Bölüm 09

Döngüler

for döngüsü
do döngüsü
while döngüsü
foreach döngüsü

Belirli bir iş bir çok kez tekrarlanacaksa, programda bu iş bir kez yazılır ve döngü deyimleriyle istenildiği kadar tekrar tekrar çalıştırılabilir.

Bir döngüde, arka arkaya tekrarlanan deyimler döngü blokunu oluşturur. Bu deyimler birden çoksa { } bloku içine alınır. Bir döngü blokunda çok sayıda deyim olabileceği gibi, iç-içe döngüler de olabilir. Program akışının döngü blokunu bir kez icra etmesine döngünün bir adımı (bir tur) diyeceğiz.

C# dilinde döngü yapan dört ayrı yapı vardır. Aşağıdaki kesimlerde bu yapıları örneklerle açıklayacağız.

for Döngüsü

Bir deyim, önceden belirli olan sayıda tekrar edecekse, for döngüsünü kullanmak çok kolaydır. Önce basit bir örnekle başlayalım :

Aşağıdaki program aynı istenildiği kadar yazar.

ForLoop01.cs

```
using System;
namespace DenetimYapıları
{
    class ForLoop02
    {
        public static void Main()
        {
            for (int a = 5; a < 10; a++)
```

```
            {
                Console.WriteLine("C# ile programlama seçkin
programcılarin zevkidir.");
            }
        }
    }
}
```

Çıktı

```
C# ile programlama seckin programcilarin zevkidir.
C# ile programlama seckin programcilarin zevkidir.
C# ile programlama seckin programcilarin zevkidir.
C# ile programlama seckin programcilarin zevkidir.
C# ile programlama seckin programcilarin zevkidir.
```

Bu çıktıya dikkat edersek, for döngüsünün yapısının

```
            for (int a = 0; a < 10; a++)
            {

            }
```

blokundan oluştuğunu olduğunu görüyoruz. Döngü { } bloku içindeki deyimleri yürütür. Buna döngü bloku diyeceğiz. Burada `int a=0` döngü sayısını sayan sayaçtır, ilk değeri 0 dır. a <10 döngüye devam ya da dur diyecek mantıksal deyim (boolean) dir. a<10 olduğu sürece döngü tekrarlanacak, a>=5 olduğunda döngü duracaktır. a++ ise her döngünün her adımında sayacı 1 artıran deyimdir. Bu deyim, istenen sayıda adım atıldıktan sonra döngünün durmasını sağlar.

Şimdi bunlara göre, for döngüsünün söz dizimini yazabiliriz:

```
for (sayaç-ilk-değeri; döngü-koşulu; sayaç-değişimi)
{
deyimler
}
```

Bu yapıyı yukarıdaki örnekle karşılaştırarak söz dizimini kolayca algılayabilirsiniz. Zaten, aşağıdaki örnek programlarda da, konu yeterince tekrar edilecektir. Şimdilik şu özelikleri bilmek yeterlidir.

- Sayacın ilk değeri için döngünün ilk adımı (ilk tur) mutlaka çalışır.
- Sonraki adımların her birisi için önce döngü koşulu denetlenir, `true` değerini alıyorsa yeni tur atılır (döngü bloku işlenir).
- Döngü koşulu `false` değerini aldığında, program akışı döngünün dışına çıkar.

Elbette döngüleri kullanmaktaki amacımız, yukarıda yaptığımız gibi, aynı sözü defalarca söyletmek değildir. Onlarla harika işler yapabiliriz. O harika işlerin nasıl yapılabileceğini, aşağıdaki basit örneklerden öğreneceğiz.

ForLoop02.cs

```
// 5 den küçük pozitif tamsayıları yazar.
using System;
namespace DenetimYapıları
{
    class forLoop02
    {
```

```
        public static void Main()
        {
            for (int a = 0; a < 5; a++)
            {
                Console.WriteLine(a);
            }
        }
    }
}
```

Çıktı

```
0
1
2
3
4
```

Bu programda döngü blokunda bir tek Console.WriteLine(a) deyimi vardır. Böyle olduğunda { } parantezlerini kullanmayabiliriz.

```
            for (int a = 0; a < 5; a++)
            {
                Console.WriteLine(a);
            }
```

ForLoop03.cs

```
//10 'a kadar pozitif tamsayıları for döngüsü ile toplar
using System;
namespace DenetimYapıları
{
    class ForLoop03
    {
        public static void Main()
        {
            int sum = 0;
            for (int i = 0; i <= 10; i++)
            {
                sum = sum + i;

                Console.WriteLine("{0}ye kadar toplam: {1} ", i, sum);
            }
        }
    }
}
```

Çıktı

```
0ye kadar toplam: 0
1ye kadar toplam: 1
2ye kadar toplam: 3
3ye kadar toplam: 6
4ye kadar toplam: 10
5ye kadar toplam: 15
```

```
6ye kadar toplam: 21
7ye kadar toplam: 28
8ye kadar toplam: 36
9ye kadar toplam: 45
10ye kadar toplam: 55
```

break ve continue deyimleri

Bazı durumlarda, belli koşul oluşunca döngü adımları bitmeden döngüden çıkmak, bazı durumlarda da, döngünün bazı adımlarını işlememek gerekebilir. Bunu aşağıdaki iki deyimle yaparız.

break

Döngünün kesilmesine ve program akışının döngü blokunun dışına çıkmasına neden olur.

continue

Döngü bloku içinde kendisinden sonra gelen deyime geçmeden akışı durdurur, döngü başına gönderir ve döngünün bir sonraki adımının atılmasını sağlar.

`break` deyiminin etkisini aşağıdaki programda görebiliriz.

BreakKullanımı.cs

```
//döngü sayısını tamamlamadan döngüyü kesme yöntemini göstermektedir.
using System;
namespace DenetimYapıları
{
    class BreakKullanımı
    {
        public static void Main()
        {
            int faktoryel = 1;
            for (int n = 1; n <= 20; n++)
            {
                if (n == 6) break;
                faktoryel = faktoryel * n;
                Console.WriteLine("{0}! = {1} ", n, faktoryel);

            }
        }
    }
}
```

Çıktı

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

Bu programı irdeleyelim. For döngüsünün koşullarına bakınca şunu anlıyoruz: Sayaç n=1 den başlayıp n<=20 olana kadar birer artarak gidecek ve böylece döngü 20 kez tekrarlanacak. Bu algımız doğrudur, ancak döngü blokunun içine baktığımızda

```
if (n==8) break;
```

deyimini görüyoruz. `break` deyimi döngünün kesilmesini ve program akışının döngü blokunun dışına çıkmasına neden olur. O halde, döngü, başlıkta istenenin aksine n==8 olduğunda kesilecektir. Dolayısıyla program koşturulunca ilk 7 sayının faktöryelleri yazılır.

Continue deyimi, break gibi döngüyü kesmez, yalnızca o an yapılan tekrarı tamamlatmaz ve sonraki tekrar için döngü başına döner.

`continue` deyiminin etkisini, yukarıdaki programın benzeri olan aşağıdaki programda görebiliriz.

ContinueKullanımı.cs

```
/*
continue deyimi, o anda blokta program akışını durdurur,
döngü başına gönderir, döngüyü sonraki adımdan yürütür.
*/
using System;
namespace DenetimYapıları
{
    class ContinueKullanımı
    {
        public static void Main()
        {
            int faktoryel = 1;
            for (int n = 1; n <= 10; n++)
            {
                if (n == 5) continue;
                faktoryel = faktoryel * n;
                Console.WriteLine("{0}! = {1} ", n, faktoryel);

            }
        }
    }
}
```

Çıktı

```
1! = 1
2! = 2
3! = 6
4! = 24
6! = 144
7! = 1008
8! = 8064
9! = 72576
10! = 725760
```

Bu çıktıda 5! olmadığını görüyoruz. Bunun nedeni şudur:

```
if (n == 5) continue;
```

deyimi, döngünün beşinci adımını tamamlatmadan program akışını durdurmuş ve döngü başına göndererek döngünün 6-ıncı adımdan devamını sağlamıştır.

Break ve continue deyimlerini amacımızı gerçekleştirmek için çok farklı biçimlerde kullanabiliriz. Aşağıdaki program her ikisini birden kullanmaktadır.

BreakContinue01.cs

```
//For döngüsünde break ve continue kullanımı
```

```
using System;
class BreakContinueTest
{
    public static void Main()
    {
        for (int i = 0; i < 20; ++i)
        {
            if (i == 10)
                break;
            if (i == 5)
                continue;
            Console.WriteLine(i);
        }
    }
}
```

Çıktı

```
0
1
2
3
4
6
7
8
9
```

Çıktıdan görüldüğü gibi, döngü 5 'e ulaştığında continue araya girmektedir. Dolayısıyla i=5 için Hiçbir iş yapılmadan, i=6 ya geçilip devam edilmektedir.

Öte yandan, döngü 20 adımda tamamlanacak olmakla birlikte, i=10 da break ile karşılaşmaktadır. Bu adımda döngü kesilmekte ve döngü blokunun dışına çıkılmaktadır.

ForLoop03.cs

```
// break ve continue kullanımı
using System;
namespace DenetimYapıları
{
    class ForLoop03
    {
        public static void Main()
        {
            for (int i = 0; i < 20; i++)
            {
                if (i == 10)
                    break;

                if (i % 2 == 0)
                    continue;

                Console.Write("{0} ", i);
            }
            Console.WriteLine();
        }
    }
}
```

Çıktı

```
1     3     5     7     9
```

Bu programı irdeleyelim. For döngüsünün koşullarına bakınca şunu anlıyoruz: Sayaç i=0 dan başlayıp i<20 olana kadar birer artarak gidecek ve döngü 20 kez tekrarlanacak iken

```
if (i==10) break;
```

deyimi, i==10 olduğunda döngüyü kesmekte ve program akışının döngü blokunun dışına çıkarmaktadır. Dolayısıyla, döngü, başlıkta istenenin aksine i==10 olduğunda kesilecektir.

Blok içindeki

```
if (i%2==0) continue ;
```

deyimine baktığımızda şunu anlıyoruz: `i%2==0` ise, yani i sayısı 2 ile tam bölünüyorsa, program akışı sonra gelen deyimleri yapmadan döngü başına dönüp döngüyü yeni adım için tekrarlamaktadır. Bu ise şu anlama gelir: i çift sayı olduğunda

```
Console.Write("{0} ", i);
```

deyimi çalışmadan döngü başına gidilecektir. Öyleyse, i çift sayı olduğunda ekrana bir çıktı yazılmayacak, yalnızca tek sayı olduğunda yazılacaktır. O nedenle, program yalnızca tek sayıları yazmıştır.

while döngüsü

Belirli bir mantıksal deyim sağlandığı sürece, belirli bir işin tekrarlanması isteniyorsa, bu denetim yapısı kullanılır. Bu yapıda, tekrarlamanın kaç kez olacağını önceden bilmek gerekmez. Sözdizimi şöyledir:

```
while (boolean)
{
deyimler
}
```

WhileLoop01.cs

```
//0 dan 9 kadar tamsayıları while döngüsü ile yazar
using System;

using System;
namespace DenetimYapıları
{
    class WhileLoop01
    {
        public static void Main()
        {
            int sayaç  = 0;

            while (sayaç < 10)
            {
                Console.Write("{0}    ", sayaç);
                sayaç++;
            }
            Console.WriteLine();
        }
    }
}
```

Çıktı

```
0    1    2    3    4    5    6    7    8    9
```

Bu programda (sayaç < 10) olduğu sürece, döngü bloku dediğimiz { } blok içindeki deyimler arka arkaya tekrar edilir. Bu tekrarların her birine döngünün bir adımı diyoruz. He adımda döngü sayacı 1 artar (sayaç++). Sonunda sayaç==10 olur ve döngü biter, program akışı, varsa döngü blokundan sonraki deyimden devam eder.

Bazı işleri birden çok döngü yapısıyla gerçekleştirebiliriz. Bu durumlarda, programcı birisini tercih etme hakkına sahiptir. Örneğin, önceki kesimde 1 den 10 'a kadar tamsayıların toplamını for döngüsü ile bulmuştuk. Aynı işi while, do-while ve foreach döngüleriyle de yapabiliriz.

While02.cs

```
using System;
namespace DenetimYapıları
{
    class while02
    {
        public static void Main()
        {
            int n = 1;
            int toplam = 0;

            while (n <= 10)
            {
                toplam = toplam + n;
                Console.WriteLine("TOPLAM[1...{0}] = {1} ", n, toplam);
                n++;
            }


        }
    }
}
```

Çıktı

```
TOPLAM[1...1] = 1
TOPLAM[1...2] = 3
TOPLAM[1...3] = 6
TOPLAM[1...4] = 10
TOPLAM[1...5] = 15
TOPLAM[1...6] = 21
TOPLAM[1...7] = 28
TOPLAM[1...8] = 36
TOPLAM[1...9] = 45
TOPLAM[1...10] = 55
```

do-while döngüsü

Sözdizimi şöyledir:

```
do
        {
        deyimler
        }
while (boolean) ;
```

Bu yapı while döngüsüne benzer; ama önce döngü blokundaki, { }, deyimler icra edilir, sonra (boolean) denetlenir. True değerini alırsa, program akışı döngünün başına döner ve bir tur daha atar. Tekrar (boolean) denetlenir true değerini alırsa yeni tura geçilir. Bu süreç (boolean) false değerini alana kadar devam eder.

Görüldüğü gibi do-while döngüsünde döngünün ilk adımı mutlaka icra edilir. While döngüsünden farkı da budur.

For ve while döngüleriyle yaptığımız toplama işlemini, aşağıda görüldüğü gibi, do-while döngüsüyle de yapabiliriz.

DoWhile01.cs

```
using System;
namespace DenetimYapıları
{
    class DoWhile01
    {
        public static void Main()
        {
            int n = 1;
            int toplam = 0;

            do
            {
                toplam = toplam + n;
                Console.WriteLine("TOPLAM[1...{0}] = {1} ", n, toplam);
                n++;
            } while (n <= 10);


        }
    }
}
```

Çıktı

```
TOPLAM[1...1] = 1
TOPLAM[1...2] = 3
TOPLAM[1...3] = 6
TOPLAM[1...4] = 10
TOPLAM[1...5] = 15
TOPLAM[1...6] = 21
TOPLAM[1...7] = 28
TOPLAM[1...8] = 36
TOPLAM[1...9] = 45
TOPLAM[1...10] = 55
```

DoWhile02.cs

```
using System;
namespace DenetimYapıları
{
    class DoWhile02
    {
        public static void Main()
        {
            string istek;

            do
            {
                // Bir menü yazar
                Console.WriteLine("Adres Defteri \n");

                Console.WriteLine("A - Yeni adres gir");
                Console.WriteLine("B - Adres sil");
                Console.WriteLine("C - Adres güncelle");
                Console.WriteLine("D - Adres gör");
                Console.WriteLine("Q - Çıkış \n");

                Console.WriteLine("Seçiminiz: (A,B, C, D, Q): ");

                // Kullanıcının isteğini oku
                istek = Console.ReadLine();

                // Kullanıcının isteğini yap
                switch (istek)
                {
                    case "A":
                    case "a":
                        Console.WriteLine("Yeni adres eklemek mi
istiyorsunuz?");
                        break;
                    case "B":
                    case "b":
                        Console.WriteLine("Bir adres silecek misiniz?");
                        break;
                    case "C":
                    case "c":
                        Console.WriteLine("Bir adres güncelleyecek
misiniz? ");
                        break;
                    case "D":
                    case "d":
                        Console.WriteLine("Bir adres mi göreceksiniz?");
                        break;
                    case "Q":
                    case "q":
                        Console.WriteLine("Hoşça kal!");
                        break;
                    default:
                        Console.WriteLine("{0} geçerli bir seçim
değildir", istek);
                        break;
                }

                // DOS ekranını bekleterek çıktının okunmasını sağlar
                Console.Write("Devam için bir tuşa basınız...");
                Console.ReadLine();
                Console.WriteLine();
```

```
                // Çıkış tusuna (Q) basılana kadar döngüyü tekrarlatan
boolean
            } while (istek != "Q" && istek != "q");
        }
    }
}
```

Çıktı

```
Adres Defteri


A - Yeni adres gir
B - Adres sil
C - Adres güncelle
D - Adres gör
Q - Çıkış

Seçiminiz: (A,B, C, D, Q):
B
Bir adres silecek misiniz?
```

foreach Döngüsü

Bir array'in ya da collection' ın öğeleri üzerinde döngü yapar.

Aşağıdaki program bir string array'in bileşenlerini yazmaktadır.

Program

```
//foreach loop
using System;
class ForEach
{
    public static void Main()
    {
        string[] a = { "Ankara", "İstanbul", "İzmir" , "Van" };
        foreach (string b in a)
            Console.WriteLine(b);
    }
}
```

Çıktı

```
Ankara
İstanbul
İzmir
Van
```

Bu programda şuna dikkat edilmelidir.

`foreach` döngüsünün kaç kez döneceğini belirten ifade `(string b in a)` dir. Bu ifade string tipinden `b` değişkenini tanımlamakta ve `b` değişkeni `a` array'inin öğelerine eşleşmektedir. Dolayısıyla, `a` array'inin bütün bileşenleri sırayla yazılmaktadır.

Bu program bir tamsayıda kaç hane olduğunu sayar.

HaneSay.cs

```
using System;

namespace DenetimYapıları
{
    class HaneSay
    {
        static void Main(string[] args)
        {
            string s;
            int  n = 0;
            Int32 i;

            Console.WriteLine("En çok 10 haneli bir tamsayı giriniz:");
            s = Console.ReadLine();
            i = Int32.Parse(s);

            do
            {
                ++n;
                i = i / 10;
            } while (i > 0);
            Console.WriteLine("{0} sayında {1} hane vardır.", i, n);
        }
    }
}
```

Çıktı

```
En çok 10 haneli bir tamsayı giriniz :
1234567890
1234567890 sayında 10 hane vardır.
Devam etmek için bir tuşa basın . . .
```

`do-while` döngüsü, döngü blokunu en az bir kez çalıştırır. Ondan sonra `while` koşulunu denetler.

Aşağıdaki program 6! değerini bulur.

```
using System;

namespace Döngüler
{
    class ForEach02
    {
        static void Main(string[] args)
        {
            int n = 1, w = 1;
            foreach (char c in "Ankara") { w = w * n; n++; }
            Console.Write(w);
        }
    }
}
```