Zanka `while`

Zanko v programu uporabljamo takrat, ko želimo enega ali več stavkov izvršiti večkrat zaporedoma. C pozna tri vrste zank: `while`, `for` in `do`. Najpreprostejša je zanka `while`:

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

To pomeni: "dokler je izpolnjen pogoj, ponavljaj stavek". Poglejmo podrobneje. Program najprej preveri pogoj, ki ga moramo vedno napisati v oklepajih. Če pogoj ni izpolnjen (če ima vrednost 0), se ne zgodi ničesar, sicer pa se izvede *telo zanke*. To je stavek, ki sledi pogoju. Potem se spet preveri pravilnost pogoja. Če je še vedno izpolnjen, se telo zanke izvede še enkrat. To se ponavlja, dokler je pogoj izpoljen. Ko pogoj ni več izpolnjen, se zanka konča. Nepazljiv programer lahko sestavi pogoj, ki bo vedno izpolnjen. V takem primeru se zanka ne bo nikoli končala. Pri programiranju zank moramo torej zelo paziti, da ne sestavimo neskončne zanke.

Verjetno ni treba dodajati, da lahko za telo zanke namesto preprostega stavka uporabimo tudi sestavljeni stavek (blok). Manj znano pa je, da je lahko telo zanke tudi prazno (še vedno moramo napisati podpičje). Taki primeri so sicer redki, vendar so dovoljeni, zato prevajalnik tega ne obravnava kot napako, čeprav se velikokrat izkaže, da gre za napako (odvečno podpičje med pogojem in stavkom).

Pri sestavljanju zank velikokrat uporabimo posebno kontrolno spremenljivko, s pomočjo katere se odločamo, kdaj je potrebno zanko končati. Največkrat je to števec, ki šteje kolikokrat se mora zanka še ponoviti ali pa kolikokrat se je že izvršila. Začetno vrednost te spremenljivke nastavimo pred zanko, njeno trenutno vrednost preverjamo v pogoju, spreminjamo pa jo v telesu zanke (največkrat tik pred koncem). Za števce lahko uporabljamo poljubne spremenljivke, največkrat pa izberemo kratka imena, kot so `i`, `j` ali `k`.

Poglejmo si primer zanke, ki izpiše vsa naravna števila med 1 in 10.

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

Pred zanko števcu `i` dodelimo začetno vrednost 1. Dokler je `i` manjši ali enak 10, izpisujemo njegovo vrednost, po vsakem izpisu pa mu vrednost še povečamo za 1. Ko števec `i` dobi vrednost 11, se zanka konča. Enak učinek bi dosegli tudi z desetimi zaporednimi stavki za izpis števil od 1 do 10, vendar pa tak način odpove, če bi morali izpisati vsa naravna števila od 1 do 10000, ali pa vsa naravna števila od 1 do `n`.

Primer

Izpišimo in preštejmo vse delitelje naravnega števila `n`. Po vrsti bomo za vsako število od 1 do `n` preverili, ali deli število `n`. Če ga deli, ga bomo izpisali in povečali števec deliteljev.

```
int stevec = 0;
int i = 1;
while (i <= n) {
   if (n % i == 0) {
      printf("%d\n", i);
      ++stevec;
   }
   ++i;
}
```

V spremenljivki `stevec` štejemo že znane delitelje števila `n`. Ker na začetku ne poznamo še nobenega delitelja, je začetna vrednost tega števca enaka 0. Spremenljivko `i` uporabimo za kandidate za delitelje. Vrednosti te spremenljivke tečejo od 1 do `n`. Pred zanko začetno vrednost nastavimo na 1, v pogoju zanke preverjamo, ali je vrednost še manjša ali enaka `n`, tik pred koncem zanke pa vrednost povečamo za 1. V zanki najprej preverimo, ali število `i` deli število `n` (ali je ostanek pri deljenju enak 0). Če je pogoj izpolnjen, izpišemo navega delitelja in povečamo števec deliteljev.

Primer

Seštejmo vse števke naravnega števila `n`. Če bi vedeli, koliko mest ima število, bi z uporabo operatorjev `/` in `%` izračunali posamezne števke in jih sešteli. Ker pa dolžine števila ne poznamo, moramo uporabiti zanko. Zanka se bo ponovila tolikokrat, kolikor števk ima število `n`. V vsaki ponovitvi zanke bomo k trenutni vsoti prišteli eno števko.

```
int vsota = 0;
while (n > 0) {
   int enice = n % 10;
   vsota += enice;
   n /= 10;
}
```

Trenutno vsoto vodimo v spremenljivki `vsota`. V vsaki ponovitvi zanke izračunamo enice števila `n`, jih prištejemo k trenutni vsoti, nato pa število `n` skrajšamo za eno mesto, tako da mu odrežemo enice (število delimo z 10). Zanko ponavljamo, dokler število `n` vsebuje še kakšno števko (dokler je večje od 0).