Zapiski s priprav

Zanke

ACM priprave na tekmovanja

Različica z dne 27. november 2024

Kazalo

1 Sintaksa

V programiranju pogosto želimo nek del kode ponoviti, zato uporabljamo *zanke*. Poznamo več zank, a najpogosteje uporabljamo zanko `for`. Le-ta ima tri posebne komponente: *začetek*, *pogoj* in *korak*. Poglejmo si preprost primer, ki trikrat izpiše besedo `nekaj`.

```c
#include <stdio.h>

int main() {
    //   začetek;      pogoj;      korak
    for (int stevec=0; stevec < 3; stevec++) {
        // med zavitimi oklepaji {} je koda,
        // ki se izvede vsako iteracijo zanke
        printf("nekaj\n");
    }
    return 0;
}
```

Poglejmo si, kako ta program deluje. Podpičja v peti vrstici razdelijo okrogle oklepaje na tri dele: začetek (`int stevec=0`), pogoj (`stevec < 3`) in korak (`stevec++`). Začetek se bo izvedel, ko se ta zanka začne. Vsakič preden se izvede koda v notranjosti zanke, se preveri pogoj. Če pogoj drži, se bo izvedla koda v zanki, sicer pa se bo zanka končala. Korak je podoben začetku, in se izvede na koncu vsake ponovitve zanke.

V zgornjem primeru začetek naredi novo številko `stevec` in jo nastavi na 0. Pogoj preveri, če je `stevec` manjši od 3, korak `stevec++` pa je okrajšava za `stevec = stevec + 1`, torej poveča `stevec` za 1. Program sledi naslednjem postopku:

1. Program se začne in pride do for zanke, najprej se izvede začetek `int stevec=0`.
2. 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, zato se `stevec` poveča na 1, program pa skoči nazaj na začetek zanke.
3. Ker smo na začetku zanke, se preveri pogoj, `stevec < 3`, `stevec` je zdaj 1 in 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.
4. Spet smo na začetku, zato se preveri pogoj, `stevec` je zdaj 2, kar je manjše od 3, zato se `nekaj` izpiše še tretjič. Program potem poveča `stevec` na 3 in skoči nazaj na začetek.
5. 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 zanka konča. Ker je naslednji ukaz `return 0`, se bo program tam končal.

Vredno je omeniti, da je naš števec zavzel vrednosti 0, 1, in 2, kar se morda zdi čudno, glede na to, da bi ponavadi šteli do tri kot $1, 2, 3$. Tovrstno štetje od nič je zelo pogosto v programiranju, in bo prišlo prav kasneje, ko se bomo učili o seznamih.

2 Primeri uporabe

2.1 Spreminjanje dolžine 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 zanke. Poglejmo si primer.

```c
#include <stdio.h>

int main() {
    int dolzina;
    scanf("%d", &dolzina);
    for (int stevec=0; stevec < dolzina; stevec++) {
        printf("-");
    }
    printf(">\n");
    return 0;
}
```

Zgornji program sprejme število in nariše puščico te dolžine. Tu uporabimo še en trik, in sicer v funkciji `printf` znotraj zanke ne dodamo `\n`, s čimer dosežemo to, da so v izhodu znaki `-` eden zraven drugega v isti vrstici in ne vsak v svoji.

2.2 Branje števil v zanki

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

```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;
}
```

V zgornjem primeru prvo preberemo število `n`, nato pa ustvarimo spremenljivko `vsota`, ki jo takoj nastavimo na 0. V tej spremenljivki bomo hranili vsoto števil, ki smo jih do sedaj videli na vhodu (razen `n`), na koncu programa bo torej enaka vsoti vseh takih števil.

Opazimo, da v zanki namesto `stevec` sedaj uporabljamo `i`, ki je tradicionalna izbira za spremenljivko v zanki. 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 kot `vsota = vsota + sestevanec`. V vsaki iteraciji tako prištejemo ravno prebrano število k vsoti, na koncu pa bomo izpisali vsoto vseh.

2.3 Zanka z drugačnim korakom in začetkom

Do zdaj so vse naše zanke takole: `for (int i=0; i < 10; i++)`, torej so začele z nič in se nekajkrat ponovile. C++ pa nam dovoli, da lahko z našimi zankami naredimo veliko več. Kot primer si poglejmo 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.

```
#include <stdio.h>

int main() {
    for (int i=2; i<=100; i += 2) {
        printf("%d\n", i);
    }
    return 0;
}
```

Ker se želena števila začnejo z dva, bomo v začetni del zanke vpisali `int i=2`. Ker želimo 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`.