Osnovi programiranja

2024/25

Ponavljanje

Naredbe ponavljanja

- Ukoliko je naredbu potrebno izvršiti konačan i mali broj puta, problem je moguće razrešiti i korišćenjem linijskih struktura, tako što bi se naredba jednostavno ponovila određeni broj puta uzastopno.
- Može se desiti da je naredbu potrebno ponoviti veliki broj puta, a veoma često je taj broj promenljiv u zavisnosti od izvršenja ostatka programa.
- **Ciklične strukure (ciklusi ili petlje)** omogućavaju izvršavanje jedne ili više naredbi određeni broj puta, pri čemu broj ponavljanja može biti definisan prirodnim brojem ili uslovom koji određuje kada se ponavljanje prekida.

Ponavljanje naredbi - FOR

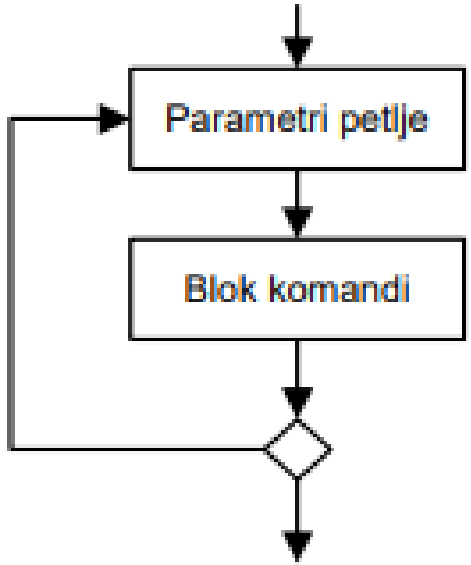
- Naredba **for** umogućava bezuslovno ponavljanje nekog dela programa određeni broj puta.
- **FOR** petlja se sastoji iz tri dela.
  - U okviru prvog dela može se definisati brojačka promenljiva (brojač) koja se u svakom prolasku kroz petlju (iteraciji) može menjati.
  - U okviru drugog dela može se definisati uslov izlaska iz petlje. To je logički izraz koji u slučaju da je netačan prekida izvršavanje petlje.
  - U okviru trećeg dela se zadaje promena brojača petlje nakon svake iteracije.

```
for ([opcioni_izraz_1];[opcioni_logicki_izraz];[opcioni_izraz_2]) <naredba>;
```

Ponavljanje naredbi - FOR

```
for ([opcioni_izraz_1];[opcioni_logicki_izraz];[opcioni_izraz_2]) <naredba>;
```

Nije obavezno definisati sve delove for petlje, ali iako neki deo ne postoji mora biti odvojen rezervisanim znakom ;.



Ponavljanje naredbi - FOR

```
for (i=1; i<=5; i++)
   printf("Zdravo");
```

```
Zdravo
Zdravo
Zdravo
Zdravo
Zdravo
```

```
for (i=0; i<5; i++)
    printf(“%d\n”,i);
```

```
0
1
2
3
4
```

Ponavljanje naredbi - FOR

```
for (i = 10; i > 0; i--) printf("%5d", i);
```

```
   10 9 8 7 6 5 4 3 2 1
```

Početna i krajnja vrednost brojačke promenljive mogu biti i izrazi:

```
for (i = -4+6; i <= 4*3-5; i++) printf("%5d", i);
```

```
   2 3 4 5 6 7
```

Ugnježdene petlje

- Ispis tablice množenja

```
for (x=1; x<=10; x++)
    for (y=1; y<=10; y++)
        printf("%d ",x*y);
```

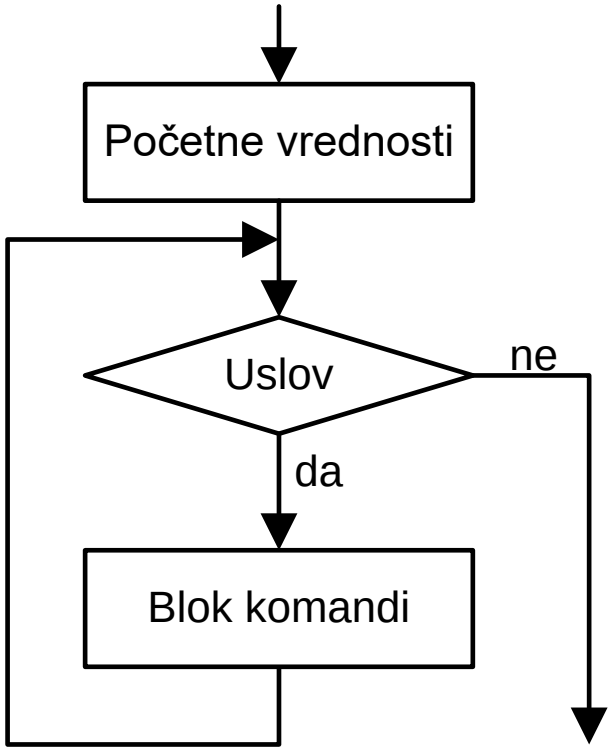
```
1  2  3  4  5  6  7  8  9  10
2  4  6  8  10  12  14  16  18  20
3  6  9  12  15  18  21  24  27  30
4  8  12  16  20  24  28  32  36  40
5  10  15  20  25  30  35  40  45  50
6  12  18  24  30  36  42  48  54  60
7  14  21  28  35  42  49  56  63  70
8  16  24  32  40  48  56  64  72  80
9  18 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

```
for (x=1; x<=10; x++)
{
    for (y=1; y<=10; y++)
        printf("%d ",x*y);
    printf("\n");
}
```

Ponavljanje naredbi - WHILE

- Osnovna karakteristika **while** petlje je da je to **petlja sa preduslovom**, što znači da se ispunjenost uslova proverava pre izvršavanja svake iteracije. Posledica toga je da je moguće da se naredbe unutar petlje ne izvrše ni jednom, ukoliko uslov u startu nije zadovoljen.

```
while (<logicki_izraz>) <naredba>;
```



Ponavljanje naredbi - WHILE

- Poređenje - želimo 5 puta da štampamo 'Zdravo!'

```
br = 0;
if (br < 5);
{
    printf(“Zdravo!”);
    br++;
}
```

```
Zdravo!
```

```
br = 0;
while (br < 5)
{
    print(“Zdravo!”)
    br++;
}
```

```
Zdravo!
Zdravo!
Zdravo!
Zdravo!
Zdravo!
```

Ponavljanje naredbi - WHILE

- Šta radi dati deo koda?

```
 n=5;
while (n > 0)
{
    printf("%d\n",n);
    n--;    // šta se dešava bez ovog reda?
}
printf("Gotovo!");
```

ili

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

Ponavljanje naredbi - WHILE

- **While** petlja mora biti tako napisana da garantuje da će u konačnom broju iteracija navedeni logički izraz postati netačan (**false**). Na taj način obezbeđuje se mehanizam izlaska iz petlje nakon konačnog broja iteracija.
- Ukoliko logički izraz nikada ne bi dobio vrednost **false** došlo bi do beskonačnog broja ponavljanja (tzv. **mrtva petlja**), odnosno do blokade izvršenja ostatka programa.

Primer 1

- Šta je rezultat datog koda?

```
#include <stdio.h>
main()
{
   int c;
   for (c = 0; c<255; c++)
     printf("%d -%c\n",c,c);
}
```

Primer 2

- Napisati program koji zadatom celom broju n uklanja sve nule sa desne strane. Na primer, ukoliko unesemo broj 21000, rezultat treba da bude 21.

```
#include <stdio.h>
main()
{
   int n;
   printf("Unesite broj n:\n");
   scanf("%d", &n);
   while(n % 10 == 0)
      n = n / 10;
   printf("Dobijeni broj je %d\n", n);
}
```

Primer 3

- Izračunati zbir unetih brojeva. Brojeve unositi dok se ne unese 0.

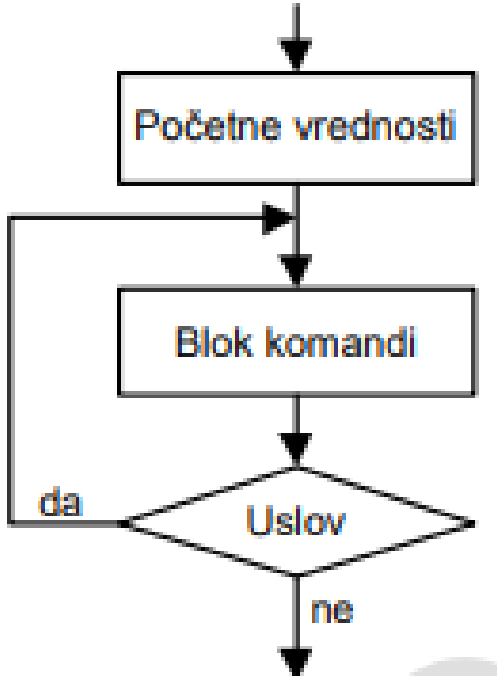
```
#include <stdio.h>
main()
{
    int x, s;
    printf(“Unesi broj:”);
    scanf("%d", &x);
    s=0;
    while (x != 0)
    {
        s = s + x;
        printf(“Unesi broj:”)
        scanf("%d", &x);
    }
    printf("Zbir unetih brojeva je %d", s)
}
```

```
Unesi broj: 3
Unesi broj: 5
Unesi broj: 4
Unesi broj: 7
Unesi broj: 0
Zbir unetih brojeva je  19
```

Ponavljanje naredbi – DO WHILE

- Za razliku od **while** petlje, naredba **do-while** omogućava realizaciju ciklusa sa **postuslovom**.

```
do
{
   <naredba_1>;
   [naredba_2];
   ...
   [naredba_n];
} while (<logicki_izraz>);
```



Primer 4

- Šta je rezultat datog koda?

```
int i = 2;
do
{
   printf("%5d", i);
   i = i + 2;
} while(i < 10);
```

```
2 4 6 8
```

Naredbe BREAK i CONTINUE

- Za prekid izvršavanje petlje, bez obzira na ispunjenost uslova koristi se komanda BREAK

```
for (i=1; i<10; i++)
{
    if (i == 3)
        break;
    printf("%d ", i);
}
```

```
1
2
```

- Komandom CONTINUE se prekida izvršavanje trenutne iteracije (preskače se deo koda) i prelazi se na sledeću iteraciju

```
for (i=1; i<10; i++)
{
    if (i == 3)
        continue;
    printf("%d ", i);
}
```