



```
nadjen = 0;  
if (vr == C1) {  
    naredbe_1;  
    nadjen = 1;  
}  
if (vr == C2 || nadjen) {  
    naredbe_2;  
    nadjen = 1;  
}  
...  
if (vr == Cn || nadjen) {  
    naredbe_n;  
}  
naredbe_n_plus1;
```

## Primjer: Učitati s tipkovnice brojevenu ocjenu (od 1 do 5), a zatim ispisati njezinu opisnu vrijednost

---

```
...
int ocjena;
printf ("Upisite ocjenu :");
scanf ("%d", &ocjena);
switch (ocjena) {
    case 1:
        printf ("Nedovoljan\n"); break;
    case 2:
        printf ("Dovoljan\n"); break;
    case 3:
        printf ("Dobar\n"); break;
    case 4:
        printf ("Vrlo dobar\n"); break;
    case 5:
        printf ("Izvrstan\n"); break;
    default:
        printf ("Unijeli ste nepostojeću ocjenu\n");
        break; /* može i bez naredbe break */
}
...
```

# Komentar rješenja

---

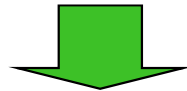
- Napomena: Ako bismo izostavili `break;` unutar svih `case` blokova, za učitanu vrijednost npr. 3 ispis bi izgledao ovako:  
Dobar  
Vrlo dobar  
Izvrstan  
Unijeli ste nepostojeću ocjenu
- Napomena: Ako je blok `default` posljednji blok naredbe `switch`, unutar njega nije obavezno napisati naredbu `break`, ali se preporuča.



"Propadanje" po labelama `case` može se iskoristiti onda kada se "ispod" nekoliko labela nalazi isti programski kôd

---

```
switch (znak) {  
    case 'a': ++brsamoglasnika; break;  
    case 'e': ++brsamoglasnika; break;  
    case 'i': ++brsamoglasnika; break;  
    case 'o': ++brsamoglasnika; break;  
    case 'u': ++brsamoglasnika; break;  
    default: ++brostalih; break;  
}
```



```
switch (znak) {  
    case 'a':  
    case 'e':  
    case 'i':  
    case 'o':  
    case 'u': ++brsamoglasnika; break;  
    default: ++brostalih; break;  
}
```