

Programiranje 1

Milena Vujošević Jančić

www.matf.bg.ac.rs/~milena

Naredbe i kontrola toka

Pregled

1 Naredbe

Pregled

- 1 Naredbe
 - Naredba izraza i složena naredba
 - Naredbe grananja
 - Petlje

Naredbe

- Osnovni elementi kojima se opisuju izračunavanja u programima su *naredbe*.
- Naredbe za kontrolu toka omogućavaju različite načine izvršavanja programa, u zavisnosti od vrednosti promenljivih.
- Naredbe za kontrolu toka uključuju naredbe grananja i petlje.
- U jeziku C postoji i naredba skoka, ali je nećemo koristiti
- Na osnovu teoreme o strukturnom programiranju, od naredbi za kontrolu toka dovoljna je naredba grananja (tj. naredba `if`) i jedna vrsta petlje (na primer, petlja `while`), ali se u programima često koriste i druge naredbe za kontrolu toka radi bolje čitljivosti kôda.

Naredba izraza

- Osnovni oblik naredbe koji se javlja u C programima je takozvana naredba izraza (ova vrsta naredbi obuhvata i naredbu dodele i naredbu poziva funkcije).
- Naime, svaki izraz završen karakterom ; je naredba.

```
3 + 4*5;
```

```
n = 3;
```

```
c++;
```

```
f();
```

Složene naredbe (blokovi)

- U nekim slučajevima potrebno je više različitih naredbi tretirati kao jednu jedinstvenu naredbu.
- Vitičaste zagrade { i } se koriste da grupišu naredbe u složene naredbe tj. blokove i takvi blokovi se mogu koristiti na svim mestima gde se mogu koristiti i pojedinačne naredbe.

Naredbe grananja

- Naredbe grananja (ili naredbe uslova), na osnovu vrednosti nekog izraza, određuju naredbu (ili grupu naredbi) koja će biti izvršena.

```
if (izraz)
    naredba1
else
    naredba2
```

- Naredba `naredba1` i `naredba2` su ili pojedinačne naredbe (kada se završavaju simbolom `;`) ili blokovi naredbi zapisani između vitičastih zagrada (iza kojih se ne piše simbol `;`).

- Deo naredbe `else` je opcioni, tj. može da postoji samo `if` grana.
- Izraz `izraz` predstavlja logički uslov

```
if (5 > 7)
```

```
    a = 1;
```

```
else
```

```
    a = 2;
```

- `if (7)`

```
    a = 1;
```

```
else
```

```
    a = 2;
```


- Na primer, `if (n != 0)` je ekvivalentno sa `if (n)`.
- ```
a = 3;
if (a = 0)
 printf("a je nula\n");
else
 printf("a nije nula\n");
```

# if-else višeznačnost

Na koje if se odnosi else?

```
if (izraz1)
 if (izraz2)
 naredba1
else
 naredba2
```

## if-else višeznačnost

```
if (izraz1) {
 if (izraz2)
 naredba1
} else

```

# Konstrukcija else-if

- Za višestruke odluke često se koristi konstrukcija sledećeg oblika:

```
if (izraz1)
 naredba1
else if (izraz2)
 naredba2
else if (izraz3)
 naredba3
else
 naredba4
```

# Konstrukcija else-if

```
if (a > 20)
 printf("A je vece od 20\n");
else if (a > 10)
 printf("A je vece od 10\n");
else if (a < -20)
 printf("A je manje od -20\n");
else if (a < -10)
 printf("A je manje od -10\n");
else
 printf("A pripada intervalu [-10, 10]\n");
```

# Naredba if-else i operator uslova

```
if (a > b)
 x = a;
else
 x = b;
```

```
x = (a > b) ? a : b;
```

# Naredba switch

- Naredba `switch` se koristi za višestruko odlučivanje i ima sledeći opšti oblik:

```
switch (izraz) {
 case konstantan_izraz1: naredbe1
 case konstantan_izraz2: naredbe2
 ...
 default: naredbe_n
}
```

# Petlja while

- Petlje (ciklusi ili repetitivne naredbe) uzrokuju da se određena naredba (ili grupa naredbi) izvršava više puta (sve dok je neki logički uslov ispunjen).

```
while(izraz)
 naredba
```

```
while (i < j)
 i++;
```

```
while (1)
 i++;
```



# Petlja for

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

```
izraz1;
while (izraz2) {
 naredba
 izraz3;
}
```

```
for(i = 0; i < n; i++)
 ...
```

# Petlja for

- Bilo koji od izraza `izraz1`, `izraz2`, `izraz3` može biti izostavljen, ali simboli `;` i tada moraju biti navedeni.
- izostavljen izraz `izraz2`, smatra se da je njegova istinitosna vrednost *tačno*.

```
for (;;)
```

```
...
```

# Petlja for

- Ako je potrebno da neki od izraza `izraz1`, `izraz2`, `izraz3` objedini više izraza, može se koristiti operator `,`

```
for (i = 0, j = 10; i < j; i++, j--)
 printf("i=%d, j=%d\t", i, j);
```

# Petlja for

- Ako je potrebno da neki od izraza `izraz1`, `izraz2`, `izraz3` objedini više izraza, može se koristiti operator `,`

```
for (i = 0, j = 10; i < j; i++, j--)
 printf("i=%d, j=%d\t", i, j);
```

`i=0, j=10`

# Petlja for

- Ako je potrebno da neki od izraza `izraz1`, `izraz2`, `izraz3` objedini više izraza, može se koristiti operator `,`

```
for (i = 0, j = 10; i < j; i++, j--)
 printf("i=%d, j=%d\t", i, j);
```

`i=0, j=10`   `i=1, j=9`

# Petlja for

- Ako je potrebno da neki od izraza `izraz1`, `izraz2`, `izraz3` objedini više izraza, može se koristiti operator `,`

```
for (i = 0, j = 10; i < j; i++, j--)
 printf("i=%d, j=%d\t", i, j);
```

`i=0, j=10`   `i=1, j=9`   `i=2, j=8`

# Petlja for

- Ako je potrebno da neki od izraza `izraz1`, `izraz2`, `izraz3` objedini više izraza, može se koristiti operator `,`

```
for (i = 0, j = 10; i < j; i++, j--)
 printf("i=%d, j=%d\t", i, j);
```

`i=0, j=10`   `i=1, j=9`   `i=2, j=8`   `i=3, j=7`

# Petlja for

- Ako je potrebno da neki od izraza `izraz1`, `izraz2`, `izraz3` objedini više izraza, može se koristiti operator `,`

```
for (i = 0, j = 10; i < j; i++, j--)
 printf("i=%d, j=%d\t", i, j);
```

`i=0, j=10`   `i=1, j=9`   `i=2, j=8`   `i=3, j=7`   `i=4, j=6`



# Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}

1 * 1 = 1
```

## Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}
```

1 \* 1 = 1    1 \* 2 = 2

## Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}
```

1 \* 1 = 1    1 \* 2 = 2    1 \* 3 = 3

## Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}
```

```
1 * 1 = 1 1 * 2 = 2 1 * 3 = 3
2 * 1 = 2
```

## Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}
```

```
1 * 1 = 1 1 * 2 = 2 1 * 3 = 3
2 * 1 = 2 2 * 2 = 4
```

## Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}
```

```
1 * 1 = 1 1 * 2 = 2 1 * 3 = 3
2 * 1 = 2 2 * 2 = 4 2 * 3 = 6
```

## Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}
```

```
1 * 1 = 1 1 * 2 = 2 1 * 3 = 3
2 * 1 = 2 2 * 2 = 4 2 * 3 = 6
3 * 1 = 3
```

## Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}
```

```
1 * 1 = 1 1 * 2 = 2 1 * 3 = 3
2 * 1 = 2 2 * 2 = 4 2 * 3 = 6
3 * 1 = 3 3 * 2 = 6
```



## Dvostruka for petlja

```
#include<stdio.h>

int main() {
 int i, j, n=3;
 for(i = 1; i <= n; i++) {
 for(j = 1; j <= n; j++)
 printf("%i * %i = %i\t", i, j, i*j);
 printf("\n");
 }
 return 0;
}
```

```
1 * 1 = 1 1 * 2 = 2 1 * 3 = 3
2 * 1 = 2 2 * 2 = 4 2 * 3 = 6
3 * 1 = 3 3 * 2 = 6 3 * 3 = 9
```

# Petlja do-while

- Petlja do-while ima sledeći opšti oblik:  
do {  
    naredbe  
} while(izraz)
- Telo (blok naredbi naredbe) naveden između vitičastih zagrada se izvršava i onda se izračunava uslov (izraz izraz). Ako je on tačan, telo se izvršava ponovo i to se nastavlja sve dok izraz izraz nema vrednost nula (tj. sve dok njegova istinitosna vrednost ne postane *netačno*).
- Za razliku od petlje while, naredbe u bloku ove petlje se uvek izvršavaju barem jednom.

# Primer

- Naredni program ispisuje cifre koje se koriste u zapisu unetog neoznačenog broja, zdesna na levo.

```
#include <stdio.h>
int main() {
 unsigned n;
 printf("Unesi broj: ");
 scanf("%u", &n);
 do {
 printf("%u ", n % 10);
 n /= 10;
 } while (n > 0);
 return 0;
}
```

Primetimo da, u slučaju da je korišćena while, a ne do-while petlja, za broj 0 ne bi bila ispisana ni jedna cifra.

# Petlja do-while

- Kako transformisati while petlju u do-while petlju?

```
while(izraz){
 naredbe
}
```

--->

# Petlja do-while

- Kako transformisati while petlju u do-while petlju?

```
while(izraz){
 naredbe
}
```

--->

```
if(izraz){
 do {
 naredbe
 } while(izraz)
}
```

# Petlja do-while

- Kako transformisati do-while petlju u while petlju?

```
do {
 naredbe
} while(izraz)
```

--->

# Petlja do-while

- Kako transformisati do-while petlju u while petlju?

```
do {
 naredbe
} while(izraz)
```

--->

```
naredbe
while(izraz){
 naredbe
}
```

# Naredba break

- U nekim situacijama pogodno je napustiti petlju ne zbog toga što nije ispunjen uslov petlje, već iz nekog drugog razloga. To je moguće postići naredbom `break`

```
for(i = 1; i < n; i++) {
 if(i > 10)
 break;
 ...
}
```

- Kod koji koristi `break` može se napisati i bez ove naredbe



# Naredba continue

- Naredbom `continue` se prelazi na sledeću iteraciju u petlji.  
Na primer,

```
for(i = 0; i < n; i++) {
 if (i % 10 == 0)
 continue; /* preskoci brojeve deljive sa 10 */
 ...
}
```

- Kod koji koristi `continue` može se napisati i bez ove naredbe

# Naredbe break i continue

- U slučaju ugnježdenih petlji, naredbe break i continue imaju dejstvo samo na unutrašnju petlju.

```
for (i = 0; i < 3; i++)
 for (j = 0; j < 3; j++) {
 if (i + j > 2) break;
 printf("%d %d ", i, j);
 }
```

# Literatura

Slajdovi su pripremljeni na osnovu materijala iz sedmog poglavlja knjige:

Filip Marić, Predrag Janičić: Programiranje 1

Za pripremu ispita nisu dovoljni slajdovi, potrebno je koristiti knjigu!