ZANKE

1

PONAVLJAJ, PONAVLJAJ, PONAVLJAJ, …

KAJ JE ŠTEVEC?
KAJ JE ZANKA?
while ZANKA
do..while ZANKA
for ZANKA

Kaj je zanka?

2

- zanka v programiranju
  - del programa, ki ga izvajamo vedno znova, dokler nimamo dovoljenja, da grem iz tega dela programa in izvedemo naslednji stavek
  - je programski stavek
  - uporabljamo za ponavljanje stavkov
  - kako dolgo ponavljamo je lahko odvisno od več stvari
    - imamo nek pogoj, ki to določa
    - imamo znano število ponovitev

Zanka - primeri

3

- poglejmo naslednje primere
  1. beri cela števila, dokler ne vnesemo sodega števila
  2. beri cela števila, dokler ne vnesemo sodega števila in preštej, koliko števil smo skupno vnesli
  3. na ekran izriši 99 zvezdic
- od česa je odvisna ponovitev
  1. od sodosti prebranega števila
  2. od sodosti prebranega števila
  3. od števila izrisanih zvezdic

Kaj je števec?

4

- števec je spremenljivka s katero štejemo
  - običajno je to celo število
- primeri (prejšnja prosojnica):
  - primer 2 – prešteti moramo koliko števil smo prebrali
  - primer 3 – štejemo število izrisanih zvezdic
- vsakemu števcu določimo
  - začetno vrednost
  - korak s katerim se spremeni
  - kdaj se števec spremeni

while zanka

5

- zamislimo si naslednjo situacijo
  - v učilnici imamo skupino računalnikov
  - mi moramo izklopiti vse vklopljene računalnike
  - kako bi pristopili k problemu?
    1. Preveri, ali je v učilnici kak vklopljen računalnik?
       Če je, nadaljuj na koraku št. 2, v nasprotnem primeru pojdi iz učilnice.
    2. Ugasni vklopljen računalnik. Vrni se na korak št. 1.
  - če v učilnici že na začetku ni vklopljenih računalnikov, nimamo dela

while zanka

6

- ali sploh kaj izvedemo je torej odvisno od nekega pogoja, ki ga preverimo na začetku
- while zanka je zanka s pogojem na začetku
- sintaksa je naslednja:

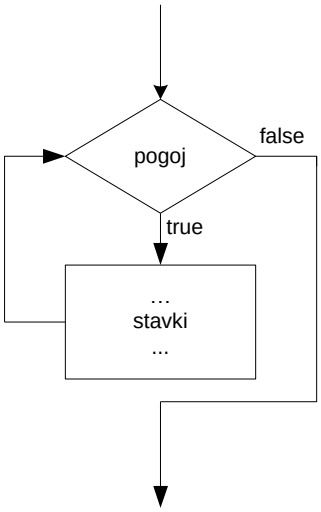
```
while (pogoj) {
   stavki;
}
```

while zanka – kako deluje

7

```
while (pogoj) {
        stavki
}
```

1. Ovrednoti pogoj. Če je pogoj resničen (*true*), nadaljuj na koraku 2. V nasprotnem primeru nadaljuj za zanko.
2. Izvedi stavke.
3. Skoči na korak 1.

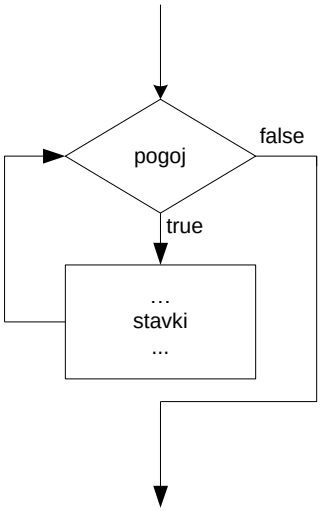


while zanka

8

```
while (pogoj) {
        stavki
}
```

- s pogojem povemo, kdaj se stavki znotraj zanke - znotraj zavitih oklepajev – izvedejo
- kadar imamo samo en stavek v zanki, lahko zavite oklepaje izpustimo



while zanka – primer in problem

9

```
// beremo cela števila, dokler ne vnesemo sodega števila


int stevilo;

// ponavljamo, če število ni sodo
while (stevilo % 2 != 0) {
  cout << "Vnesi celo sodo število: ";
  cin >> stevilo;
}
```

Ne vemo, koliko je število!

Kaj se zgodi, če je v pomnilniku ravno sodo število?
Potem se zanka nikoli ne izvede – nikoli ne vtipkamo števila.

while zanka - primer

10

```
// beremo cela števila, dokler ne vnesemo sodega števila

// zagotovimo, da število ni sodo
int stevilo = 1;

// ponavljamo, če število ni sodo
while (stevilo % 2 != 0) {
  cout << "Vnesi celo sodo število: ";
  cin >> stevilo;
}
```

do...while zanka

11

- v prejšnjem primeru smo želeli, da se zanka vsaj enkrat izvede
- to dosežemo z *do...while* zanko
- je zanka s pogojem na koncu
  - zato se stavki vsaj enkrat izvedejo – pogoj se preverja na koncu
- sintaksa

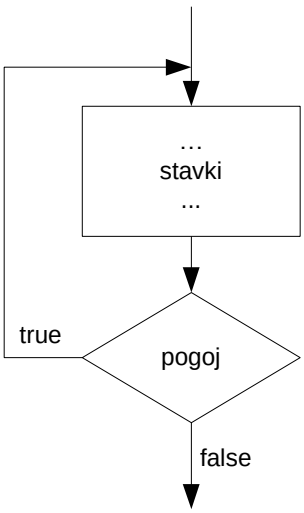
```
do {
   stavki;
} while (pogoj);
```

do...while zanka – kako deluje

12

```
do {
    stavki;
} while (pogoj);
```

1. Izvedi stavke.
2. Ovrednoti pogoj. Če je pogoj resničen (*true*), skoči na korak 1. V nasprotnem primeru nadaljuj za zanko.

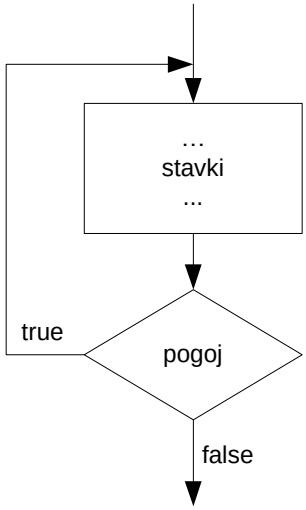


do...while zanka – kako deluje

13

```
do {
    stavki;
} while (pogoj);
```

- s pogojem povemo, kdaj se stavki znotraj zanke ponovno izvedejo



do...while zanka – primer 1

14

```
// beremo cela števila, dokler ne vnesemo sodega števila

int stevilo;



do {
  cout << "Vnesi celo sodo število: ";
  cin >> stevilo;
} while (stevilo % 2 != 0); // ponovimo, če število ni sodo
```

Inicializacija spremenljivke v tem primeru ni pomembna, saj na naslednjem koraku preberemo vrednost.

Torej imamo pri preverjanju pogoja že nadzor nad vrednostjo v spremenljivki.

do...while zanka – primer 2

15

```
// beremo cela števila, dokler ne vnesemo sodega števila
// in preštejemo, koliko števil smo vnesli

int stevilo;
int stevec = 0; // števec postavimo na začetno vrednost 0
                // saj do sem še nismo prebrali nobenega števila

do {
  cout << "Vnesi celo sodo število: ";
  cin >> stevilo;
  stevec = stevec + 1;  // s prejšnjim ukazom preberemo število
                        // zato na tem mestu povečamo števec –
                        // imamo eno število več
} while (stevilo % 2 != 0); // ponovimo, če število ni sodo

cout << "Vnesel si " << stevec << " stevil.";
```

do...while zanka – primer 3

16

```
// na ekran izriši 99 zvezdic

int stZvezdic = 0; // števec postavimo na začetno vrednost 0
                   // saj do sem še nismo izrisali nobene zvezdice

do {
  cout << "*";              // izrišemo eno zvezdico
  stZvezdic++;              // na ekranu je ena zvezdica več
} while (stZvezdic < 99);   // ponovimo, če je na ekranu manj
                            // kot 99 zvezdic
```

for zanka

17

- v prejšnjem primeru smo vedeli, **kolikokrat** se zanka ponovi
- ustreznejša izbira je for zanka
- je zanka namenjena za štetje
- sintaksa

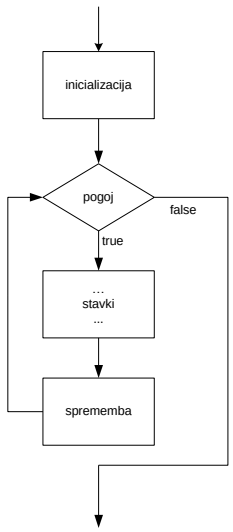
```
for (inicializacija; pogoj; sprememba) {
   stavki;
}
```

for zanka – kako deluje

18

```
for (inicializacija; pogoj; sprememba) {
     stavki;
};
```

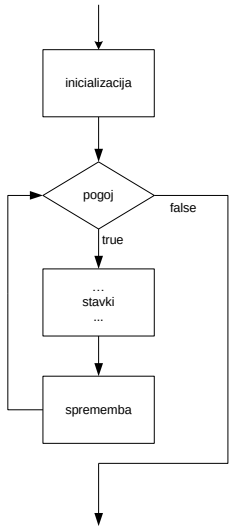
1. Izvedi inicializacijo.
2. Ovrednoti pogoj. Če je pogoj resničen (*true*), skoči na korak 3. V nasprotnem primeru nadaljuj za zanko.
3. Izvedi stavke.
4. Izvedi spremembo.
5. Skoči na korak 1.



for zanka – kako deluje

19

```
for (inicializacija; pogoj; sprememba) {
     stavki;
};
```



- inicializacija je lahko prazen stavek ali več stavkov ločenih z **vejicami**
- pogoj je nek logični izraz
- sprememba je lahko nič, en ali več stavkov ločenih z **vejicami**
- podpičja v for( ) pišemo vedno

for zanka – primer

20

```
// na ekran izriši 99 zvezdic – rešitev z do … while

int stZvezdic = 0;

do {
  cout << "*";
  stZvezdic++;
} while (stZvezdic < 99);
```

```
// na ekran izriši 99 zvezdic – rešitev s for zanko

for(int stZvezdic = 0; stZvezdic < 99; stZvezdic++) {
  cout << "*";
}
```

zanke – katero naj uporabim?

21

- pri zankah moramo določiti
  1. kaj se ponavlja – kateri stavki
  2. kdaj se ponavlja
     - bodisi je znano število ponovitev
     - bodisi je podan pogoj ustavitve
- na podlagi točke 2 se odločimo za ustrezno zanko (diagram na naslednji strani)

zanke – katero naj uporabim?

22