Programski jezik c – nadaljevanje

21.10.1997

While zanka

While zanko zapišemo v obliki:

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

Če je stavek v while zanki sestavljen iz večih stavkov, le te navedemo v zavitih oklepajih:

```
while(pogoj) {
  stavek1;
  stavek2;
}
```

Pri zaključku zanke (}) je priporočljivo pripisati komentar, posebej v primerih, ko je stavkov oziroma vrstic v zanki več kot 20 ali pa ko so v zanki vsebovane še druge `while` oziroma `for` zanke. Tako označimo kje se konča kakšna zanka.

Pozorni moramo biti na to, da za `while` (pogoj) ne pišemo podpičja, saj lahko s tem ustvarimo neskončno zanko (v primeru ko je pogoj izpolnjen) ali pa se zanka šteje kot nepotrebna, saj jo računalnik preskoči v primeru ko pogoj ni izpolnjen).

Primer programa z while zanko

Napiši program, ki bo seštel števila od 1 do n. Pred zanko moramo zapisati, da začnemo s številom n=1 in je začetna vsota enaka 0.

Program:

```
#include <stdio.h>
#include <conio.h>
void main (void) {
  clrscr();
  int n,np,sum ;

  n= 1;
  sum= 0;
  printf("Vsoto koliko stevil zelimo izracunati?\n");
  scanf("%d", &np);
  while (n<=np) {
    sum = sum + n;      /* pisemo lahko tudi sum += n */
    n = n + 1;          /* ali n += 1 ali n++ */
  }                     /* ali sum += n++; kot en stavek   */
  printf("Vsota %d stevil je %d \n",np,sum);
}
```

For stavek

For stavek ima obliko:

```
for (izraz 1 ; izraz 2 ; izraz 3)
  stavek;
```

Če pa so stavki sestavljeni:

```
for (izraz1: izraz2: izraz3) {
  stavek 1;
  stavek 2;
  ...
 }
```

Izraze med sabo ločimo z podpičji. V njih pa navajamo naslednje stvari:

- V *izrazu 1* navedemo, kaj mora računalnik narediti na začetku (inicializacija).
- V *izrazu 2* povemo, do kdaj naj ponavlja.
- V *izrazu 3* pa povemo, kaj naj v zanki izračuna oz. spremeni.

Prejšnji program sestavljen s for stavkom

```
...
sum=0;
for (n=1; n<=np; n++)
  sum+=n
...
```

S tem smo prejšnjo while zanko nadomestili s stavkom for.

Izrazi v for stavku so lahko tudi nizi izrazov. Takšne izraze ločimo med seboj z vejicami.

Prejšnji program s sestavljenimi izrazi

```
...
for (n=1,sum=0; n<=np; sum+=n,n++);
                  /* for (n=1,sum=0; n<=np; sum += n++ */
...
```

V tem `for` stavku smo zapisali vse začetne podatke, pogoj za ponavljanje in celotno računanje v zanki. S podpičjem povemo, da ni zunaj te for zanke nobene dodatne operacije.

Tu moramo paziti kaj pomeni `n++` in kaj `++n`. Če pišemo:

- `n++` pomeni, da najprej velja vrednost `n`, potem pa se `n` poveča.
- `++n` pa pomeni, da se `n` najprej poveča potem pa se začnejo izvajati ostali stavki.

Računanje $I_n = e \int_0^1 x^n e^{-x} dx$

```
            /* e je oznaka za stevilo 2.71828...*/
#define e 2.71828183
{
  int np,n;
  float I;

  n= 0;
  I= e-1;

  while (n<=np) {
    printf("%d %f \n", n, I);
```

```
    n++;
    I=n*I-1;
 }
}
```

Prejšnji program napisan s for stavkom:

```
  ...
  I=e-1;
   for(n=0; n<=np, n++)
     {
     printf("%d %f \n",n,I);
     I = (n+1)*I - 1;
     }
 ...
```

Enostavnejša oblika prejšnjega for stavka

```
 ...
 for(n=0, I=e-1; n<=np; printf(...), I=(n+1)*I-1, n++);
 ...
```

Še druga oblika

```
 ...
 for(......; n<=np; printf(...), I=++n*I-1);
 ...
```

Opozorilo: *Podpičje za stavkom for pomeni, da ni v zanki nobene dodatne operacije več.*

Kadar v izrazu zanke `for` nastopa več izrazov, je vprašanje vrstnega reda izvajanja odvisno od prevajalnika.

Pogojni stavek

Pogojni stavek izgleda takole:

```
  if(pogoj)
    stavek 1;
  else
    stavek 2;
```

Če je pogoj izpolnjen se izvede stavek 1, drugače pa se izvede stavek 2.

Primer

```
int a, b, max;
if (a>b)
  max=a;
else
  max=b;
```

V primeru, da znotraj pogojnega stavka nastopa še kakšen pogojni stavek, je priporočljivo označiti tiste, ki spadajo skupaj s komentarjem.

Primer programa s pogojnim stavkom

Računanje rešitev kvadratne enačbe $ax^2 + bx + c = 0$. Podatki so a, b, c.

```
...
if (a==0) {                        /* linearna enacba */
  if(b==0) {
    if (c==0)
      printf("Neskoncno resitev\n");
    else                           /* c ni enak 0 */
      printf("Ni resitve \n");
    }
  else                             /* b ni enak 0 */
    printf("Resitev=%f \n", -c/b);
  }
else {                             /* a ni enak 0 */
  d= b*b - 4*a*c;
  if (d >= 0)
    printf("x1= %f, x2= %f \n",
           -b+sqrt(d)/(2*a), -b-sqrt(d)/(2*a));
  else                             /* d < 0 */
    printf("x1,2= %f +- %fi \n",
           -b/(2*a), sqrt(-d)/(2*a));
}
```