



# PROGRAMIRANJE I

---

## **P-05: Naredbe u programskom jeziku C**

prof. dr **Dražen Brđanin**  
2023/24



# P-05: Naredbe u programskom jeziku C

---

## ■ **Sadržaj predavanja**

- vrste naredbi
- naredbe izraza
- naredbe grananja i selekcije
- iterativne naredbe (petlje)
- naredbe za nasilnu kontrolu toka



# Klasifikacija naredbi u jeziku C

---

## Naredbe u jeziku C

- C raspolaže malim skupom naredbi
- Svaka naredba završava sa ;  
(terminator naredbe)
- Osnovne grupe naredbi:
  - naredbe izraza
  - naredbe grananja i selekcije:  
`if, switch`
  - naredbe petlji:  
`while, do while, for`
  - naredbe za nasilnu promjenu toka:  
`break, continue, goto, return`



# Naredba izraza

## Naredba izraza

- Osnovni oblik naredbe u jeziku C je **naredba izraza**
- Naredba izraza obuhvata i naredbu dodjele i naredbu poziva funkcije.

Npr:

```
3 + 4*5; // naredba izraza s cijom
          // vrijednoscu se nista
          // dalje ne radi

n = 3;    // naredba dodjele

c++;      // naredba izraza u kojoj se
          // inkrementuje promjenljiva c

f();      // naredba poziva funkcije

;         // prazna naredba
```

- **Komponovana/složena naredba** (blok)

- jednu ili više naredbi grupišemo korištenjem velikih zagrada
- opšti oblik:

```
{
    naredba1;
    naredba2;
    ...
    naredbaN;
}
```

- iza složene naredbe ne stavlja se terminator naredbe

**Složena naredba izvršava se tako što se izvršavaju jedna po jedna prosta naredba od kojih je sastavljena**

# Naredbe grananja

## Naredbe grananja

- Grananje u algoritmu (diskriminator) imlementira se pomoću naredbe **if**
- Tri vrste grananja: **jednoblokovsko**, **dvoblokovsko**, **višeblokovsko**

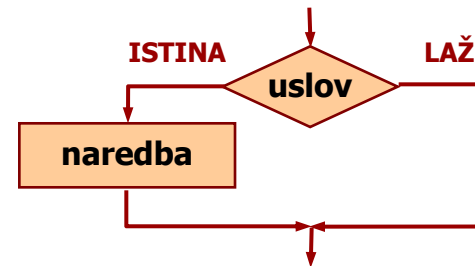
### Jednoblokovsko grananje

#### ➤ sintaksa

```
if (uslov) naredba;
```

```
if (uslov)
{
    naredba1;
    naredba2;
    ...
    naredbaN;
}
```

#### ➤ semantika



Izračunava se vrijednost uslova i ako je uslov istinit, izvršava se naredba (prosta ili složena).  
Ako uslov nije istinit, naredba se ignoriše.

# Naredbe grananja

## Jednoblokovsko grananje

### ➤ Pragmatika

C nije pozicioni jezik pa su ravnopravni različiti načini formatiranja

`if (u) n;`  $\Leftrightarrow$  `if (u)  
n;`

`if (u)  
{  
n1;  
...  
nN;  
}`  $\Leftrightarrow$  `if (u) {  
n1;  
...  
nN;  
}`

uslov je logički izraz, čijim se sračunavanjem dobija istinitosna vrijednost

- **Uslov ne može da se izostavi**

`if ( ) n;` ✗

- **Uslov ne mora nužno da bude logički izraz**

`if ( i+2 ) n;`

`i+2` je izraz čijim se sračunavanjem dobija neka vrijednost koja se intepretira kao istinitosna – ako je rezultat različit od nule izvršiće se naredba

- **Uslov ne mora nužno da bude logički izraz**

`if ( i=2 ) n;` ✗

`if ( i==2 ) n;` ✓

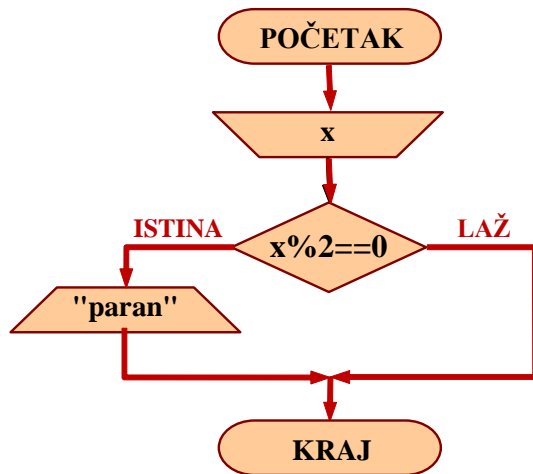
Često se greškom umjesto operatora poređenja koristi operator dodjele pa je u gornjem slučaju (`i=2`) uslov uvijek tačan

# Naredbe grananja

## Jednoblokovsko grananje

### Primjer:

Program koji provjerava da li je učitani broj paran.



```
#include <stdio.h>
int main()
{
    int x;
    printf( "Unesite broj: " );
    scanf( "%d", &x );
    if ( x%2==0 )
        printf( "Broj je paran\n" );
    return 0;
}
```

```
Unesite broj: 10
Broj je paran
```

```
Unesite broj: 9
```

# Naredbe grananja

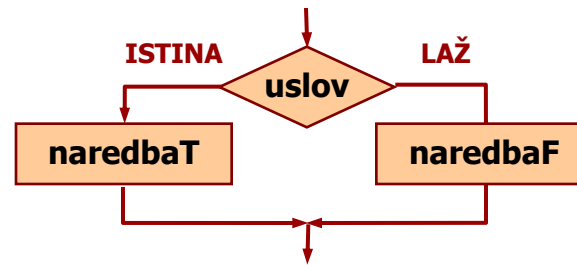
## Dvoblokovsko grananje

### ➤ sintaksa

```
if (uslov)
    naredbaT;
else
    naredbaF;
```

```
if (uslov)
{
    naredbaT1;
    ...
    naredbaTN;
}
else
{
    naredbaF1;
    ...
    naredbaFN;
}
```

### ➤ semantika



Izračunava se vrijednost uslova.

Ako je uslov istinit, izvršava se naredbaT (prosta ili složena).  
Ako uslov nije istinit, izvršava se naredbaF (prosta ili složena).

### ➤ Pragmatika

Važi sve što je navedeno za pragmatiku jednoblokovskog grananja!





# Naredbe grananja

---

## Dvoblokovsko grananje


### ➤ Pragmatika

- Naredbe koje se izvršavaju uslovno mogu da sadrže nove naredbe uslova, tj. može biti više ugnježđenih if naredbi.
- Ukoliko vitičastim zagradama nije obezbijeđeno drugačije, klauzula else se odnosi na posljednji prethodni neupareni if.
- Ukoliko se želi drugačije ponašanje, neophodno je navesti vitičaste zagrade.


**Npr:**

**Da li se else odnosi na if (u1) ili na if (u2) ?**

```
if ( u1 )  
    if ( u2 )  
        n1;  
else  
    n2;
```



```
if ( u1 )  
    if ( u2 ) n1;  
else  
    n2;
```



# Naredbe grananja

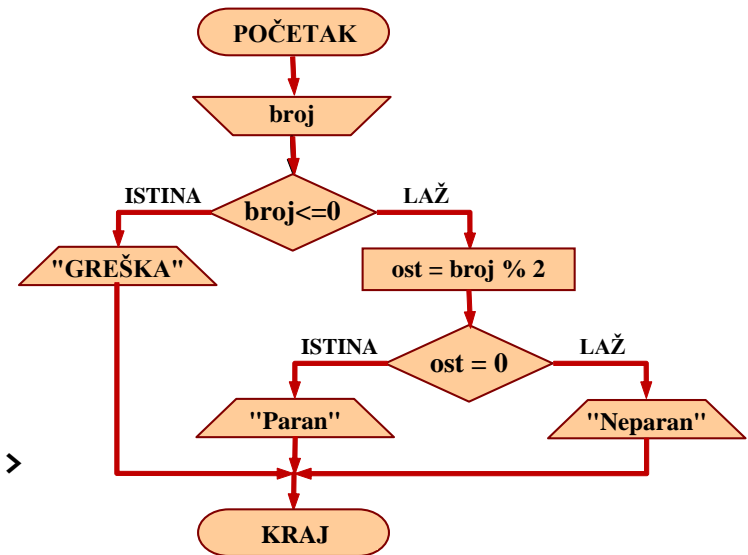
## Dvoblokovsko grananje

### Primjer:

Program koji provjerava da li je učitani prirodni broj paran ili neparan.

```
#include <stdio.h>
int main()
{
    int broj, ost;
    printf( "Broj? " );
    scanf( "%d", &broj );
    if ( broj<=0 )
        printf("Nije prirodan");
    else
    {
        ost = broj % 2;
        if ( ost==0 )
            printf("Paran");
        else
            printf("Neparan");
    }
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int broj;
    printf( "Broj? " );
    scanf( "%d", &broj );
    if ( broj<=0 )
        printf("Nije prirodan");
    else
        if (broj % 2)
            printf("Neparan\n");
        else
            printf("Paran\n");
    return 0;
}
```



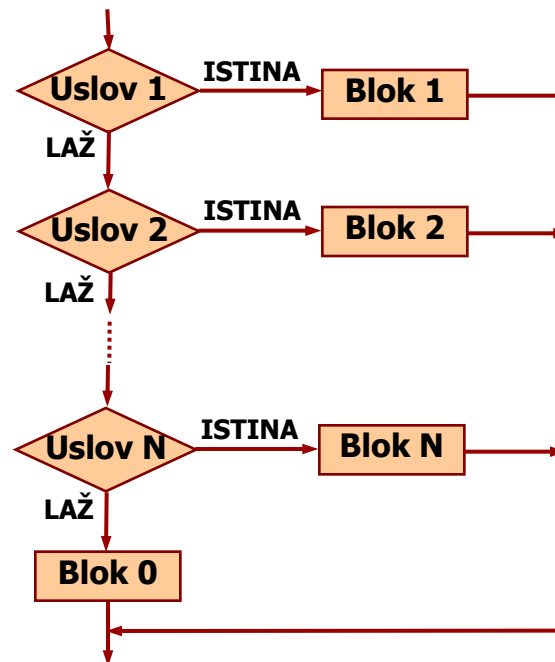
# Naredbe grananja

## Višeblokovsko grananje

### ➤ sintaksa

```
if (uslov1)
    naredba1;
else if (uslov2)
    naredba2;
else if (uslov3)
    naredba3;
else if (uslov4)
    ...
else if (uslovN)
    naredbaN;
[else
    naredba0;]
```

### ➤ semantika



Ako je uslov1 istinit, izvršava se pripadajuća naredba1 (prosta ili složena).

Inače (ako nije ispunjen uslov1), ako je ispunjen uslov2, izvršava se naredba2 (prosta ili složena).

Inače (ako nije ispunjen ni uslov1 ni uslov2), ako je ispunjen uslov3, izvršava se naredba3 (prosta ili složena).

Itd.

Ako nije ispunjen nijedan uslov, izvršava se naredba0 (prosta ili složena)

### ➤ Pragmatika

Važi sve što je navedeno za pragmatiku jednoblokovskog i dvoblokovskog grananja!

Posljednja else klauzula i pripadajuća naredba0 mogu da se izostave

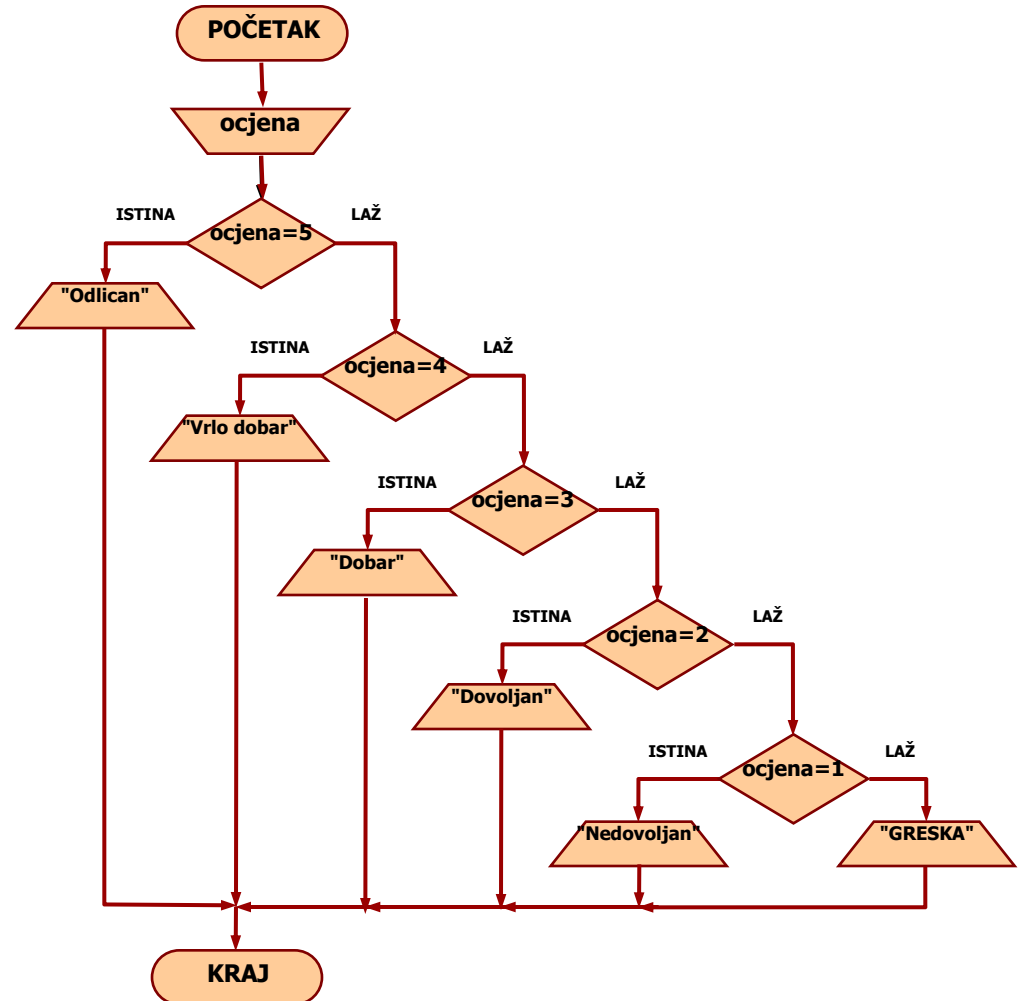
# Naredbe grananja

## Višeblokovsko grananje

### Primjer:

Program koji za učitane numeričku ocjenu ispisuje opisnu ocjenu.

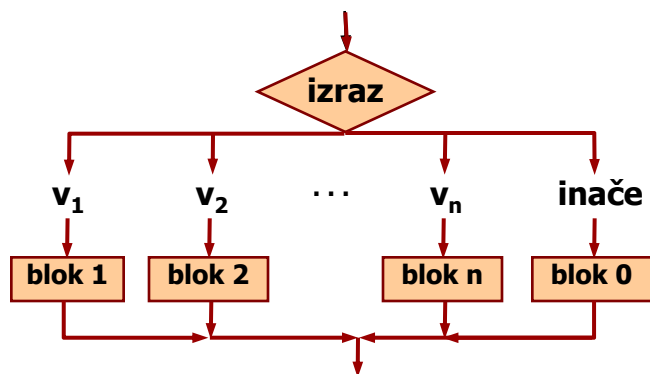
```
#include <stdio.h>
int main()
{
    int ocjena;
    printf( "Ocjena? " );
    scanf( "%d", &ocjena );
    if (ocjena==5)
        printf("Odlican\n");
    else if (ocjena==4)
        printf("Vrlo dobar\n");
    else if (ocjena==3)
        printf("Dobar\n");
    else if (ocjena==2)
        printf("Dovoljan\n");
    else if (ocjena==1)
        printf("Nedovoljan\n");
    else
        printf("GRESKA\n");
    return 0;
}
```



# Naredbe selekcije

## Naredbe selekcije

- Višeblokovsko grananje ponekad nije praktično, zbog velikog broja uslovnih naredbi
- Alternativno mogu da se koriste naredbe selekcije (izbora)
- U programskim jezicima tipično se implementira u jednom od dva oblika



Izračunava se vrijednost selektorskog izraza i na osnovu izračunate vrijednosti pronalazi odgovarajući slučaj.

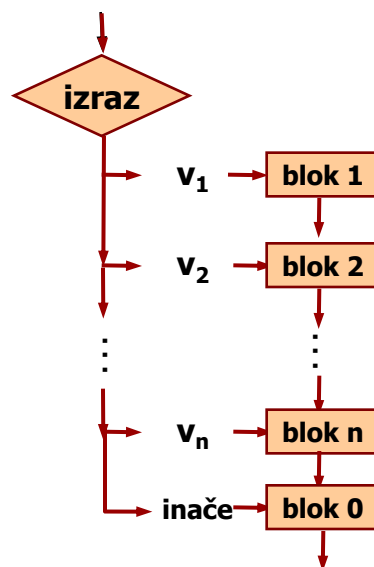
Ako je  $\text{izraz} == v_1 \Rightarrow$  izvršava se samo blok 1

Ako je  $\text{izraz} == v_2 \Rightarrow$  izvršava se samo blok 2

...

Ako je  $\text{izraz} == v_n \Rightarrow$  izvršava se samo blok n

inače se izvršava podrazumijevani samo blok 0



Izračunava se vrijednost selektorskog izraza i na osnovu izračunate vrijednosti pronalazi odgovarajući slučaj i izvršavanje nastavlja od tog slučaja.

Ako je  $\text{izraz} == v_1 \Rightarrow$  izvršava se blok 1 i svi blokovi iza bloka 1.

Ako je  $\text{izraz} == v_2 \Rightarrow$  izvršava se blok 2 i svi blokovi iza bloka 2

...

# Naredbe selekcije

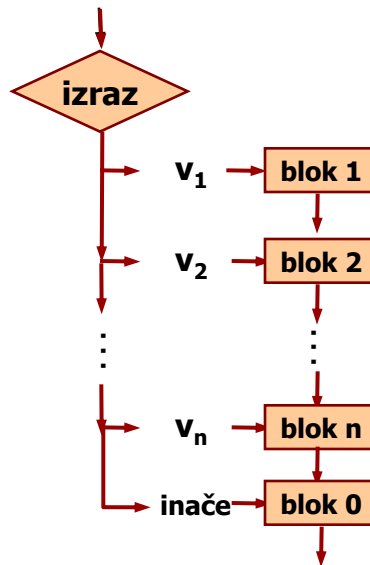
## Naredbe selekcije

➤ U programskom jeziku C selekcija se implementira naredbom switch

➤ sintaksa

```
switch (izraz)
{
    case v1:
        naredba11;
        ...
        naredba1N;
    case v2:
        naredba21;
        ...
        naredba2N;
    :
    case vN:
        naredbaN1;
        ...
        naredbaNN;
    [default:
        iskaz0;]
}
```

➤ semantika



Izračunava se vrijednost selektorskog izraza i na osnovu izračunate vrijednosti pronalazi odgovarajući slučaj i izvršavanje nastavlja od tog slučaja.

Ako je  $\text{izraz} = v1 \Rightarrow$  izvršava se blok 1 i svi blokovi iza bloka 1.

Ako je  $\text{izraz} = v2 \Rightarrow$  izvršava se blok 2 i svi blokovi iza bloka 2

...



# Naredbe selekcije

---

## Naredbe selekcije - switch

### ➤ Pragmatika

- Vrijednost selektorskog izraza mora biti nekog prebrojivog tipa (cjelobrojne vrijednosti)
- ukoliko isti blok treba da se izvrši u više različitih slučajeva

```
switch (izraz)
{
    case v1:
        naredba1;
    case v2:
    case v3:
        naredba23;
    ...
    [default:
        iskaz0;]
}
```



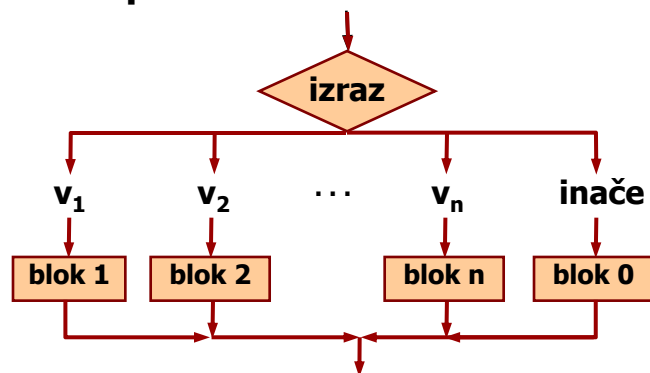
```
switch (izraz)
{
    case v1:
        naredba1;
    case v2,v3:
        naredba23;
    ...
    [default:
        iskaz0;]
}
```



# Naredbe selekcije

## Naredbe selekcije - switch

- Podrazumijevana semantika može da se izmijeni naredbom **break**
- Naredba **break** prekida izvršavanje naredbe **switch** – prekida se izvršavanje datog bloka i prelazi na prvu naredbu iza naredbe **switch**



Izračunava se vrijednost selektorskog izraza i na osnovu izračunate vrijednosti pronalazi odgovarajući slučaj.

Ako je  $\text{izraz} == v_1 \Rightarrow$  izvršava se samo blok 1, jer na kraju bloka 1 postoji naredba **break** kojom se prekida izvršavanje naredbe **switch**.

...

```
switch (izraz)
{
    case v1:
        naredba1;
        break;
    case v2:
        naredba2;
        break;
    :
    :
    case vN:
        naredbaN;
        break;
    [default:
        iskaz0;]
}
```

Naredba **break** nije obavezna i ne mora se koristiti u svim slučajevima





# Naredbe selekcije

---

## Primjer:

Program koji za učitano numeričku ocjenu ispisuje opisnu ocjenu koristeći selektivno višeblokovsko grananje.

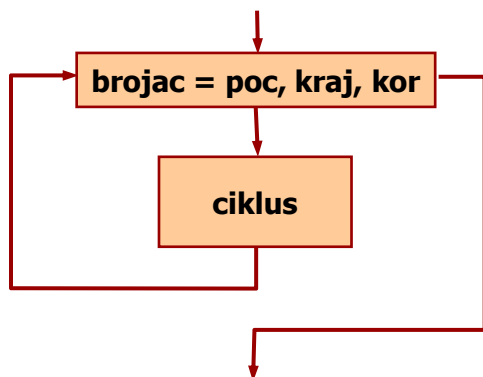
```
#include <stdio.h>
int main()
{
    int ocjena;
    printf("Ocjena? ");
    scanf("%d", &ocjena);
    switch (ocjena)
    {
        case 5:
            printf("Odlican\n"); break;
        case 4:
            printf("Vrlo dobar\n"); break;
        case 3:
            printf("Dobar\n"); break;
        case 2:
            printf("Dovoljan\n"); break;
        case 1:
            printf("Nedovoljan\n"); break;
        default:
            printf("GRESKA\n");
    }
    return 0;
}
```

# Ciklične algoritamske strukture

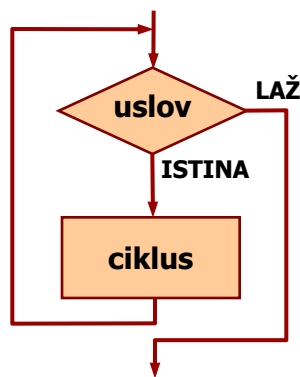
## Ciklična algoritamska struktura (petlja)

- Ponekad je potrebno neke algoritamske korake ponoviti. Takvi **algoritamski koraci, koji se ponavljaju**, predstavljaju **ciklus**.
- Algoritamske strukture koje sadrže ciklus nazivaju se **ciklične algoritamske strukture** ili **petlje**.
- U teoriji je poznato više različitih oblika petlji.
- U programskim jezicima obično se implementiraju:

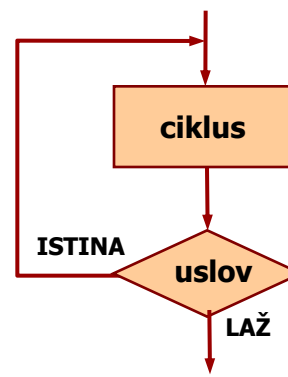
**petlja s fiksnim brojem izvršavanja**  
(brojačka petlja)



**petlja s izlazom na vrhu**  
(uslovni ciklus s izlazom na vrhu)



**petlja s izlazom na dnu**  
(uslovni ciklus s izlazom na dnu)

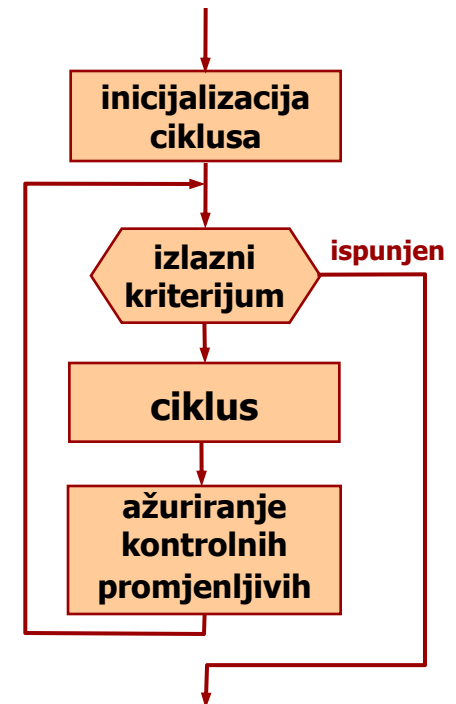


# Ciklične algoritamske strukture

## Osnovni elementi svake petlje

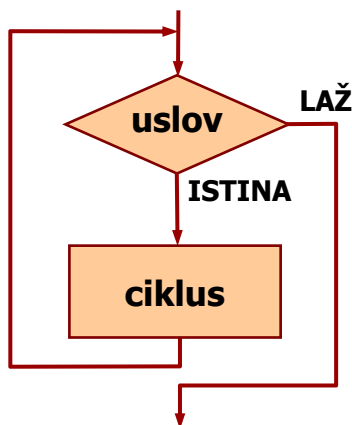
➤ Svaka petlja, bez obzira na strukturu, ima sljedeće tipične elemente:

- da bi se obezbijedio potreban broj ponavljanja ciklusa, obično se koristi **jedna ili više kontrolnih promjenljivih**;
- prije nego što se uđe u petlju postavljaju se potrebne **početne vrijednosti kontrolnih promjenljivih**;
- petlja mora da ima **uslov pod kojim se izlazi iz petlje** (prekida s ponavljanjem ciklusa) – ovaj uslov naziva se **izlazni kriterijum**;
- sve dok se ne zadovolji izlazni kriterijum, ponavljaju se algoritamski koraci koji čine tijelo petlje (ciklus);
- po završetku svakog ciklusa treba ažurirati vrijednosti kontrolnih promjenljivih.



# Ciklične algoritamske strukture

## Petlja sa izlazom na vrhu (uslovni ciklus sa izlazom na vrhu)



### ➤ Semantika:

1. Izračunava se vrijednost uslova
2. Ako je uslov istinit tada:
  - 2.1. izvršava se ciklus
  - 2.2. skok na korak 1.
3. Inače (ako uslov nije istinit), prekini izvršavanje petlje

**Ciklus će se ponavljati sve dok je uslov istinit  
(ne mora ni jednom!)**

### Primjer:

Algoritam koji za učitani prirodan broj izračunava i ispisuje zbir njegovih cifara.

### Kako izdvojiti cifru?

$315 \% 10 \rightarrow 5$

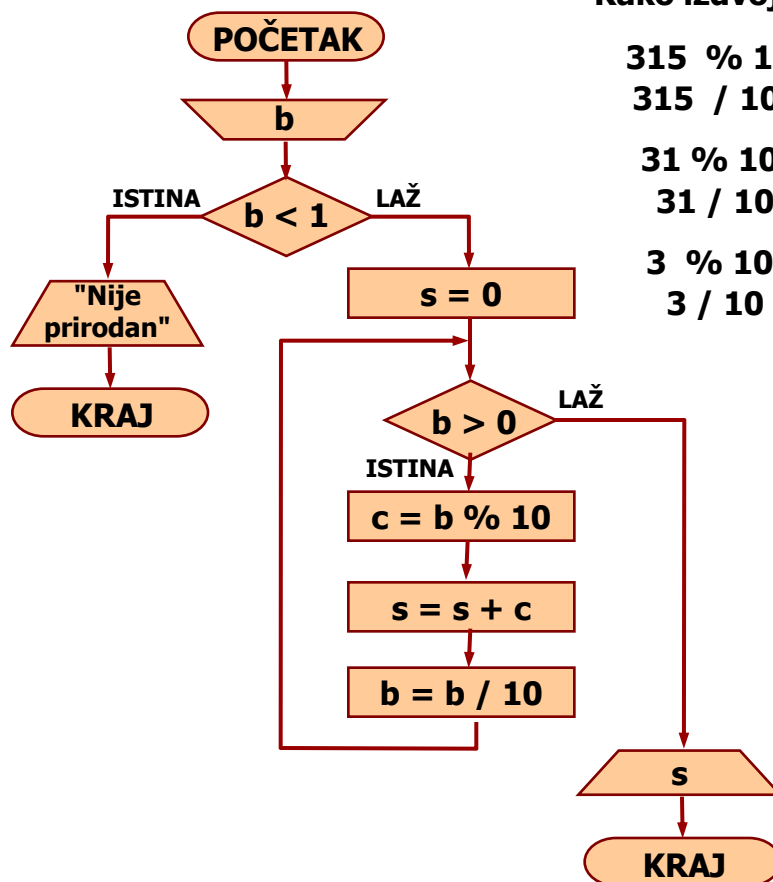
$315 / 10 \rightarrow 31$

$31 \% 10 \rightarrow 1$

$31 / 10 \rightarrow 3$

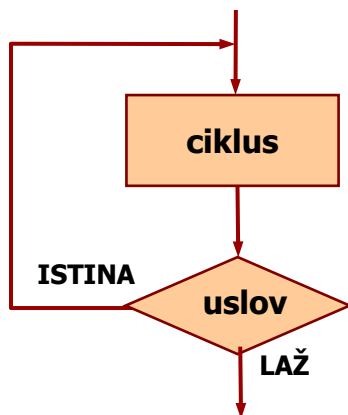
$3 \% 10 \rightarrow 3$

$3 / 10 \rightarrow 0$



# Ciklične algoritamske strukture

## Petlja sa izlazom na dnu (uslovni ciklus sa izlazom na dnu)



### ➤ Semantika (**pozitivna logika**):

1. Izvršava se ciklus
2. Izračunava se vrijednost uslova
3. Ako je uslov istinit, skok na korak 1.
4. Inače (ako uslov nije istinit, prekini izvršavanje petlje

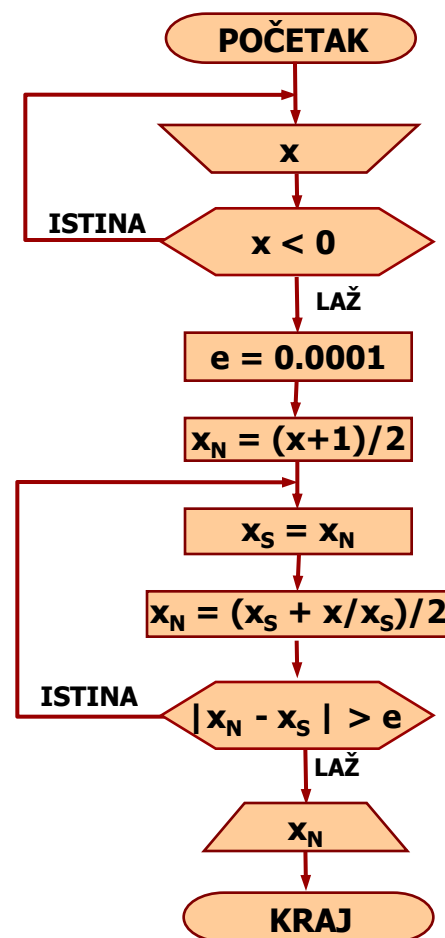
**Ciklus će se ponavljati sve dok je uslov istinit  
(mora bar jednom!)**

### Primjer:

Algoritam koji računa drugi korijen iz pozitivnog realnog broja  $X$  na četiri decimale koristeći Njutnovu iterativnu formulu:

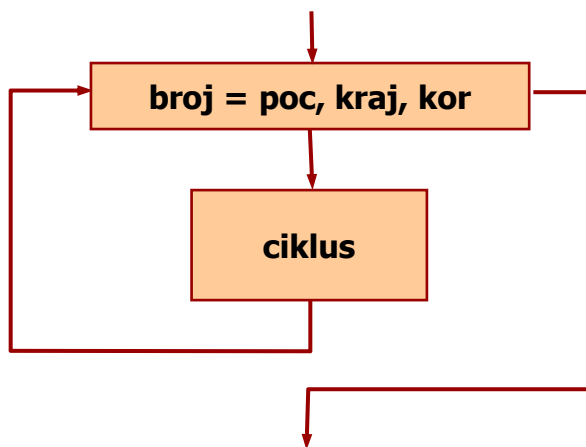
$$x_0 = \frac{X+1}{2}$$

$$x_{n+1} = \frac{x_n + \frac{X}{x_n}}{2}, \quad n = 0, 1, 2, \dots$$



# Ciklične algoritamske strukture

## Brojačka petlja



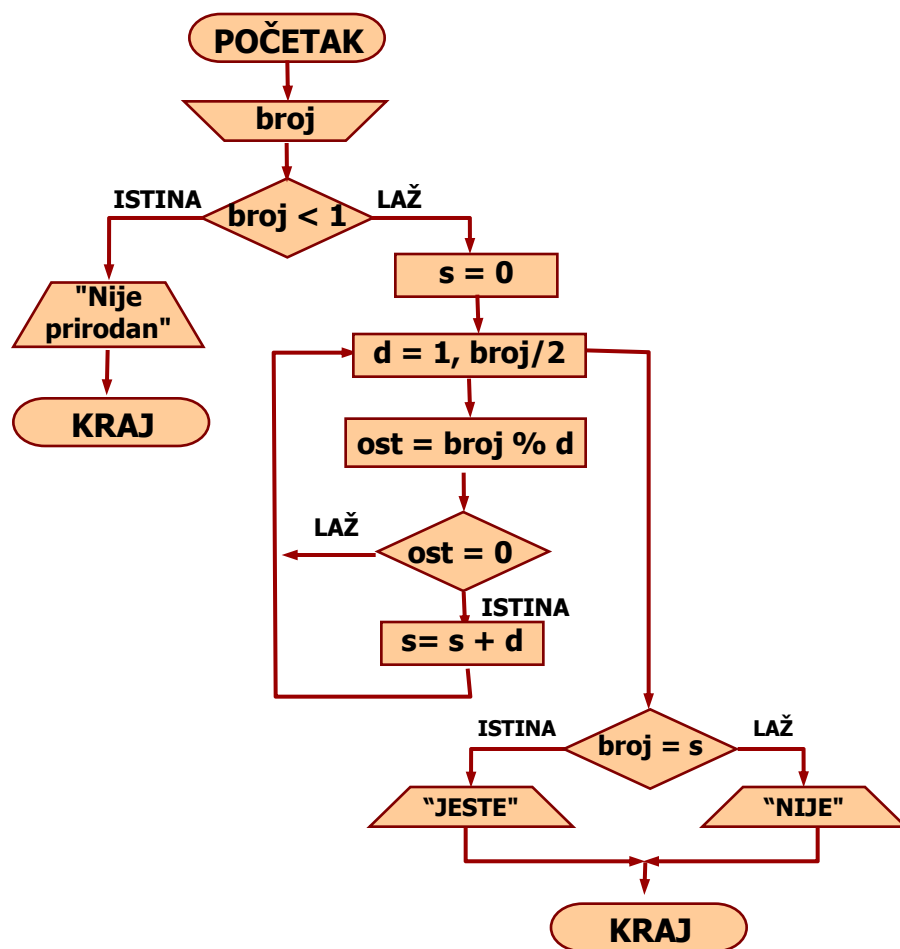
Ako je korak  
jedinični,  
može da se  
izostavi

## Semantika (korak **pozitivan**/negativan):

1. Brojač se inicijalizuje početnom vrijednošću ( $\text{broj} = \text{poc}$ )
2. Ako je  $\text{broj} \leq \text{kraj}$  ( $\text{broj} \geq \text{kraj}$ ), tada
  - 2.1. izvršava se ciklus
  - 2.2. **uvećava se**/**umanjuje se** vrijednost brojača za vrijednost koraka ( $\text{broj} = \text{broj} + \text{kor}$ ) ( $\text{broj} = \text{broj} - \text{kor}$ )
  - 2.3. skok na korak 2.
4. Inače (ako je  $\text{broj} > \text{kraj}$ , odnosno  $\text{broj} < \text{kraj}$ ), prekini izvršavanje petlje

## Primjer:

Algoritam koji provjerava da li je učitani prirodni broj savršen.

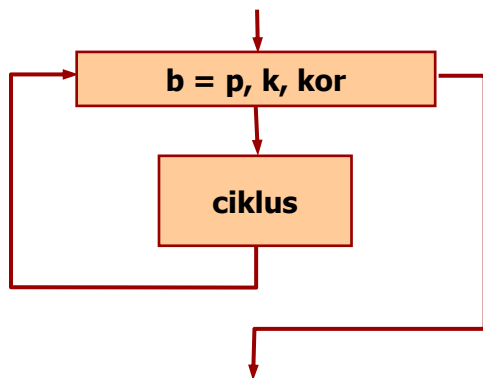


# Iterativne naredbe

## Iterativne naredbe

- Iterativne naredbe **omogućavaju realizaciju petlji.**
- Alternativni termini u literaturi: **naredbe ponavljanja / repetitivne naredbe / petlje**
- Različite implementacije u različitim jezicima:

**brojačka petlja**



**Tipična implementacija:  
naredba FOR**

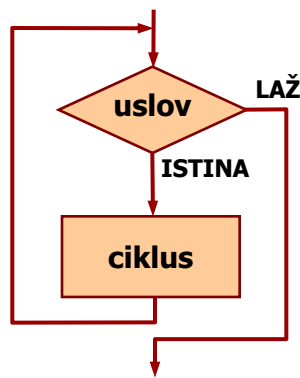
**PASCAL: FOR b:=p TO k DO**

**PASCAL: FOR b:=p DOWNTO k DO**

**C, C++: for (b=p; b<=k; b+=kor)**

**C, C++: for (b=p; b>=k; b-=kor)**

**petlja s izlazom na vrhu**



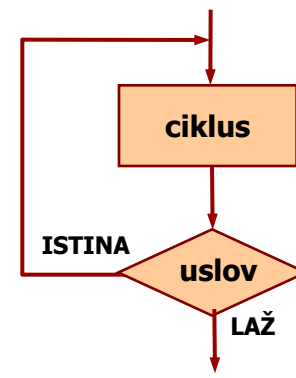
**Tipična implementacija:  
naredba WHILE**

**PASCAL: WHILE uslov DO**

**C, C++: while (uslov)**

**C, C++: for (poc\_izr; uslov; izraz)**

**petlja s izlazom na dnu**



**Tipična implementacija:  
naredba REPEAT / DO**

**PASCAL: REPEAT .. UNTIL uslov**

**C, C++: do .. while (uslov)**

# Iterativne naredbe u jeziku C

## Naredba while

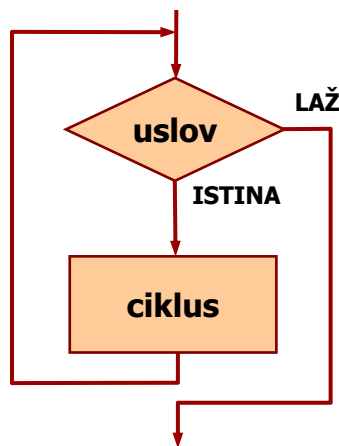
➤ Omogućava realizaciju petlje sa izlazom na vrhu.

➤ sintaksa

```
while (uslov) naredba;
```

```
while (uslov)
{
    naredba1;
    naredba2;
    ...
    naredbaN;
}
```

➤ semantika



1. Izračunava se vrijednost uslova
2. Ako je uslov istinit tada:
  - 2.1. izvršava se ciklus
  - 2.2. skok na korak 1.
3. Inače (ako uslov nije istinit), prekini izvršavanje petlje

**Ciklus će se ponavljati  
sve dok je uslov istinit  
(ne mora ni jednom!)**



# Iterativne naredbe u jeziku C

## Naredba while

### ➤ Pragmatika

C nije pozicioni jezik pa su ravnopravni različiti načini formatiranja

`while (u) n;`  $\Leftrightarrow$  `while (u)`  
`n;`

`while (u)`  
`{`  
`n1;`  
`...`  
`nN;`  
`}`  $\Leftrightarrow$  `while (u) {`  
`n1;`  
`...`  
`nN;`  
`}`

**uslov** je logički izraz,  
čijim se sračunavanjem dobija istinitosna vrijednost

- **Uslov ne može da se izostavi**

`while ( ) n;`



- **Uslov ne mora nužno da bude logički izraz**

`while ( i )`  
`n;`

"i" je izraz čijim se sračunavanjem dobija neka vrijednost koja se intepretira kao istinitosna – sve dok je  $i \neq 0$  ponavljaje se ciklus

- **Uslov ne mora nužno da bude logički izraz**

`while (i=2) n;`



`while (i==2) n;`



`while (1) n;`

`while (0) n;`



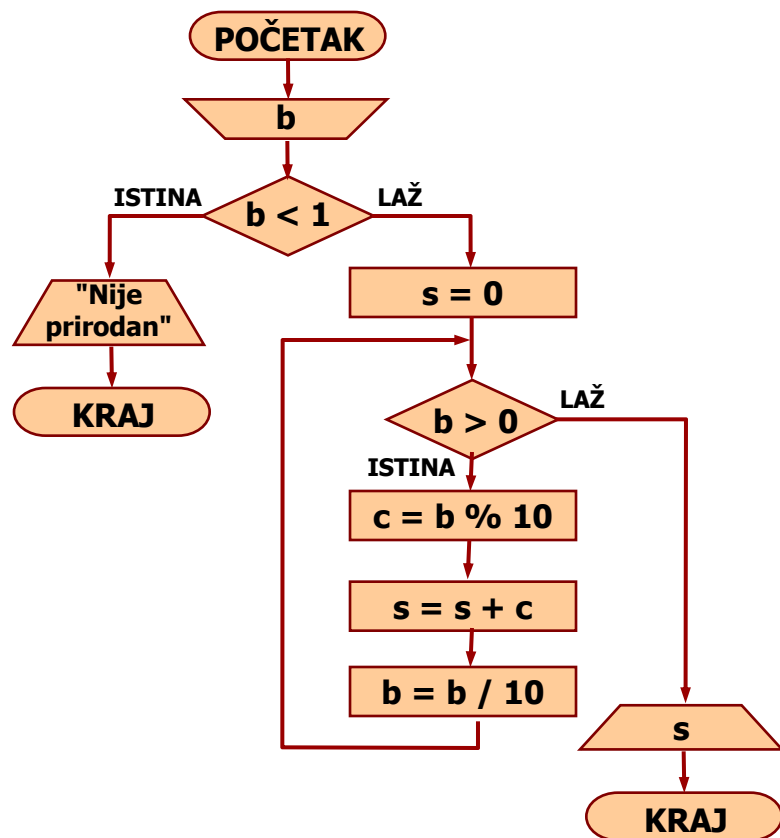
Često se greškom umjesto operatora poređenja koristi operator dodjele pa je u gornjem slučaju ( $i=2$ ) uslov uvijek tačan – **beskonačna petlja**

# Iterativne naredbe u jeziku C

## Naredba while

### Primjer:

Program koji za učitani prirodan broj izračunava i ispisuje zbir njegovih cifara.



```
#include <stdio.h>
int main()
{
    int b, s=0;
    printf("Unesite broj: ");
    scanf("%d", &b);
    if (b<1)
        printf("Nije prirodan!");
    else
    {
        while (b)
        {
            s += b % 10;
            b /= 10;
        }
        printf("Zbir cifara je: %d", s);
    }
    return 0;
}
```

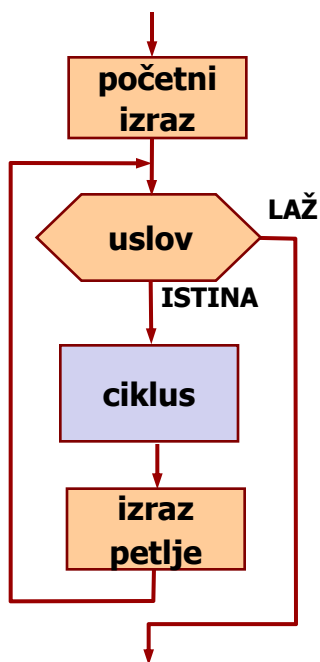
```
Unesite broj: 376
Zbir cifara je: 16
```

# Iterativne naredbe u jeziku C

## Naredba for

➤ Omogućava realizaciju petlje sa izlazom na vrhu i brojačke petlje.

➤ semantika



1. Izvrši početni izraz
2. Ako je uslov istinit tada:
  - 2.1. izvrši ciklus
  - 2.2. izvrši izraz petlje
  - 2.3. skok na korak 2.
3. Inače (ako uslov nije istinit), prekini izvršavanje petlje

Ekvivalentno sljedećoj while petlji:

```
pocetni_izraz;  
while (uslov)  
{  
    ciklus; izraz_petlje;  
}
```

➤ sintaksa

```
for (poc_izraz; uslov; izraz_petlje)  
    naredba;
```

```
for (poc_izraz; uslov; izraz_petlje)  
{  
    naredba1;  
    ...  
    naredbaN;  
}
```

# Iterativne naredbe u jeziku C

## Naredba for

### ➤ Pragmatika

C nije pozicioni jezik pa su ravnopravni različiti načini formatiranja

```
for (poc_izr; uslov; izr_petlje) n;
```



```
for (poc_izr; uslov; izr_petlje)
    n;
```



```
for (poc_izr; uslov; izr_petlje)
{
    n;
}
```



```
for (poc_izr; uslov; izr_petlje) {
    n;
}
```

### pragmatika početnog izraza

- **Početni izraz nije obavezan i može da se izostavi**  
for ( ; uslov; izraz\_petlje) n; ✓
- **Početni izraz tipično služi za inicijalizaciju kontrolne promjenljive**  
for ( i=0; i<n; i++ ) n; ✓
- **Početni izraz može biti složen (koristiti operator ,)**  
for ( i=0, j=k; i<n; i++ ) n; ✓
- **Promjenljive definisane u okviru početnog izraza, dostupne su samo unutar ciklusa i nisu dostupne izvan petlje**  
for (int i=2; i<n; i++) n;  
i++; ?

# Iterativne naredbe u jeziku C

## Naredba for

### ➤ Pragmatika

#### pragmatika izraza petlje

- Izraz petlje nije obavezan i može da se izostavi

`for ( poc_izraz; uslov; ) n; ✓`

`for ( ; uslov; ) n; ✓`

- Izraz petlje može biti jednostavan (tipično inkrementovanje/dekrementovanje)

`for ( i=0; i<n; i++ ) n; ✓`

- Izraz petlje može biti složen

`for ( i=2, f=1; i<=n; f*=i++ ) ; ✓`

**Prazna naredba ≡ Prazan ciklus !**

- Izraz petlje može biti složen

`for ( i=0, j=10; i<=j; i++, j-- ) ; ✓`

#### pragmatika uslova petlje

- I uslov petlje može da se izostavi

`for ( poc_izr; ; izr_petlje) n; ✓`

`for ( ; ; izr_petlje) n; ✓`

`for ( ; ; ) n; ✓`

`for ( ; ; ) ;`

**Izostavljanje uslova petlje ekvivalentno je istinitom uslovu – beskonačna petlja!**

`for ( ; ; ) n; ⇔ for ( ; 1 ; ) n;`

# Iterativne naredbe u jeziku C

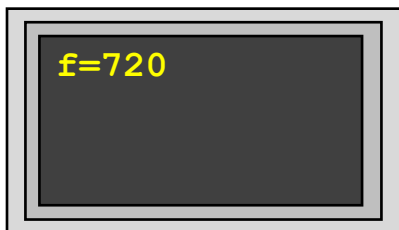
## Naredba for

### ➤ Pragmatika

#### pragmatika ciklusa

- Ciklus može biti i prazan, ponekad i greškom

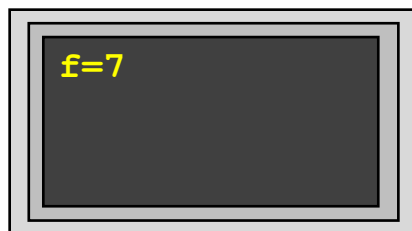
```
for ( i=2, f=1; i<=6; f*=i++ ) ;  
printf("f=%d", f);
```



A terminal window with a dark background and a light border. The text `f=720` is displayed in yellow.

Prazna naredba =  
Prazan ciklus !

```
for ( i=2, f=1; i<=6; i++ ) ;  
    f*=i;  
printf("f=%d", f);
```



A terminal window with a dark background and a light border. The text `f=7` is displayed in yellow.

- Naredba for omogućava pisanje "kompaktnijeg" koda u odnosu na naredbu while

```
s=0;  
while (b)  
{  
    s += b % 10;  
    b /= 10;  
}  
printf("Zbir cifara je: %d", s);
```

```
=====  
for (s=0; b; b/=10) s+=b%10;  
printf("Zbir cifara je: %d", s);
```

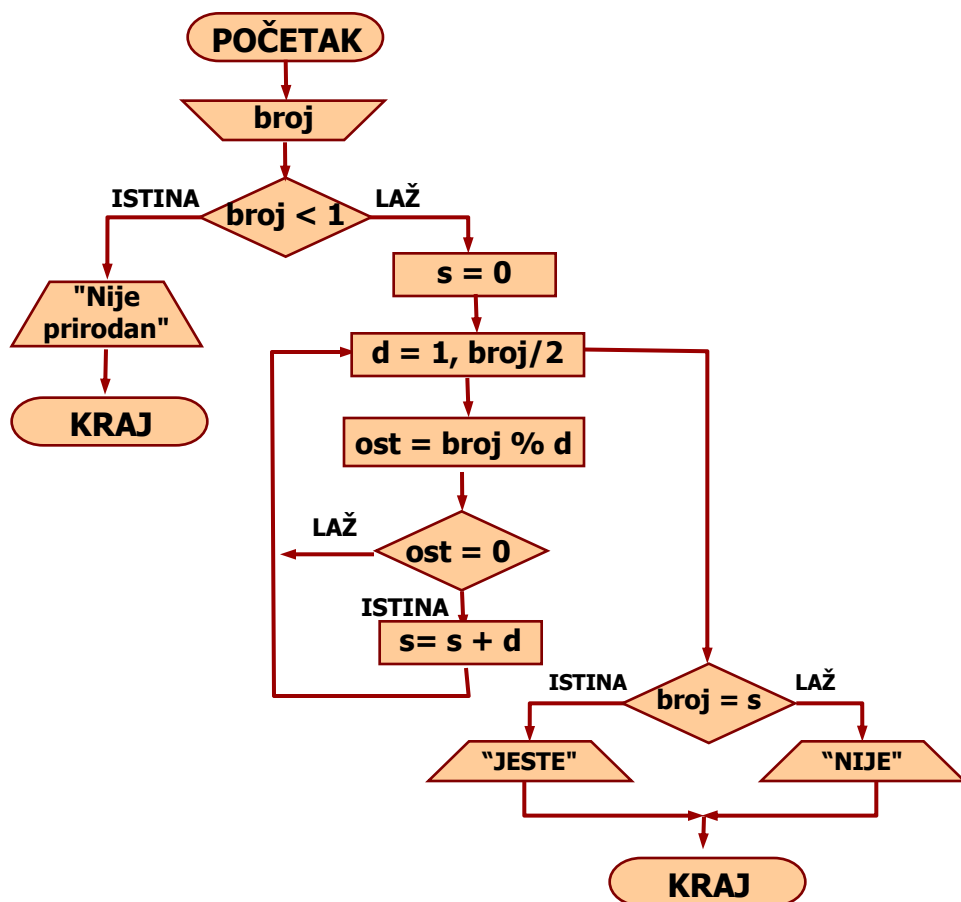
```
=====  
for (s=0; b; s+=b%10, b/=10);  
printf("Zbir cifara je: %d", s);
```

# Iterativne naredbe u jeziku C

## Naredba for

### Primjer:

Program koji provjerava da li je učitani prirodni broj savršen.



```
#include <stdio.h>
int main()
{
    int broj, d, s;
    printf("Unesite prirodan broj:");
    scanf("%d", &broj);
    if (broj<1)
        printf("Nije prirodan!");
    else
    {
        for ( s=1, d=2 ; d<=broj/2 ; d++ )
            s += broj%d ? 0 : d;
        printf("%d %s savrsen", broj,
            (broj==s) ? "je" : "nije" );
    }
    return 0;
}
```

Unesite prirodan broj:28  
28 je savrsen

# Iterativne naredbe u jeziku C

## Naredba do .. while

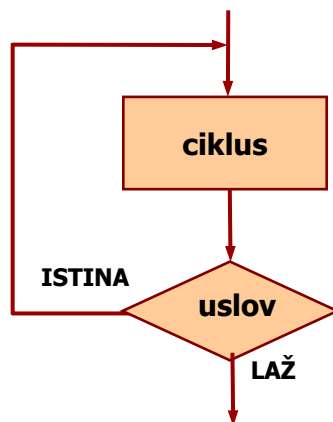
➤ Omogućava realizaciju petlje sa izlazom na dnu.

➤ sintaksa

```
do naredba while (uslov);
```

```
do
{
    naredba1;
    naredba2;
    ...
    naredbaN;
} while (uslov);
```

➤ semantika



1. Izvršava se ciklus
2. Izračunava se vrijednost uslova
3. Ako je uslov istinit, skok na korak 1.
4. Inače (ako uslov nije istinit, prekini izvršavanje petlje)

**Ciklus će se ponavljati  
sve dok je uslov istinit  
(mora bar jednom!)**



# Iterativne naredbe u jeziku C

## Naredba do .. while

### ➤ Pragmatika

C nije pozicioni jezik pa su ravnopravni različiti načini formatiranja

```
do
    n;
while (u);
```

⇔

```
do n; while (u);
```

```
do
{
    n1;
    ...
    nN;
} while (u);
```

⇔

```
do {
    n1;
    ...
    nN;
} while (u);
```

**uslov** je logički izraz,  
čijim se sračunavanjem dobija istinitosna vrijednost

- **Uslov ne može da se izostavi**

do n; while (); ❌

- **Uslov ne mora nužno da bude logički izraz**

do n; while (i);

"i" je izraz čijim se sračunavanjem dobija neka vrijednost koja se interpretira kao istinitosna – sve dok je  $i \neq 0$  ponavljaće se ciklus

- **Uslov ne mora nužno da bude logički izraz**

do n; while (i=2); ?

do n; while (i==2); ✓

do n; while (1);

do n; while (0); ?

Često se greškom umjesto operatora poređenja koristi operator dodjele pa je u gornjem slučaju ( $i=2$ ) uslov uvijek tačan – **beskonačna petlja**

# Iterativne naredbe u jeziku C

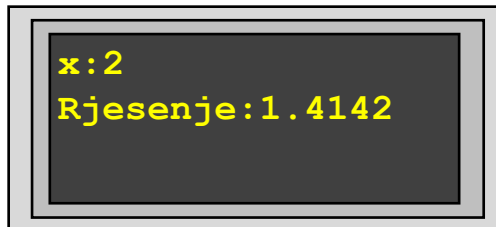
## Naredba do .. while

### Primjer:

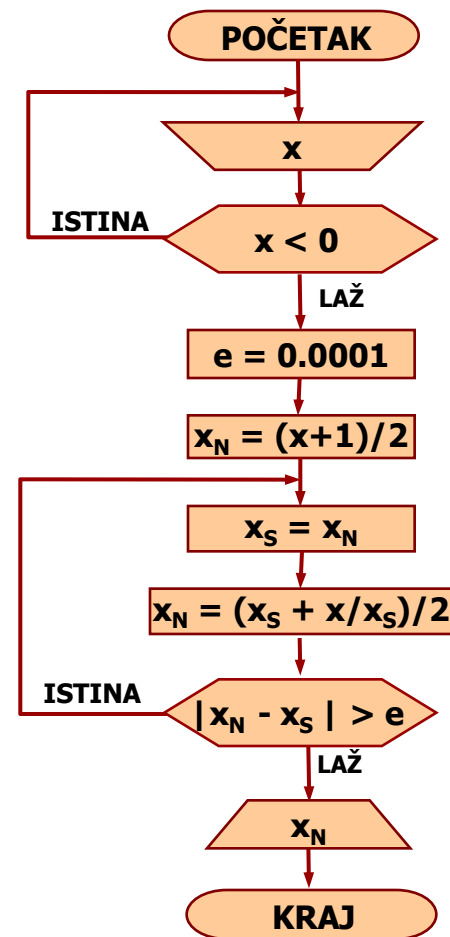
Program koji računa drugi korijen iz pozitivnog realnog broja X na četiri decimale...

```
#include <stdio.h>
#include <math.h>
#define EPS 0.0001
int main()
{
    float x, xs, xn;
    printf("x:"); scanf("%f", &x);
    if (x<0) printf("Nije pozitivan!");
    else
    {
        xn=(x+1)/2;
        do
        {
            xs=xn;
            xn=(xs+x/xs)/2;
        } while (fabs(xn-xs)>EPS);
        printf("Rjesenje: %.4f", xn);
    }
    return 0;
}
```

$$x_0 = \frac{X+1}{2}$$
$$x_{n+1} = \frac{x_n + \frac{X}{x_n}}{2}, \quad n = 0, 1, 2, \dots$$

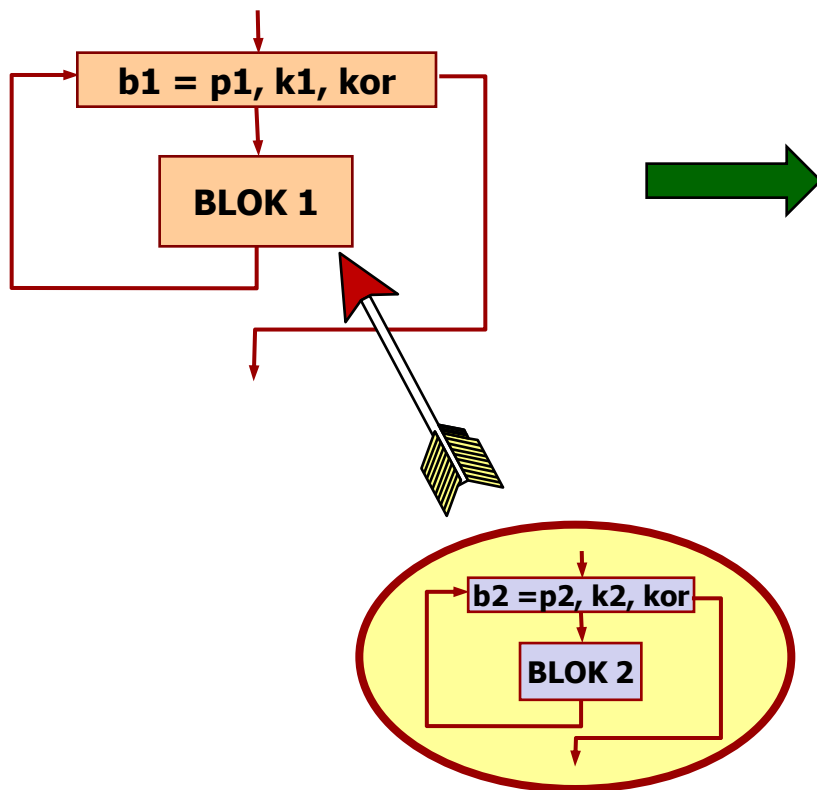


```
x:2
Rjesenje:1.4142
```

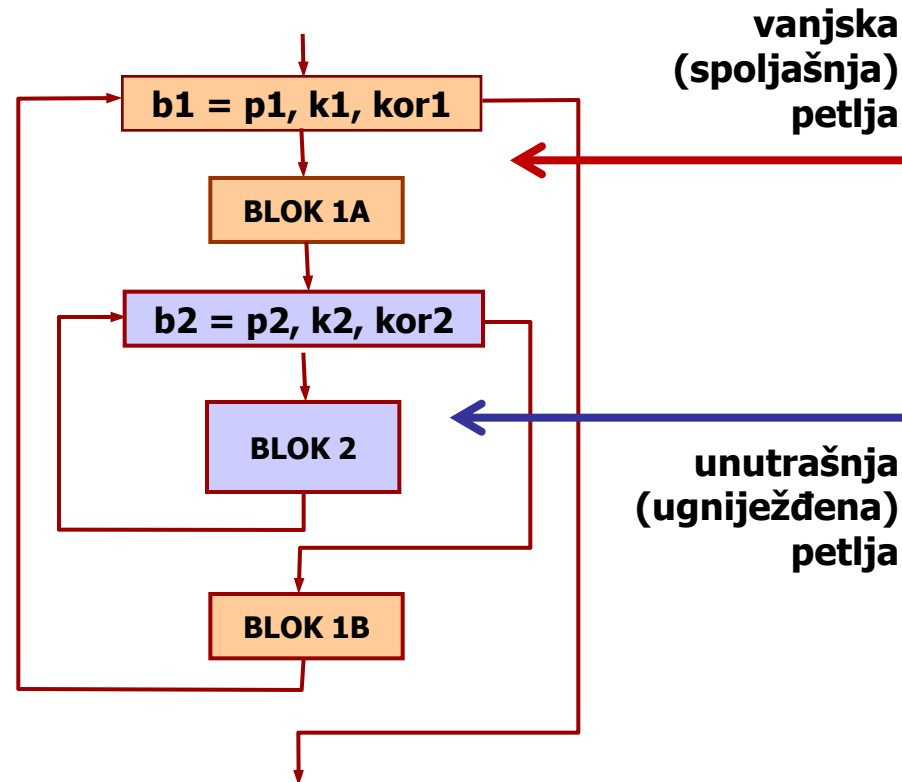


# Ugniježdene petlje (petlja u petlji)

## Ugniježdjena petlja / "Petlja u petlji"



Cjelokupna petlja može da se posmatra kao jedna naredba!



Unutrašnja petlja predstavlja dio ciklusa spoljašnje petlje.

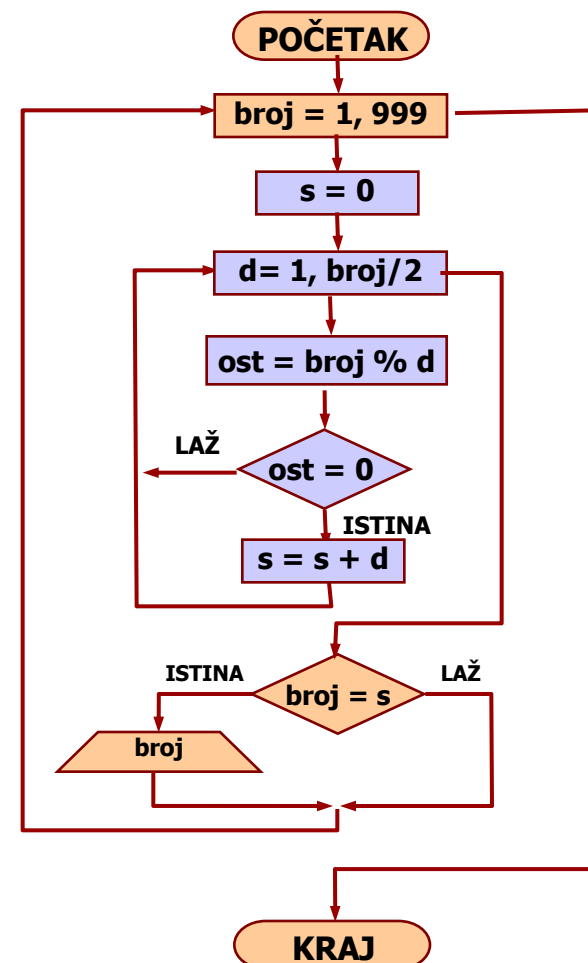
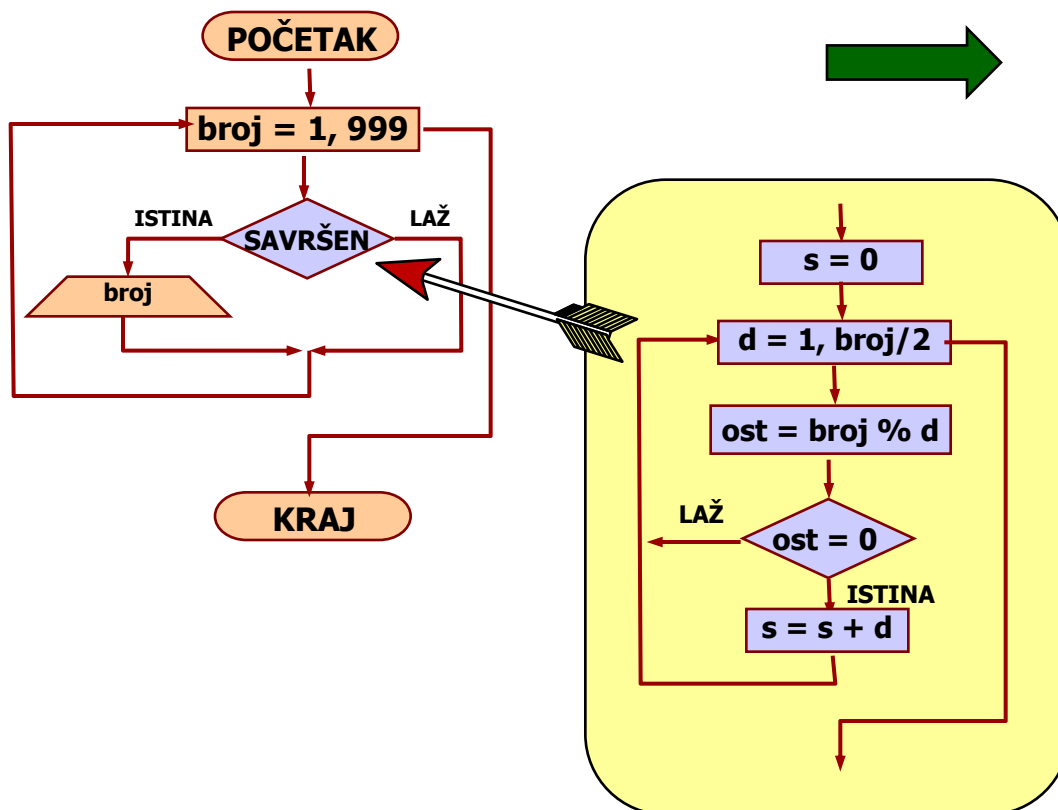
U svakoj iteraciji spoljašnje petlje, unutrašnja petlja se u potpunosti izvrši

# Ugniježdene petlje (petlja u petlji)

## Ugniježdene petlja / "Petlja u petlji"

### Primjer:

Program koji ispisuje sve savršene prirodne brojeve manje od 1000.



# Ugniježdene petlje (petlja u petlji)

## Ugniježdjena petlja / "Petlja u petlji"

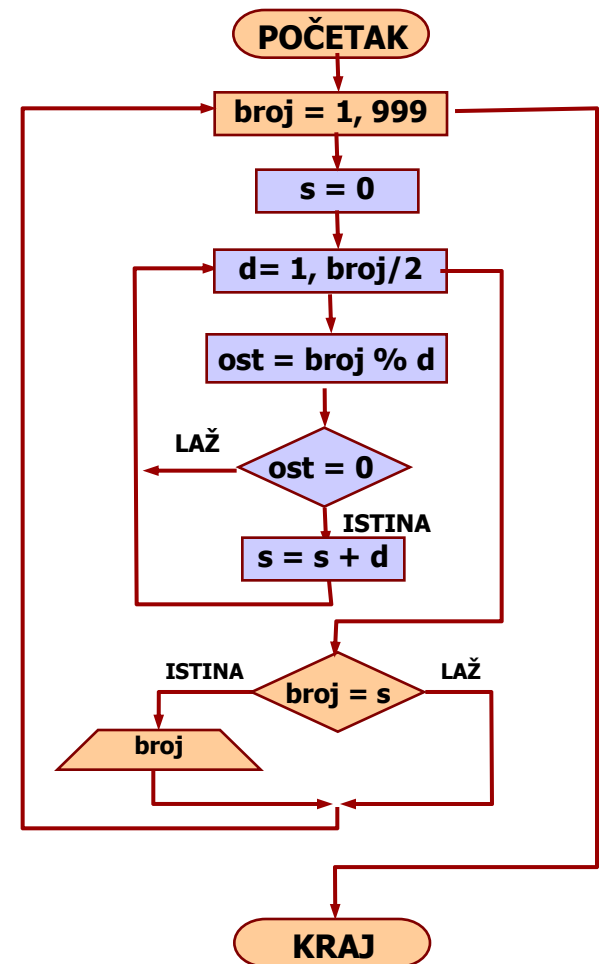
### Primjer:

Program koji ispisuje sve savršene prirodne brojeve manje od 1000.

```
#include <stdio.h>
int main()
{
    int broj, d, s;
    for ( broj=1 ; broj<1000 ; broj++ )
    {
        for ( s=1, d=2 ; d<=broj/2 ; d++ )
            s += (broj%d) ? 0 : d;
        if (broj==s)
            printf(" %d", broj);
    }
    return 0;
}
```



1 6 28 496





# Ugniježdene petlje (petlja u petlji)

## Ugniježdene petlja / "Petlja u petlji"

### Primjer:

Program koji ispisuje prvih n redova dvodimenzionalne strukture

```
*      1
**     12
***    123
:      :
:      :
```

```
      1
     121
    12321
   1234321
  :
  :
```

```
#include <stdio.h>
int main()
{
    int n, red, broj;
    do
    { printf("n=? "); scanf("%d",&n); }
    while (n<1 || n>20);
    for ( red=1 ; red<=n ; red++ )
    {
        for ( broj=1 ; broj<=red ; broj++ )
            printf("***");
        printf("\n");
    }
    return 0;
}
```

```
for ( red=1 ; red<=n ; red++ )
{
    for ( broj=1 ; broj<=n-red ; broj++ )
        printf(" ");
    for ( broj=1 ; broj<=2*red-1 ; broj++ )
        printf("%d", broj%10);
    for ( broj=2*red-2 ; broj>=1 ; broj-- )
        printf("%d", broj%10);
    printf("\n");
}
```



# Naredbe za nasilnu kontrolu toka

---

## Naredbe za nasilnu kontrolu toka

- Podrazumijevano, naredbe se izvršavaju sukcesivno (sekvencijalno) – jedna iza druge
- Podrazumijevani tok može da se promijeni naredbama za nasilnu kontrolu toka
- Nasilna kontrola toka podrazumijeva “skokove”, npr. nasilni izlazak iz petlje
- Naredbe za nasilnu promjenu toka u jeziku C:
  - **break**
  - **continue**
  - **goto**
  - **return**

## Naredba **return**

- **Dejstvo: izlazak iz funkcije**
  - prekida izvršavanje date funkcije (ignorišu se svi iskazi iza return)
  - ako se nalazimo u main funkciji – prekid programa
- **Omogućava da funkcija vrati neku vrijednost**
  - opšti oblik  
**return vrijednost;**  
ili  
**return izraz;**



# Naredbe za nasilnu kontrolu toka


## Naredba `break`

### ➤ Dejstvo:

- u naredbi `switch`:  
nasilni prekid izvršavanja naredbe `switch`  
prelazak na prvu naredbu iza naredbe `switch`
- u petljama:  
nasilni prekid izvršavanja petlje  
prelazak na prvi iskaz iza petlje
- u ostalim slučajevima:  
nema dejstva

### Npr.

```
while (1)
{
    // ...
    if (uslov) break;
    // ...
}
```



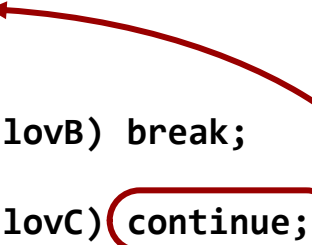
## Naredba `continue`

### ➤ Dejstvo:

- u petljama:  
završava trenutni ciklus (ignoriše sve naredbe do kraja ciklusa) i skače na provjeru uslova
- u ostalim slučajevima:  
nema dejstva

### Npr.

```
while (1)
{
    // ...
    if (uslovB) break;
    // ...
    if (uslovC) continue;
    // ...
}
```





# Naredbe za nasilnu kontrolu toka

## Naredbe break i continue

### ➤ Pragmatika

**U slučaju ugniježđenih petlji, naredbe break i continue imaju dejstvo samo na unutrašnju petlju.**

**Npr.**

```
for (red=1; red<5; red++)
{
    for (kol=1; kol<5; kol++)
    {
        if (kol>red) break;
        printf("%d ", kol);
    }
    printf("\n");
}
```

```
1
1 2
1 2 3
1 2 3 4
```

**Petlje koje sadrže naredbe break i continue, mogu da se restrukturisu tako da ekvivalentne petlje ne sadrže naredbe break i continue**

**Npr.**

```
while (A)
{
    if (B) break;
    C;
}

⇔

while (A && !B)
{
    C;
}
```

**Npr.**

```
for( ; A; )
{
    if (B)
        continue;
    C;
}

⇔

for ( ; A; )
{
    if (!B)
        C;
}

⇕

for ( ; A && !B; ) C;
```

# Naredbe za nasilnu kontrolu toka

## Naredba goto

➤ Omogućava безусловni skok na neku označenu naredbu u okviru iste funkcije

➤ Opšti oblik:

**goto labela;**

Npr.

```
#include <stdio.h>
int main()
{
    int broj;
    lab1: printf("Unesite prirodan broj: ");
    scanf("%d",&broj);
    if (broj<1)
    {
        printf("Nije prirodan!\n");
        goto kraj;
    }
    printf ("%d %s paran\n", broj, (broj%2)? "nije" : "je");
    goto lab1;
    kraj: printf("KRAJ...");
    return 0;
}
```

```
graph TD
    Lab1((lab1:)) --> Sscanf[scanf]
    Sscanf --> If[if]
    If --> PrintNije[printf "Nije prirodan!"]
    PrintNije --> GotoKraj((goto kraj;))
    GotoKraj --> Kraj((kraj:))
    Kraj --> PrintKraj[printf "KRAJ..."]
    PrintKraj --> Return[return 0]
    GotoLab1((goto lab1;)) --> Lab1
```

# Naredbe za nasilnu kontrolu toka

## Naredba goto

### ➤ Pragmatika

**Posljedica:  
Naredba goto nije potrebna!**

### VELIKA STRUKTURNA TEOREMA / VELIKA TEOREMA STRUKTURNOG PROGRAMIRANJA

“Svaka algoritamska struktura može da se reprezentuje sekvencom, grananjem i nekom vrstom petlje”  
Korado Bem i Đuzepe Jakopini (Corrado Böhm & Giuseppe Jacopini), 1966.

### Npr.

```
#include <stdio.h>
int main()
{
    int broj;
lab1: printf("Unesite prirodan broj: ");
    scanf("%d", &broj);
    if (broj<1)
    {
        printf("Nije prirodan!\n"); goto kraj;
    }
    printf ("%d %s paran\n", broj,
            (broj%2)? "nije" : "je");
    goto lab1;
kraj: printf("KRAJ...");
    return 0;
}
```



```
#include <stdio.h>
int main()
{
    int broj;
    do
    {
        printf("Unesite prirodan broj: ");
        scanf("%d", &broj);
        if (broj<1)
            printf("Nije prirodan!\n");
        else
            printf ("%d %s paran\n", broj,
                    (broj%2)? "nije" : "je");
    } while (broj>=1);
    printf("KRAJ...");
    return 0;
}
```