For zanka

1 Kako napišemo zanko?

V programiranju pogosto želimo nek del kode ponoviti, zato imamo *zanke*. Zanka, ki jo bomo najpogosteje uporabljali je *for* zanka, ki deluje na princip **začetka**, **pogoja** in **koraka**.

Primer

Najbolj osnoven program, ki uporablja for zanko.

```
#include <stdio.h>

int main() {

    //     začetek       pogoj       korak
    for (int stevec=0; stevec < 3; stevec++) {
        // koda, ki se izvedce vsako zanko
        printf("nekaj\n");
    }

    return 0;
}
```

Primer vhoda in izhoda

```
nekaj
nekaj
nekaj
```

Poglejmo si, kako ta program deluje. Podpičja v vrstici
`for (int stevec=0; stevec < 3; stevec++) {`
razdelijo okrogle oklepaje na tri dele; začetek, pogoj in korak. Začetek se bo izvedel, ko se ta zanka začne. Vsakič preden se izvede koda v notranjosti zanke se preveri pogoj, če drži, se bo še enkrat izvedla koda v zanki, sicer se bo pa zanka končala. Korak je podoben začetku, le da se izvede na koncu vsake ponovitve zanke.
Tu začetek naredi novo številko *stevec* in jo nastavi na 0. Pogoj preveri, če je *stevec* manjši od 3. Korak `stevec++` je pa okrajšava za `stevec = stevec + 1`, torej poveča *stevec* za 1.

- Program se začne in pride do for zanke, najprej se izvede začetek `int stevec=0`.
- Zdaj se je začela zanka, preveri se pogoj `stevec < 3`. Ker je *stevec* za zdaj še 0, je pogoj izpolnjen. Izvede se vsebina zanke, torej program izpiše *nekaj*. Zdaj smo prišli do konca zanke, izvede se korak, *stevec* se poveča na 1, program pa skoči nazaj na začetek zanke.
- Ker smo na začetku zanke se preveri pogoj, `stevec < 3`, ker je *stevec* zdaj 1 je pogoj še vedno izpolnjen, zato se izvede vsebina zanke. Ko program še enkrat izpiše *nekaj* izvede korak, *stevec* poveča na 2 in skoči nazaj na začetek.

- Spet smo na začetku, zato se preveri pogoj, *stevec* je zdaj 2, kar je manjše od 3, zato se *nekaj* spet izpiše. Program poveča *stevec* na 3 in skoči nazaj na začetek.
- Ker smo spet na začetku se bo še enkrat preveril pogoj, a zdaj je *stevec* enak 3 in 3 ni manjše od 3, zato se for zanka konča. Ker je naslednji ukaz `return 0;` se bo program tam končal.

Vidimo, da bo res izpisal *nekaj* trikrat. Vredno je omeniti, da je naš števec zavzel vrednosti 0, 1, 2, kar se mogoče zdi čudno, glede na to, da bi ponavadi šteli do tri kot 1, 2, 3, a je *štetje od nič* zelo pogosto v programiranju, zato smo se odločili, da vas že od začetka poskušamo navaditi nanj.

2 Razni primeri uporabe for zanke

2.1 Spreminjanje dolžine for zanke

Zanka, ki smo jo napisali zgoraj, se bo vedno ponovila trikrat, kaj pa če hočemo da se zanka ponovi glede na neko število na vhodu? Seveda je tudi to mogoče in sicer tako, da vstavimo našo spremenljivko v pogoj for zanke. Poglejmo si primer:

Primer

```
Program, ki dobi število in nariše puščico te dolžine.
Še en trik tu je, da printf-ja v for zanki ne končamo z \n, kar doseže
to, da so v izhodu pomišljaji eden zraven drugega v isti vrstici in ne
vsak v svoji vrstici.
```

```c
#include <stdio.h>

int main() {

    int dolzina;
    scanf("%d", &dolzina);

    for (int stevec=0; stevec < dolzina; stevec++) {
        printf("-");
    }

    printf(">\n");

    return 0;
}
```

Primer vhoda in izhoda

```
4
```

```
---->
```

2.2 Branje števil v for zanki

Ena od moči računalnikov je zelo hitra obdelava velike količine podatkov, računalnik bo zlahka seštel 1000 števil, medtem ko bi bilo to početi na roko precej zamudno. Poglejmo si, kako bi napisali program, ki bi nekaj izračunal z več števili.

Primer

Najprej preberemo eno število, recimo mu n.
Po njemu moramo prebrati še n števili in jih sešteti.

```c
#include <stdio.h>

int main() {

    int n;
    scanf("%d", &n);

    int vsota = 0;

    for (int i=0; i < n; i++) {
        int sestevanec;
        scanf("%d", &sestevanec);
        vsota += sestevanec;
    }

    printf("%d\n", vsota);

    return 0;
}
```

Primer vhoda in izhoda

```
4
12
13
8
1
```

```
34
```

V zgornjem primeru se je zgodilo precej novih stvari, poglejmo si vse po vrsti.
Najprej preberemo n iz navodil. Naredimo novo spremenljivko z imenom *vsota*, v njo bomo sešteli vsa dana števila.
Opazimo, da je v for zanki namesto *stevec* zdaj uporabljen i, tradicionalno se namreč v for zankah uporablja i. V notranjosti zanke so zdaj trije ukazi. Najprej naredimo novo spremenljivko, ki jo poimenujemo *sestevanec* in preberemo naslednje število iz vhoda. Nato pa z okrajšavo `vsota += sestevanec;` prištejemo spremenljivki *vsota* spremenljivko *sestevanec*. Na daljše bi to lahko napisali `vsota = vsota + sestevanec`.

2.3 While zanka in branje neznano mnogo števil

Naučili smo se, kako se prebere nekaj števil, ko nam je podano koliko števil moramo prebrati. Kaj pa če tega ne vemo, če hočemo pač prebrati vsa števila, ki so na vhodu? Zato lahko uporabimo *while* zanko, ki deluje tako kot for zanka, le da ima samo pogoj.

Primer

Preperimo vsa števila in jih zmnožimo.

```c
#include <stdio.h>

int main() {

    int produkt = 1;

    int clen;

    while (scanf("%d", &clen) == 1) {
        produkt = produkt * clen;
    }

    printf("%d\n", produkt);

    return 0;
}
```

Primer vhoda in izhoda

```
2
3
4
5
```

```
120
```

Kot že povedano, zgoraj napisani stavek while bi enako napisali kot `for ( ; scanf("%d") == 1 ; )`, torej for zanka, ki ima samo pogoj. Mogoče se zdi čudno, da ukaz scanf *primerjamo* s številko, a bomo kasneje v letu, ko se bomo učili o funkcijah razumeli, kaj to pomeni. Za zdaj pa bomo rekli le, da to, da je scanf *enak* ena pomeni, da je uspešno prebral in shranil eno številko. Če bi zgornji program prebral v človeških besedah, bi bilo:

- Nastavi *produkt* na 1
- Naredi novo spremenljivko *clen*
- Preberi eno število in ga shrani v *clen*, dokler tega ne moreš narediti več.
- Nastavi *produkt* na $produkt * clen$
- Ko končaš s množenjem vseh členov, izpiši rezultat.

Pogoste napake

Tu na začetku nastavimo produkt na 0, kar pa je napaka, saj je 0 krat karkoli še vedno 0. Naš program bo veno izpisal 0 ne glede na vhod.

```
#include <stdio.h>

int main() {

    int produkt = 0;

    int clen;

    while (scanf("%d", &clen) == 1) {
        produkt = produkt * clen;
    }

    printf("%d\n", produkt);

    return 0;
}
```

2.4 For zanka z drugačnim korakom in začetkom

Do zdaj so vse naše for zanke izgledale nekako tako `for (int i=0; i < 10; i++)`, torej so začele na nič in se nekajkrat ponovile. Ampak zapis for zanke, ki ga imamo v c++ lahko naredi veliko več. Poglejmo kot primer zanko, ki izpiše vsa soda števila med 1 in 100.
Pozorno poglejmo števila, ki jih moramo izpisati. Ker 1 ni sodo, bo prvo izpisano število 2. Število 3 prav tako ni sodo, tako da bomo izpisali 4, po tem pa 6, 8, 10 in tako dalje. Vidimo, da vsak korak povečamo izpisano število za 2, napišimo torej program.

Primer

Izpišemo vsa soda števila med 1 in 100.

```c
#include <stdio.h>

int main() {

    for (int i=2; i<=100; i += 2) {
        printf("%d\n", i);
    }

    return 0;
}
```

Primer vhoda in izhoda

```
2
4
6
8
10
12
⋮
96
98
100
```

Ker se želena števila začnejo z dva, bomo v začetni del for zanke vpisali `int i=2`. Ker hočemo izpisati števila med 1 in 100 in ne med 1 in 99 bomo v pogojnem delu uporabili znak manjše ali enako `i<=100` (pogoj `i<100` ne bi veljal za število 100). Ker želimo povečati naše število za 2 vsak korak, smo v polje za korak napisali `i += 2`. Edina stvar, ki jo naredimo v notranjosti napisane zanke pa je, da izpišemo trenutno vrednost spremenljivke i.