

IMPERATIVNI PROGRAMSKI JEZICI

Razvoj imperativnih programskih jezika

Osnovne programske (upravljačke) strukture

Naredbe prekida

Imperativna paradigma programiranja

- Program predstavlja skup naredbi koje menjaju njegovo stanje
- Naredbama se opisuje algoritam za rešavanje problema
- Poreklo imena:
 - Imperativ – glagolski oblik kojim se iskazuje naredjenje, poznat kao „zapovedni način“

Naredbe u imperativnim jezicima

- Naredbe obrade (obično izrazi dodele)
- Upravljačke strukture

Upravljačke strukture

- **Upravljačka struktura** je instrukcija programa koja određuje redosled izvršavanja drugih instrukcija programa

Razvoj imperativnih programskih jezika i upravljačkih struktura

- 1957. definisan prvi imperativni programski jezik
FORTRAN
 - Upravljačke strukture su izvedene iz osnovnih mašinskih naredbi.
 - Često u upotrebi naredba bezuslovnog skoka (goto)

Razvoj imperativnih programskih jezika i upravljačkih struktura

- Početkom sedamdesetih Wirt definiše metodologiju strukturnog programiranja i programski jezik **Pascal** sa skupom upravljačkih struktura kojima se implementiraju osnovne algoritamske strukture.
- **Strukturno programiranje – programske strukture se mogu ugnježdavati, ne i preklapati.**
- Ovaj koncept je široko prihvaćen tako da danas svi proceduralni i objektno-orijentisani programski jezici podržavaju Wirtov standardan skup upravljačkih struktura.
- 1972. se pojavljuje **C** koji se smatra najkorišćenijim imperativnim jezikom

Osnovne upravljačke strukture

- Osnovne upravljačke strukture su: sekvenca, selekcija i programska petlja.
 - **Sekvenca** predstavlja niz instrukcija u programu koje se izvršavaju onim redosledom kako su zapisane.
 - **Selekcija** na osnovu ispunjenosti ili neispunjenosti određenog uslova određuje koje će deo koda biti izvršen,
 - **Petlja** definiše ponavljanje izvršenja određenog skupa instrukcija.

Sekvenca naredbi ili blok

- Pascal:

```
begin
```

```
    naredba1;
```

```
    naredba2;
```

```
    . . .
```

```
    naredbaN;
```

```
end
```

- C#:

```
{
```

```
    naredba1;
```

```
    naredba2;
```

```
    . . .
```

```
    naredbaN;
```

```
}
```


Opseg važenja imena

- Blokovi se često kotiste i za ograničavanje opsega važenja simboličkih imena
 - Simboličko ime definisano u okviru jednog bloka je lokalna za taj blok, a globalna za unutrašnje blokove (blokove definisane unutar njega)

Opseg važenja imena - primer

```

A: { int A; ...
    B: {float B; ... }
    C: {double C; ...
        D: {char D; ...}
        E: {int E; ...}
    }
    E: {float E; ... }
    F: {float F; ...
        G: {double G; ... }
    }
}

```

Tabela 5.1

Promenljiva	Blok						
	A	B	C	D	E	F	G
A	L	G	G	G	G	G	G
B		L					
C			L	G	G		
D				L			
E					L		
F						L	G
G							L

Problem konflikta imena

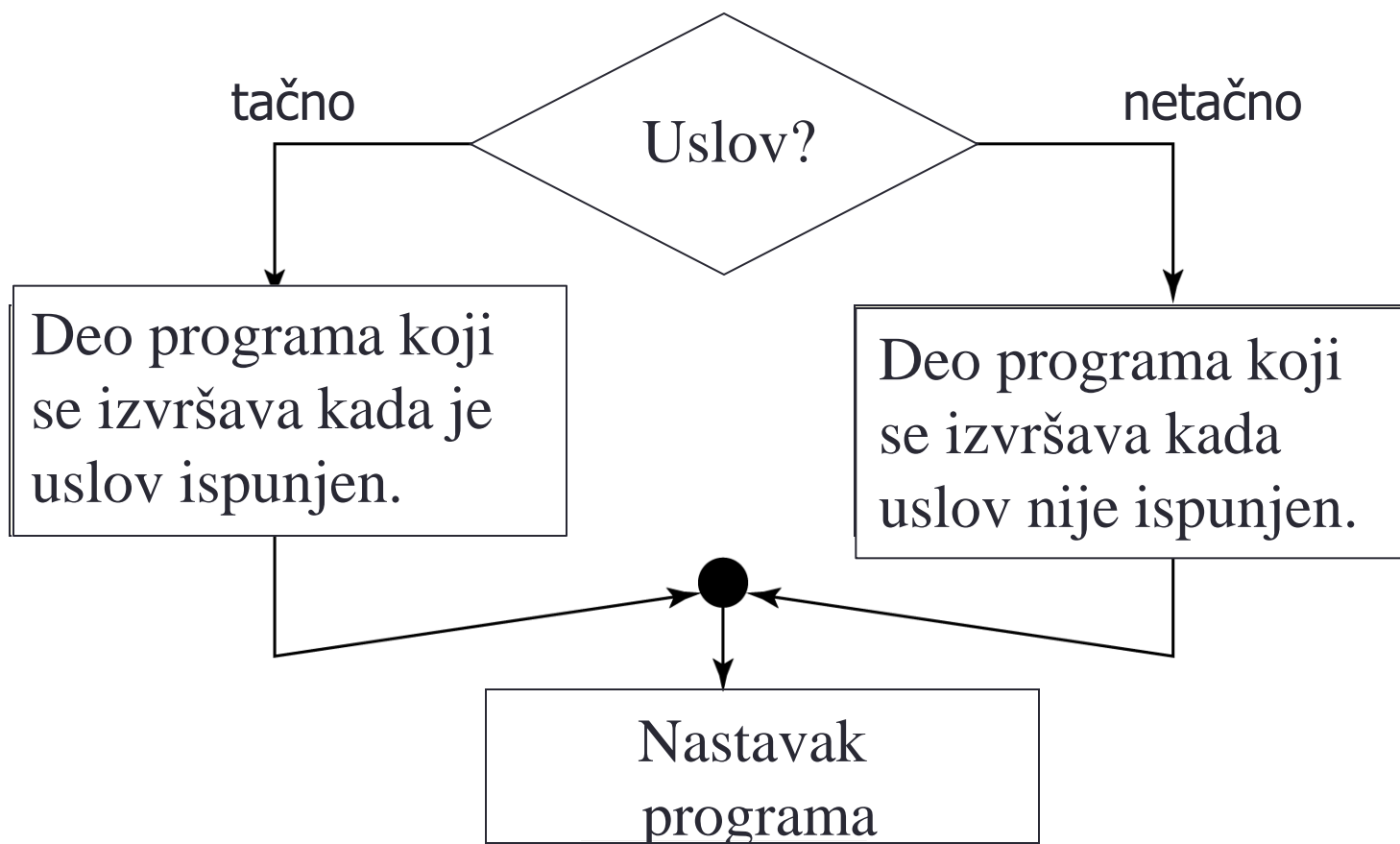
- Nastaje kada se isto ime deklariše više puta:

```
A: { int A; ...  
    B: {float A; ... A ... }  
}
```

- Razrešenje:
 - C, C++: Važi poslednja definicija imena (u bloku najbližem tački korišćenja)
 - C#, JAVA: Kompajler prijavljuje grešku

Struktura selekcije (grananja)

- Naredba selekcije (**if naredba**) omogućava da se izvrši jedan ili drugi deo programa zavisno od toga da li je neki uslov ispunjen ili ne.



Primer strukture grananja

Pascal:

```
if a < b then  
    min := a  
else  
    min := b;
```

C-like jezici (C, C++, Java, C#):

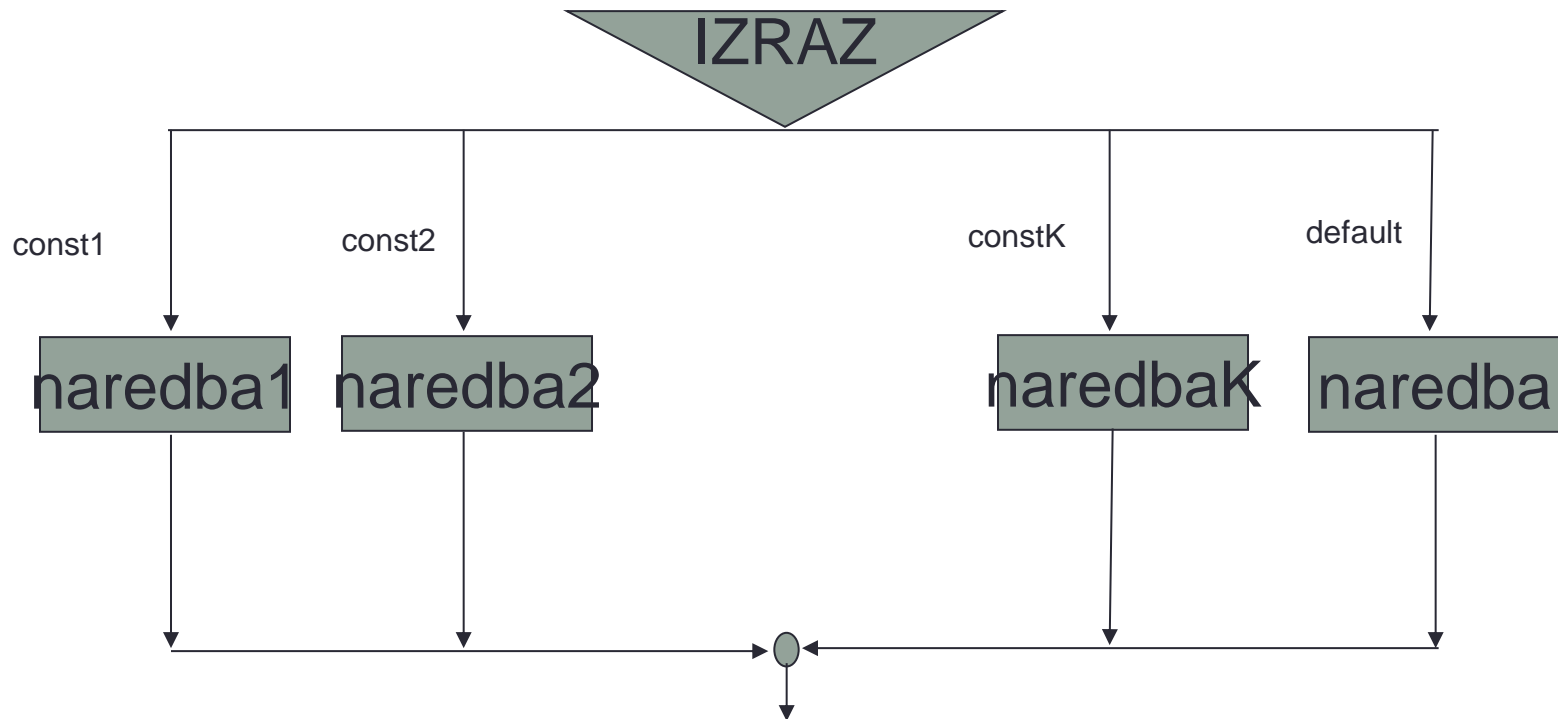
```
if (a < b)  
    min = a;  
else  
    min = b;
```

Problem ugnježdavanja struktura grananja

```
if (B1) if (B2) Sa else Sb
```

U ovoj naredbi nije potpuno jasno da li se Sb izvršava kada je $B1 = \text{false}$ ili kada je $B1 = \text{true}$ i $B2 = \text{false}$ (što je u ovom slučaju tačno).

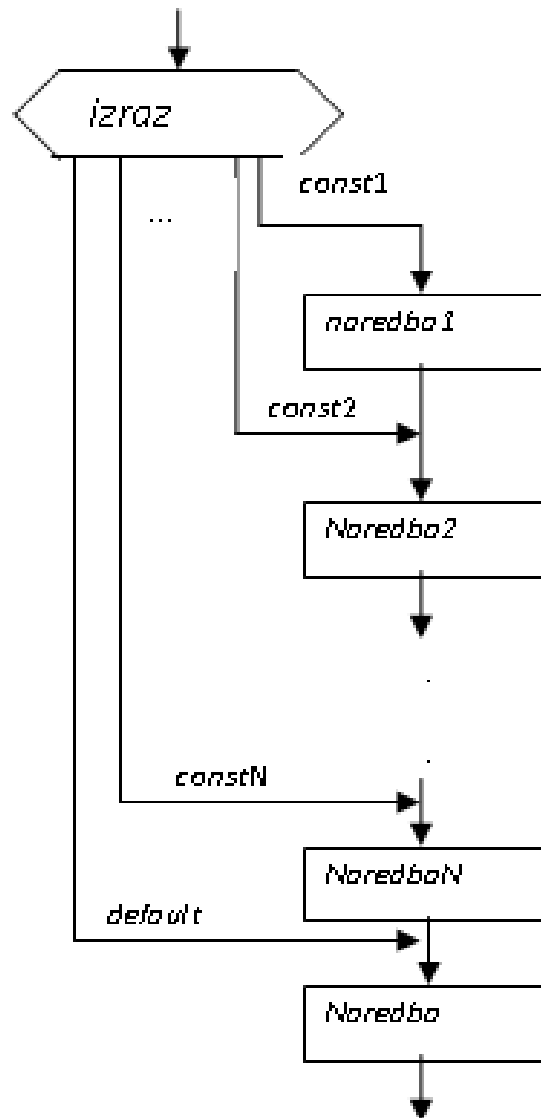
Višestruko grananje (Struktura „češlja“)



Pascal:

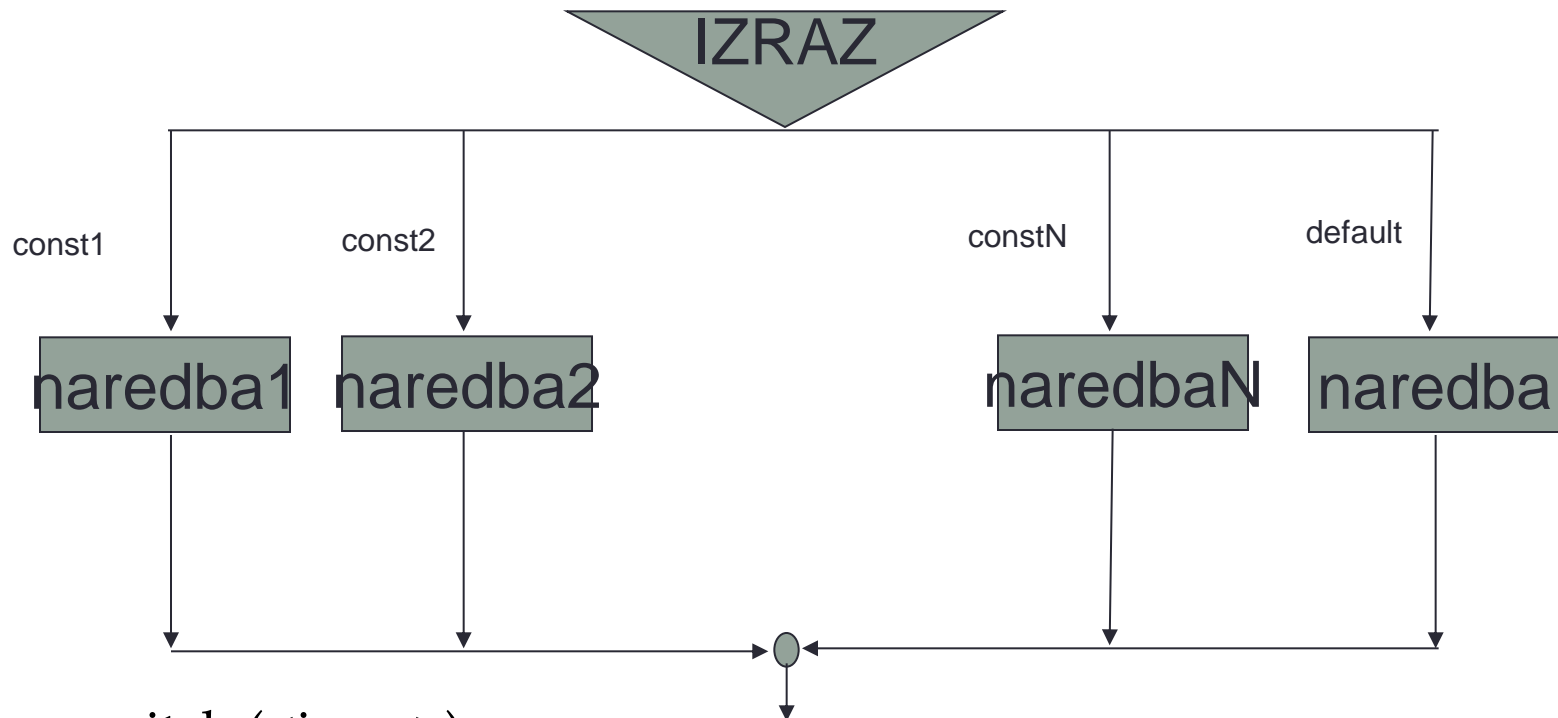
```
case izraz of
  lista_konstanti_1: naredba_1;
  lista_konstanti_2: naredba_2;
  .....
  lista_konstanti_k: naredba_k;
end
```

Višestruko grananje u programskom jeziku C



```
switch (<izraz>)
{
    case <const1>: <naredba1>
    case <const2>: <naredba2>
    ...
    case <constN>: <naredbaN>
    [default: <naredba>]
}
```


Struktura „češlja“ u programskom jeziku C



switch (<izraz>)

{

case <const1>: <naredba1> break;

case <const2>: <naredba2> break;

...

case <constN>: <naredbaN> break;

[default: <naredba>]

}

Višestruko grananje u programskim jezicima C# i Java

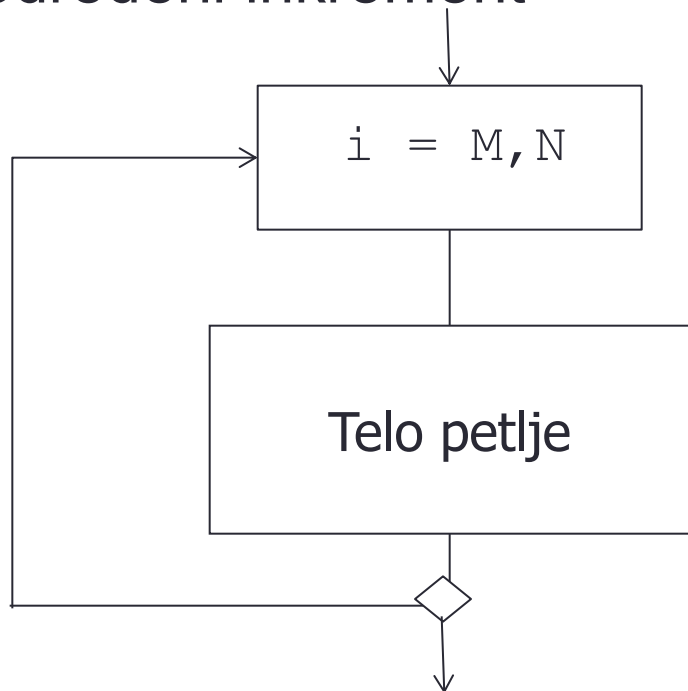
- U programskom jeziku Java *switch* struktura ima potpuno identičnu semantiku kao u programskom jeziku C.
- U programskom jeziku C# *switch* struktura je namenjena kreiranju strukture tipa “češlja”
 - *break* naredba je obavezna nakon svake *case* klauzule pa čak i nakon klauzule *default*.

Programske petlje

- Omogućavaju višestruko izvršavanje istog bloka naredbi programa
- Vrste programskih petlji
 - Brojačke petlje - petlje sa unapred poznatim brojem ponavljanja
 - Petlje sa unapred nepoznatim brojem prolaza - petlje koje se ponavljaju dok neki uslov jeste/nije ispunjen
 - Petlje sa uslovom ponavljanja na pocetku
 - Petlje sa uslovom ponavljanja na kraju

Brojačke petlje

- Koristi specijalnu promenljivu nazvanu brojač petlje, koja pri svakom izvršenju naredbi u petlji menja svoju vrednost počev od startne vrednosti, do konačne vrednosti za određeni inkrement



Pascal:

```
for i:=1 to n do  
  s := s*a;
```

C:

```
for (i=1; i<=n; i++)  
  s = s*a;
```

Obilazak kolekcija

- Često se brojačka petlja koristi i za obilazak kolekcija podataka.
- Primer:

```
int[] a = { 1, 2, 6, 4 };  
for (int i=0; i<4; i++)  
    ...a[i]...
```

Petlje namenjene obilasku kolekcija

- Java:

`for (<var> : <collection>) <statement>`

- Java primer:

```
int[] a = { 1, 2, 6, 4 };  
for(int element : a)  
    ...element...
```

- C#:

`foreach (<var> in <collection>) <statement>`

- C# primer:

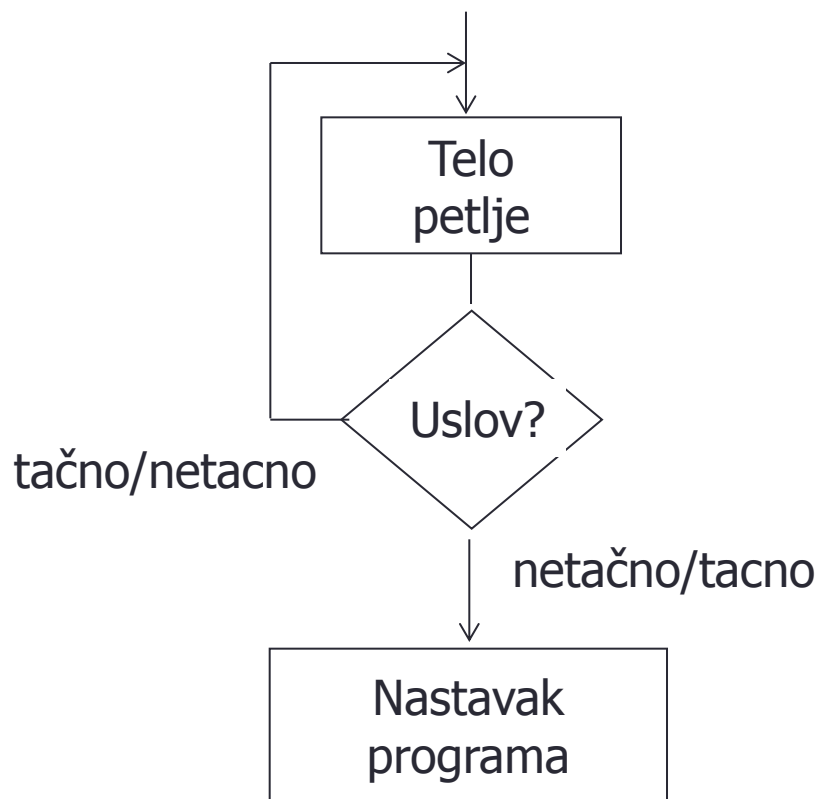
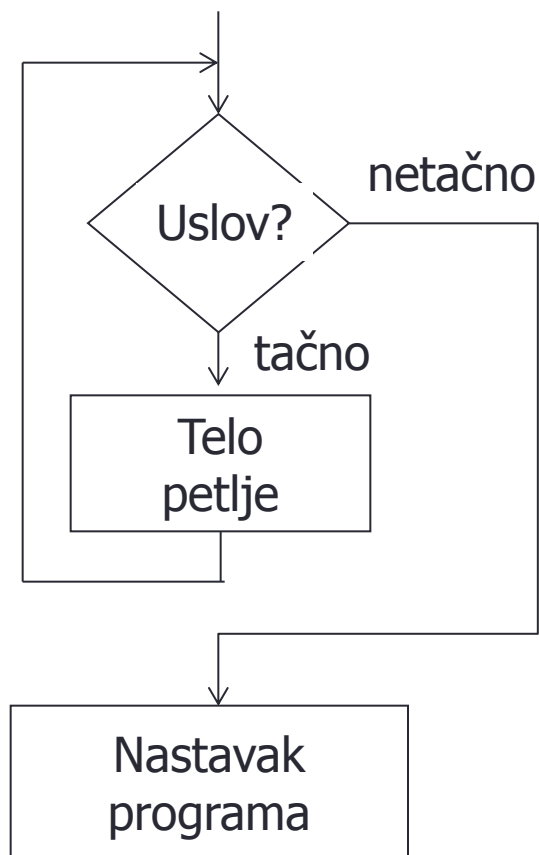
```
int[] a = { 1, 2, 6, 4 };  
foreach(int element in a)  
    ...element...
```

Ograničenja za upotrebu petlji za rad sa kolekcijama

- Java:
 - Kolekcija može da bude
 - polje ili
 - klasa koja implementira interfejs `Collection`.
- C#:
 - Kolekcija može da bude
 - polje ili
 - klasa koja implementira jedan od interfejsa
 - `System.Collections.IEnumerable`
 - `System.Collections.IEnumerator`
 - `System.Collections.Generic.IEnumerable<T>`
 - `System.Collections.Generic.IEnumerator<T>`

Petlje sa nepoznatim brojem prolaza

- Uslov ponavljanja petlje se može nalaziti na početku ili na kraju petlje, tj. ispituje se pre/posle svakog prolaska kroz petlju.



Primer petlje sa nepoznatim brojem prolaza u C-like jezicima

```
// odredjivanje broja cifara u celom broju N
// pomocu petlje sa uslovom ponavljanja na pocetku
brCifara = 1;
N = N / 10;
while ( N != 0 ) {
    N /= 10;
    brCifara++;
}
```

```
// odredjivanje broja cifara u celom broju N
// pomocu petlje sa uslovom ponavljanja na kraju
brCifara = 0;
do {
    N /= 10;
    brCifara++;
} while ( N != 0 );
```

Naredbe prekida

- Za prekidanje tekuće iteracije petlje:

`continue;`

- Java ima i mogućnost prekidanja izvršenja tekuće iteracije obeležene petlje :

`continue labela;`

Naredbe prekida

- Za prekidanje tekuće programske strukture:

break ;

- Java ima i mogućnost prekidanja izvršenja tekuće iteracije obeležene petlje :

break *labela* ;

Naredbe prekida

- Za prekidanje izvršenja tekućeg potprograma:

return;

ili

return *izraz*;

Naredbe prekida

- Za prekidanje tekuće strukture, niza ugnježdenih struktura, tekuće funkcije ili niza pozvanih funkcija:

throw izraz;