Še o zankah

Zanka `for`

Zanka `for` je izpopolnjena oblika zanke `while`, ki nam omogoča bolj strnjen zapis stavkov, s katerimi kontroliramo potek zanke. Osnovna oblika je:

```
for (start; pogoj; korak) stavek;
```

Pri tem je `start` stavek, ki se izvrši samo enkrat, preden se zanka sploh prične. Običajno s tem stavkom nastavimo začetno vrednost kontrolne spremenljivke (števec ali kaj podobnega), ki jo uporabljamo v zanki. Takoj zatem se preveri pogoj. Če je izpolnjen (če ima od 0 različno vrednost), se izvede telo zanke (stavek, ki sledi), nato pa še stavek `korak`, s katerim spremenimo vrednost kontrolne spremenljivke. Potem se spet preveri pogoj. Dokler je izpolnjen, se izvajata telo zanke in stavek `korak`. Ko pogoj ni več izpolnjen, se zanka konča.

Če je stavek `start` definicija nove spremenljivke, potem lahko to spremenljivko uporabljamo samo znotraj zanke. Izven zanke je ne moremo uporabljati. Če nas po izteku zanke zanima njena končna vrednost, jo moramo deklarirati že pred zanko.

Izpis prvih desetih naravnih števil bi z zanko `for` naredili takole:

```
for (int i = 1; i <= 10; ++i) printf("%d\n", i);
```

Zanka `for` je torej samo krajši zapis naslednje oblike zanke `while`:

```
{
   start;
   while (pogoj) {
      stavek;
      korak;
   }
}
```

Zunanji zaviti oklepaji so potrebni, da omejimo vidnost števca, definiranega v stavku `start` (kadar gre za definicijo nove spremenljivke).

Zanka `do`

Zanka `do` je v primerjavi s prejšnjima dvema zankama nekaj posebnega. Pri zankah `while` in `for` se veljavnost pogoja preverja pred vsako ponovitvijo telesa zanke, pri zanki `do` pa se najprej izvede telo zanke in šele nato preveri veljavnost pogoja. Zanka `do` se torej v vsakem primeru izvede vsaj enkrat. Zapišemo jo takole:

```
do stavek; while (pogoj);
```

Zanko `do` torej uporabimo takrat, kadar se mora telo zanke izvršiti vsaj enkrat. Tipičen primer uporabe je preverjanje smiselnosti vnešenih podatkov. Od uporabnika zahtevamo vnos podatka tako dolgo, da vnese smiselno vrednost.

```
int stevilo;
do {
   printf("Vnesi stirimestno naravno stevilo: ");
   scanf("%d", &stevilo);
} while (stevilo < 1000 || stevilo > 9999);
```

Uporabnik mora število vnesti vsaj enkrat. Če ne vnese štirimestnega števila, se zanka ponovi. Deklaracija spremenljivke `stevilo` stoji pred zanko. Če bi jo napisali znotraj zanke, ne bi bila vidna v pogoju, tako da ne bi mogli preveriti smiselnosti njene vrednosti.

Stavek `break`

Stavek `break` uporabljamo znotraj zank, kadar želimo izvajanje zanke predčasno prekiniti. Program se nadaljuje s stavkom, ki sledi zanki.

Stavek `continue`

Stavek `continue` uporabljamo znotraj zank, kadar želimo trenutno ponovitev zanke predčasno prekiniti in nadaljevati z naslednjo ponovitvijo (če je pogoj za ponovitev zanke še vedno izpolnjen). Če ga uporabimo v telesu zank `while` ali `do`, takoj preveri pogoj za ponovitev zanke, v zanki `for` pa najprej izvede stavek za spremembo kontrolne spremenljivke (korak), in šele nato preveri pogoj.

Primer

Sestavimo zanko, v kateri bo uporabnik vnašal cela števila, dokler ne vnese števila 0. Na koncu naj program izpiše povprečje pozitivnih vnešenih števil.

```
int n = 0;
int vsota = 0;
while (1) {
   int stevilo;
   printf("Vnesi celo stevilo: ");
   scanf("%d", &stevilo);
   if (stevilo < 0) continue;
   if (stevilo == 0) break;
   vsota += stevilo;
   ++n;
}
if (n == 0) printf("Ni bilo pozitivnih stevil.\n");
else printf("Povprecje pozitivnih je %g.\n", (double)vsota / n);
```

Povprečje znamo izračunati, če poznamo vsoto števil in njihovo število (koliko jih je). Zato pred zanko definiramo spremenljivki `n` in `vsota`. V zanki ju ustrezno povečujemo, na koncu (po izteku zanke) pa uporabimo pri izpisu.

V telesu zanke poskrbimo za vnos celega števila, potem pa ustrezno ukrepamo. Če je število negativno, ga ignoriramo (sprožimo naslednjo ponovitev zanke), če je enako 0, prekinemo izvajanje zanke (to je tudi edini izhod iz zanke, saj je pogoj za izvajanje zanke vedno različen od 0, torej vedno izpolnjen). Do zadnjih dveh stavkov v telesu zanke pridemo samo, če je število pozitivno, zato bi bil še en pogojni stavek odveč.

Gnezdenje

Tudi zanke so čisto običajni stavki, zato jih lahko uporabljamo znotraj drugih zank. Temu rečemo gnezdenje zank. Pri tem je zelo pomembno, da ne pomešamo števcev posameznih zank (več zank bi načeloma lahko uporabljalo isti števec, a taki primeri so zelo redki). Če bi na primer želeli na zaslon izpisati `n` vrstic zvezdic, pri čemer bi v prvi vrstici imeli eno zvezdico, v drugi dve, v tretji tri, itn., bi zanki sestavili takole:

```
for (int i = 1; i <= n; ++i) {
   for (int j = 1; j <= i; ++j) printf("*");
   printf("\n");
}
```

Števec `i` v zunanji zanki šteje vrstice. V `i`-ti vrstici moramo izpisati `i` zvezdic, kar storimo v notranji zanki, ki `i`-krat izpiše po eno zvezdico. Ko je notranje zanke konec, skočimo v novo vrstico, in ponovimo zunanjo zanko (če je pogoj še vedno izpolnjen).