4. Vezérlési szerkezetek

4.1 Szelekció

Az utasításokat a '{' (begin) ill. '}' (end) zárójelek használatával egyetlen összetett utasításba, más néven blokkba foghatjuk össze. A '}' után nincs pontosvesszô !

Az if - else utasítás

Az if - else utasítás alakja :

if(kifejezés)
1. utasítás(blokk)
else
2. utasítás(blokk)

Amennyiben a kifejezés igaz - **azaz értéke nem nulla** - akkor az 1. utasitás(blokk), ha a kifejezés hamis - **azaz értéke nulla** - akkor a 2. utasítás(blokk) kerül végrehajtásra. Az else használata nem kötelezô ! Ha az if ill. else kulcsszót csak egy utasítás követi, akkor nem kötelezô a '{' ill. '}' jelek használata, egyébként igen !

1. sz. példaprogram (részlet) :

```
if( a%2 == 0 )
{
    a = a/2;
}
else
{
    a = 3*a + 1;
    if( a>max ) max = a;
}
```

Fontos : Mivel az else használata nem kötelezô, ezért elôfordulhat, hogy összetettebb alakokban nem egyezik meg az if-ek és az else - k száma. Annak eldöntésére, hogy melyik else melyik if - hez tartozik a következô szabály érvényes : Az else a hozzá legközelebb esô else nélküli if-hez tartozik !
Ajánlatos a '{' ill. a '}' zárójelek alkalmazása, így egyértelmûvé tehetjük a szerkezetet !

2. *sz. példaprogram (részlet) :*

```
if( n == 0 )
    if( a>b )
       x = a;
    else
          y = b;
```

Az elsô if után azért nem kell '{' ill. '}' mert az azt követô if egy utasításnak felel meg. Jelen konstrukcióban az else ág az if(a>b) - hez tartozik. Ha eredetileg nem ezt akartuk akkor a '{' ill. '}' használatával tudunk változtatni:

3. sz. példaprogram (részlet) :

```
if( n == 0 )
{
    if( a>b )
            x = a;
}
else
    y = b;
```

A ? operátor

Az if - else kifejezés egy speciális rövidítése : az ilyen típusú alakot, mint az

```
if( a>b )
   z = a;
else
   z = b;
```

a '?' operátor használatával rövidíthetünk.

Általános alak :

kif1 ? kif2 : kif3 ;

Ha kif1 értéke igaz akkor kif2, egyébként kif3 lesz ennek a kifejezésnek az értéke:

```
z = (a>b) ? a : b;        /* z = max(a,b) */
```

Az else - if szerkezet - többszörös elágazás I.

Általános alakja :

```
if( kifejezés )
      utasítás(blokk)
else if( kifejezés )
      utasítás(blokk)
else if( kifejezés )
      utasítás(blokk)
else
      utasítás(blokk)
```

Itt meg kell jegyezni, hogy gyakorlatilag nem külön utasításról van szó, hiszen egymásba ágyazott kétágú szelekciókkal van dolgunk. Mivel gyakran elôfordul ilyen döntési szerkezet, ezért az áttekinthetô programozási alak miatt külön tárgyaljuk.

A kifejezések - egymás után történô - kiértékelése után az az utasítás kerül végrehajtásra, amelyhez tartozó kifejezés igaznak bizonyult. Ezután a program kilép az else-if szerkezetbôl. Az utasítások helyén egy utasítás vagy '{' ill. '}' között utasításblokk állhat. Az utolsó else a "fentiek közül egy sem" szituációt kezeli. A hozzá tartozó utasítás csak akkor kerül végrehajtásra, ha elôtte mindegyik kifejezés hamisnak bizonyult.

4. sz. példaprogram (részlet) :

```
k = (x-u)*(x-u) + (y-v)*(y-v);


if( k == r*r )
    printf("\nRajta van a körön !");

else if( k > r*r )
    printf("\nKivül van a körön !");

else
    printf("\nBelül van a körön !");
```

A switch utasítás - többszörös elágazás II.

Általános alakja :

```
switch( kifejezés )
      {
      case konstans_1:
            utasítás(blokk)
            break;
 case konstans_2:
            utasítás(blokk)
            break;
      .
      .
      .
      case konstans_i:
            utasítás(blokk)
            break;
      default:
            utasítás(blokk)
            break;
     }
```

A kifejezés csak egész típusú lehet ! Kiértékelése után azzal az utasítás-(blokk)-kal folytatódik a program amelyhez tartozó case konstans megegyezett a kifejezés értékével. A break hatására befejezôdik a switch végrehajtása. A default utáni utasítás(blokk) - ra akkor adódik a vezérlés, ha egyik case-re sem teljesült az egyenlôség. A case után csak a kifejezés típusának megfelelô állandót lehet megadni ! A **case** utáni kifejezés típusa csak egész jellegû lehet, tehát char, int vagy long. Egymás után több case-t is fel sorolhatunk.

5. sz. példaprogram (részlet) :

```
switch(c)  /* c - ben egy karakter értéke található */
{
    case '2':
    case '4':
    case '6':
    case '8':
                printf("\nPáros !");
                break;
    case '1':
    case '3':
    case '5':
    case '7':


    case '9':
                printf("\nPáratlan !");
                break;
    case '0':
                printf("\Nulla !");
                break;
    default:
                if( c >= 'A'  && c <= 'Z' )
                    printf("\nNagybetû !");
                else if( c >= 'a'  && c <= 'z' )
                    printf("\nKisbetû !");
                else
                    printf("\nEgyéb !");

                break;
}
```

Fontos : Ha egyszer beléptünk egy *switch* utasítás valamelyik *case* ágába, akkor az utasítások mindaddig végrehajtódnak amíg egy break utasításra nem jut a vezérlés ! Két case-nek nem lehet ugyanaz az értéke !

4.2 Ciklusok

Elôltesztelô ciklus : A while utasítás.

A while utasítás alakja :

while(kifejezés)
utasítás(blokk)

A program mindaddig végrehajtja az utasítás(blokk)-ot amíg a while-t követô kifejezés értéke IGAZ, azaz 0-tól különbözik .

6. sz. példaprogram :

```
      // Az  1 - 1/2 + 1/4 - 1/8 ... váltakozó elôjelü sor
      // összegének kiszámítása a billentyûzetrôl megadott
      // pontossággal. A sor a (-1)^(n-1)*(1/2^(n-1)) zárt
      // alakban írható fel és összege : 2/3.
      // Vezérlési szerkezet : a WHILE utasítás

      #include <stdio.h>
      #include <conio.h>
      #include <math.h>     // Az fabs() és a pow() függvé-
                                 // nyek miatt szükséges
      void main()
      {

         double n, e, s, os;

         printf("\nKérem a pontosságot : ");
         scanf("%lf",&e) ;

         s = 1 ; os = 0 ; n = 1 ;

                    // Amíg az új és a régi összeg
különbségének
                    // abszolút értéke nagyobbb mint a
hibahatár,
                    // addig, folytatódjon a sorösszeg képzés.
         while( fabs( s - os ) > e )
         {
                    os = s ; n += 1 ;
                    s += pow( -1, n-1 ) * (1.0/pow(2,n-1)) ;
         }
         printf("\nAz összeg : %lf",s);
         getch();
      }
```

Hátultesztelô ciklus : A do - while utasítás

A do - while utasítás alakja :

do
    utasítás(blokk)
while(kifejezés);

A program elôször végrehajtja az utasítás(blokk)-ot, majd kiértékeli a kifejezést. Ez a ciklikus tevékenység folyik mindaddig, amíg a kifejezés értéke IGAZ, azaz 0-tól különbözik. Mint látható, a C nyelv hátultesztelô ciklusa eltér a - több programnyelvnél - megszokott UNTIL típusú hátultesztelô ciklustól, amelynél a ciklus addig mûködik, amíg az UNTIL-t követô feltétel hamis.
Amennyiben valaki az UNTIL alakot tartja "igazi" hátultesztelô ciklusnak, úgy a #define direktíva segítségével, bármikor elôállíthatja az UNTIL alakot, amint azt a 8. sz példaprogram mutatja.

7. sz. példaprogram :

```
// A 6. sz. példaprogramban szereplô feladat megol-
// dása a DO WHILE utasítás alkalmazásával.

#include <stdio.h>
#include <conio.h>
#include <math.h>

void main()
{

   double n, e, s, os;

   printf("\nKérem a pontosságot : ") ;
   scanf("%lf",&e) ;

   s = 0 ; os = 0 ; n = 1 ;

   do
   {
        os = s;
        s += pow( -1, n-1 ) * (1.0/pow(2,n-1)) ;
        n += 1 ;
   } while( fabs( s - os ) > e ) ;

   printf("\nAz összeg : %lf",s);
   getch();
}
```

8. sz. példaprogram :

```
// A 6. sz. példaprogramban szereplô feladat megol-
// dása az általunk definiált UNTIL utasítás alkal
// mazásával.

#include <stdio.h>
#include <conio.h>
#include <math.h>

#define UNTIL(condition) while(!(condition))

void main()
{
   double n, e, s, os;

   printf("\nKérem a pontosságot : ") ;
   scanf("%lf",&e) ;
   s = 0 ; os = 0 ; n = 1 ;

   do
   {
        os = s;
        s += pow( -1, n-1 ) * (1.0/pow(2,n-1)) ;
        n += 1 ;
   } UNTIL( fabs( s - os ) <= e ) ;

   printf("\nAz összeg : %lf",s); getch();
}
```

A for utasítás

A for utasítás alakja :

for(kifejezés1; kifejezés2; kifejezés3)
utasítás(blokk)

A kifejezések általában a következôket jelentik :

kifejezés1 : Kezdeti értékadás a ciklusváltozónak.
kifejezés2 : A lefutási feltétel(ek) megfogalmazása. Lefutási feltételként nem csak a ciklusváltozóval kapcsolatos feltételt lehet megadni hanem bármilyen más feltételt is.
,
kifejezés3 : A ciklusváltozó növelése vagy csökkentése.
Bármelyik kifejezés hiányozhat !

Példák :

1./ for(i=1; i<=100; i++)
```
    {
            printf("\n A(z) %d négyzete = %d",i,i*i);
    }
```

// Kinyomtatja a természetes számok négyzetét 1-100 - ig.

2./ A for ciklusok egymásba ágyazhatóak :

```
int i, j, prim, n;
...
for( i=5; i<=n; i=i+2 )          // Páratlan számok képzése 5-tôl n-ig
{
      prim = 1;
      for( j=3; j<=i-1 && prim == 1; j++)    // Természetes számok képzése
      {                                              // 3-tól i-1 ig.
          if( i%j == 0 ) prim = 0;               // Ha j osztója i-nek akkor i
      }                                              // nem prímszám
      if( prim == 1 ) printf("\n%3d",i);
}
```

A belsô for ciklus akkor is befejezôdik, ha a prim változó értéke 0 lesz így nem biztos, hogy lefut (i - 1) - ig. A külsô for ciklus i változójának egy értékéhez a belsô for ciklus a feltételében meghatározottak szerint kerül végrehajtásra.
Ezután i értéke kettôvel növelôdik, és újra a belsô ciklus fut le. Ez így megy mindaddig, amíg i el nem éri n értékét. A for ciklus több ciklusváltozót is tartalmazhat, ezeket a ',' operátorral kell elválasztani egymástól.

9. sz. példaprogram (részlet) :

```
for( i=1, j=2; i<=100; i+=2, j+=2)
{
    printf("\n Páratlan = %d < - > Páros = %d",i,j);
}
```

Bizonyos esetekben nagyon kényelmes a használata !

Fontos : Ez a konstrukció nem tévesztendô össze az egymásba ágyazott for ciklussal.

Megjegyzés : A for utasítás egyenértékü a

```
kifejezés1;
while( kifejezés2 )
{
   utasítás(blokk)
   kifejezés3;
}
```

alakkal.

Kilépés ciklusból ill. a switch-bôl : A break utasítás.

A break hatására a vezérlés azonnal kilép a for, while, do-while ill. switch konstrukciókból, és a konstrukciót követô elsô utasítással folytatódik a program végrehajtása. Egymásba ágyazott szerkezetek esetén mindig abból a legbelsô ciklusból lép ki a break, amelyik azt tartalmazza.

Következô iteráció kezdeményezése : A continue utasítás.

A continue utasítás csak ciklusokban használható. A while ill. do-while ciklus belsejében az azonnali feltételvizsgálat végrehajtását eredményezi, for ciklus esetén pedig a ciklusváltozó módosítását. A switch utasításban hatástalan, ill. ha a switch egy ciklusban szerepel, akkor erre a ciklusra érvényesek a fent leírtak.

Az utóbbi két utasítást nem használjuk gyakran !