6.Bölüm

DÖNGÜLER (LOOPS)

Döngü adı verilen kontrol yapısı bir tümce blokunu bir kaç kez işletir. Döngü sayısının nasıl belirleneceği, döngünün tipine bağlıdır. C'de üç tip döngü vardır.

- while döngüsü
- for döngüsü
- do döngüsü

WHILE DÖNGÜSÜ

Bu döngü bir koşul doğru olduğu sürece tümce blokunu tekrarlar. Koşul geçerli olmadığı anda program döngüyü izleyen komutlara geçer. Koşul tümcesinin kullanımı ile aynıdır. Genel şekli;

```
while (koţul)
 blok1;
```

ÖRNEK

```
# include <stdio.h>
main( )
{
int a;
a=10;
while (a>0)
{
 a=a-1;
 printf("a'nın değeri= %d\n", a);
}
printf("döngü bitti\n");
}
```

SONSUZ DÖNGÜ

ÖRNEK

```
# include <stdio.h>
main( )
{
int a;
a=10;
while (a>0)
{
 printf("a'nın değeri= %d\n", a);        }
```

```
printf("döngü bitti\n");          }
```

a değişkeni hiç değişmediği için koşul hiç değişmez ve döngü tekrarlanıp durur.

ARTIMLAR VE EKSİLİMLER

Döngü kullanımında kontrol değişkenine 1 eklenir (ileri sayma) veya 1 çıkartılır. (geri sayma)

Aritmetik olarak a=a+1 veya a=a-1 biçiminde yapılır. Ancak C bu işlemler için daha kısa yollar sunar. İşlem işaretlerinden ++ ve - artım ve eksilim işaretleri olarak tanınırlar. ++ değişken 1 ekler, -- değişkenden 1 çıkarır. Yukarıdaki örnek şu şekilde yapılabilir di.

```
# include <stdio.h>
main( )
{
int a=10;
while (a>0)
{
a--;
printf("a'nın değeri= %d\n", a);        }
printf("döngü bitti\n");          }
```

Bu işlemlerin avantajı, bir eşitliğin içinde kullanılabilmeleridir. Her bir işlem iki biçimde kullanılabilir.

```
--a
a--
++a
a++
```

Sembol değişkenden önce gelirse artım veya eksilim işlemin bulunduğu tümceden önce yapılır. Sembol değişkenden sonra gelirse işlem tümceden sonra yapılır.

```
a=a-1;
printf("a'nın değeri= %d\n", a);
```

Bu aşağıdaki tek satıra dönüştürülebilir.

```
printf("a'nın değeri= %d\n", --a);
```

Aşağıdaki tümceyi ele alalım;

```
printf("a'nın değeri= %d\n", a--);
```

Bu aşağıdaki iki tümce ile aynı işi yapar;

```
printf("a'nın değeri= %d\n", a);
a=a-1;
```

NOT: Bu işlemler bulundukları eşitliğin bir parçası olmadıklarından diğer işlemlerden biraz farklı çalışırlar. Diğer tüm işlemlerden daha büyük bir önceliğe sahiptirler.

ÖRNEK

```
a=c++*12+saatlerin_toplamı;
```

Önceliği en yüksek işlem ++ olduğu için eşitlik şu duruma gelir.

```
a=c*12+saatlerin_toplamı;

c=c+1;
```

Burada c++ olduğu için a üzerinde bir etkisi yoktur.

```
A=++c*12+saatlerin_toplamı;
```

Burada ++c olduğundan ++ işleminin a üzerinde bir etkisi vardır.

c=a+++b; eşitliğinde belirsizlik vardır. C, artımı a değişkenine bağlar fakat istenmeyen durumlardan kaçınmak için

c=a+++b; eşitliği ya c=a++ +b; veya c=(a++) +b; şeklinde yazılır.

FOR DÖNGÜSÜ

For döngüsü program kontrolünü biraz daha basitleştirir. While döngüsü kadar esnek olmamasına rağmen anlaşılması ve kullanımı daha kolaydır. Genel bir kural olarak, bütün döngülerin üç ayrı parçası olmalıdır.

- ilk durum belirlenmelidir. (a=10;)
- durumu değiştirilmelidir. (a--;)
- döngünün sürüp sürmeyeceğinin belirlenmesi için durumu belirlemenin bir yolu olmalıdır. (a>0)

for döngüsünün kullanımı şöyledir.

```
for (ilk durum; koşul ; değiştirici tümceler)
 blok;
```

ÖRNEK

```
# include <stdio.h>
main( )
{
```

```
   int a;
   for (a=10; a>0; a--)
    printf("a'nın değeri %d\n", a);
   printf("döngü bitti");
   }

for(a=10; a<0;)
printf("a'nın değeri\n");
```

sonsuz döngü; tümcenin yanlış olduğu hemen görülebilir.

ÖRNEK

```
   # include <stdio.h>
   main( )
   {
   int a;
   printf("ilk değeri girin="); /*Başlangıç koşulunun girilmesi*/
   scanf("%d", &a);
   for(;a>0;a--)
    printf("a'nın değeri %d\n", a);
   printf("döngü bitti\n");
   }
```

ÖRNEK

```
   # include <stdio.h>
   main( )
   {
   char ch=0;
   while (ch!='q')
   {
   printf("döngü başlangıcı\n"); /* Bir miktar tümce */
   printf("bitirmek için q'ya devam etmek için herhangi bir tuşa basın;");
   ch=getch( );
   printf("\n");
   }
}
```

Bu program q harfi yazılana kadar açıklama satırı ile aynı blokta bulunan tümceleri tekrarlar. Yazılan tuşu okumak için getch( ) fonksiyonu kullanılır. getch( ) <ENTER> tuşuna basılmasını beklemez.

For döngüsü genellikle tekbir değişkenin değiştiği durumlarda kullanılır.

ÖRNEK

```
# include <stdio.h>
main( )
{
int a,b;
a=0;
b=100;
while (a!=b)
          {
           printf("ilk değer: \n"); scanf("%d", &a);
           printf("2.değer: \n"); scanf("%d", &b);
 }
printf("Bu sayılar eşittir\n");    }
```

Yukarıda verilen örneğin for döngüsü kullanılarak yapılması, bunun yazılması ve de anlaşılması daha zordur. While ile yapılan program tercih edilir. Döngünün bitiş noktasının program yazılırken bilindiği uygulamalarda for döngüsü kullanılır.

ÖRNEK

```
# include <stdio.h>
main( )
{
int a,b;
for (a=0, b=100;a!=b)
          {
           printf("ilk= %d\n"); scanf("%d", &a);
           printf("ikinci= %d\n"); scanf("%d", &b);
          }
printf("Bu sayılar eşittir\n");
}
```

ÖRNEK

```
for(i=0;i<10;i++)
```

Bu döngü tam olarak 10 defa işletilir ve yalnızca döngü tümcesine bakılarak bu durum anlaşılabilir. While döngüleri genellikle koşul değişkenlerinin bir kaç değişik şekilde değiştirilebildiği durumlarda kullanılır.

```
while(i<10 && ch!='q')
```

Bu döngünün kaç defa işletileceği ancak döngü içindeki tümcelere bakılarak anlaşılır.

DO DÖNGÜSÜ

Do döngüsü for ve while döngüsünde kontrol edilen bitiş koşullarını döngü tümceleri işletildikten sonra kontrol eder. Bunun en önemli etkisi bir do döngüsündeki tümcelerin en az bir kez işletilmesidir. Genel kullanımı şöyledir;

```
do
 blok;
while (koţul);
```

While döngüsünde olduğu gibi blok içindeki tümceler koşul doğru olduğu sürece iţletilirler. Koţul kontrol edilmeden önce blok içindeki tümceler bir defa iţletilir.

ÖRNEK

```
# include <stdio.h>
main( )
{
int ch;
do
 {
        printf("Bitirmek için q'ya basın:\n");
        ch=getch( );    }
while(ch!='q');
printf("döngü bitti\n");          }
```

ÖRNEK

```
# include <stdio.h>
main( )
{
do
        {
         printf("bitirmek için q'ya basınız\n");
         }
while(getch( )!='q');
printf("döngü bitti\n");          }
```