



Emrah Porgalı

<http://emrah.csharpurk.net>

emrah@csharpurk.net

while Döngüsü

01.02.2007

Programlama gerçekten de tekrarlardan ibarettir. Bizler nadiren sadece bir kez çalışan programlar yazarız. Yazdığımız çoğu program bir kez çalışır ve her çalışmasında hemen hemen aynı şeyi tekrarlar. Ancak her çalışmada doğru sonucu oluşturacak kadar ufak farklar vardır. Sıklıkla, bir programın içinde bile tekrarlamalar bulunur. Tabii yine ufak farklarla. Bu da bizim bu makalede inceleyeceğimiz konu. C# dili özellik olarak tekrarlama durumlarını icra eder. Biz bu tekrarlama durumlarına Döngü deriz.

Yeni kiraladığımız bir evin kirasının 1000 YTL olduğunu varsayalım. Ve her yıl kirayı %5 arttıracığımızı düşünelim. Önümüzdeki 10 yıl içinde vereceğimiz kira miktarının ne olacağını hesaplayan bir program yazalım.

```
using System;

class KiraHesabi
{
    static void Main()
    {
        decimal Kira = 1000, ArtisMiktari = 5.0m;
        Console.WriteLine("2006 yılındaki kira={0} YTL", Kira);
        Console.WriteLine("2007 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2008 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2009 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2010 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2011 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2012 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2013 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2014 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2015 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
        Console.WriteLine("2016 yılındaki kira={0} YTL", Kira *= (1 + ArtisMiktari / 100));
    }
}
```

Kopyalama ve yapıştırma mükemmel değil mi? Rakamların alt alta gelmesi için matematik hesabını WriteLine cümlesinin içinde yaptık. Her ifade Kira değişkeninin değerini artırıyor. 2005 yılı için ifadeyi yazdıktan sonra bu ifadeyi kopyaladım ve 9 kere yapıştırdım ve yılların yazıldığı rakamları değiştirdim. Sonucu aşağıdaki gibi elde ettik.

```
C:\WINDOWS\system32\cmd.exe
2006 yılındaki kira=1000 YTL
2007 yılındaki kira=1050,00 YTL
2008 yılındaki kira=1102,5000 YTL
2009 yılındaki kira=1157,625000 YTL
2010 yılındaki kira=1215,50625000 YTL
2011 yılındaki kira=1276,2815625000 YTL
2012 yılındaki kira=1340,095640625000 YTL
2013 yılındaki kira=1407,10042265625000 YTL
2014 yılındaki kira=1477,4554437890625000 YTL
2015 yılındaki kira=1551,328215978515625000 YTL
2016 yılındaki kira=1628,89462677744140625000 YTL
Devam etmek için bir tuşa basın . . .
```

decimal değişken kullandığımızdan dolayı oluşan fazla haneleri saymasak iyi çalışmış gibi görünüyor. Fakat eminim ki bu programda güzel bir şeylerin eksik olduğuna benle aynı fikirdesiniz. Sadece birkaç yıl için pek fazla kötü görünmüyor. Fakat daha fazla yıl için bir hesaplama yapacak olsak işler kötüye giderdi. Birbirinin aynı koddan sayfalar dolusu istemezsiniz. Ya bu kodları yazdıktan sonra küçük bir yanlışlığa rastlarsanız ne olacak?

Zorlama ile bazı problemler çözmek işe yarayabilir, genellikle döngü kullanmaktan daha kolaydır. C# 'ta en basit döngü **while** ifadesidir. **else** kısmı olmayan bir **if** ifadesine çok benziyor. **while** anahtar kelimesini bir parantezler içerisindeki bir **boolean** ifade takip eder. Bir çift küme parantezi bir veya birden fazla döngü kodunu takip eder.

```
while (boolean ifade)
{
    // döngünün içindeki diğer ifadeler
}
```

while ile **if** arasındaki farklar şunlardır:

if cümlesinde eğer **boolean** ifade doğru (**true**) ise küme parantezleri arasındaki ifadeleri sadece bir kez çalıştırır. **while** ise **boolean** ifadesi doğru olduğu müddetçe küme parantezleri arasındaki ifadeleri çalıştırmaya devam eder. Çalışma **boolean** ifade yanlış (**false**) olana kadara devam eder.

Değeri 55 olan bir **i** değişkeni olduğunu varsayalım. Şimdi bir **if** cümlesi yazalım

```
if ( i < 100 )
{
    Console.WriteLine ( i ) ;
}
```

Sonuç 55 olarak ekrana yazılacaktır.

Aynı ifadeyi **while** için yazalım.

```
while ( i < 100 )
{
    Console.WriteLine( i ) ;
}
```

```
}
```

Bu sefer sonuç aşağıdaki gibi olacaktır:

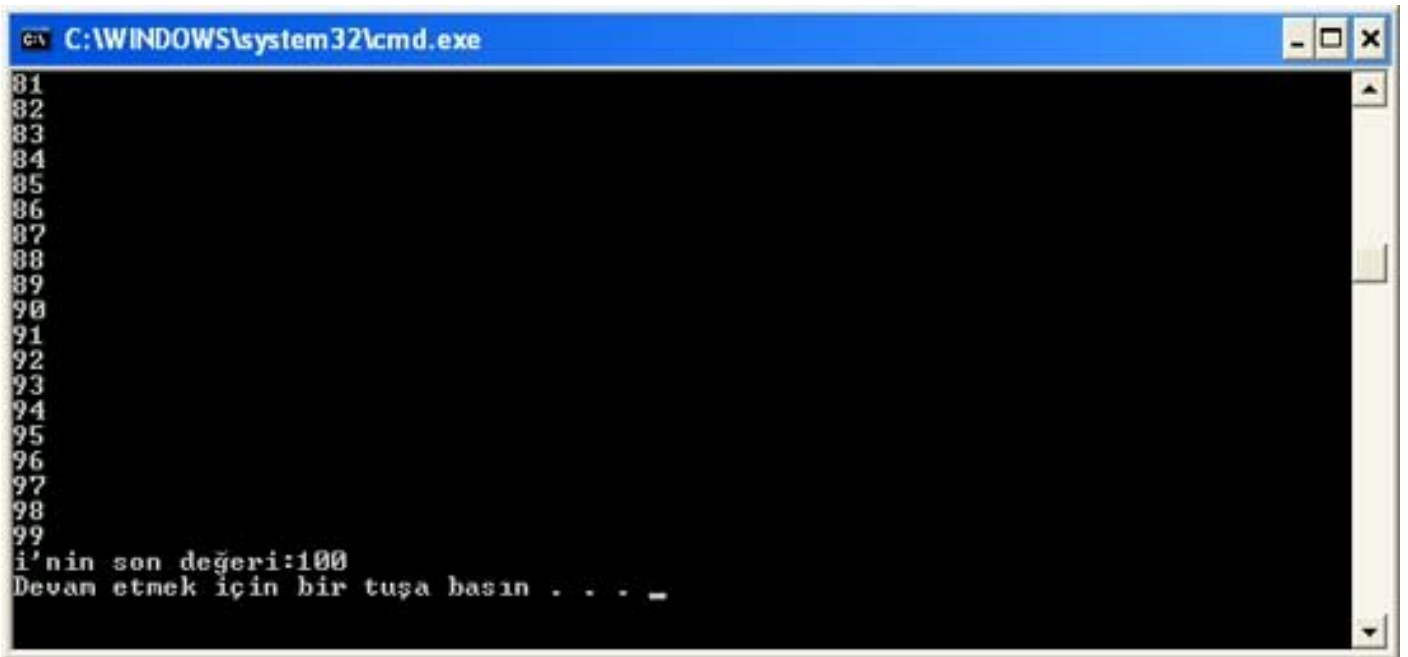
```
55  
55  
55  
55  
...
```

Sonsuz döngü olarak bilinen bu döngü devamlı çalışmaya devam edecektir. Fakat genellikle program bitmeden bu sonsuz döngüleri bitirmek isteyeceksin. Program sonsuz döngüye girdiğinde konsol penceresini kapatarak veya "Ctrl +C" tuşlarını kullanarak döngüyü durdurabilirsin. Tabii ki kesinlikle sonsuz döngü içeren uygulamalar var, fakat çoğu durumda küme parantezinin içindeki kodlar **boolean** ifadenin içindeki değişkeni değiştirirler.

Şimdi 100'e kadar sayan bir program yazalım.

```
using System;  
  
class CountUntill100 // 100'e kadar say  
{  
    static void Main()  
    {  
        int i = 0;  
        while (i < 100)  
        {  
            Console.WriteLine(i);  
            i++;  
        }  
        Console.WriteLine("i'nin son değeri:{0}", i);  
    }  
}
```

i'nin değeri 100'den küçük olduğundan, işletimin devamında **while** ifadesine gelindiğinde işletim **while** ifadesinin küme parantezlerini işleterek ilerleyecek. Küme parantezlerinin içindeki ifade **i**'nin değerinin yazacak ve değişkeni arttıracak. Ve işletim boolean ifadeye geri dönecek. **i** hala 100'dem küçük mü? Eğer öyleyse küme parantezinin içindeki ifade tekrar işletilecek. Bu işlemlerin her bir ilerlemesine iterasyon (yineleme) adı verilir. Programın çıktısı aşağıdaki gibidir.



```
C:\WINDOWS\system32\cmd.exe  
01  
02  
03  
04  
05  
06  
07  
08  
09  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
i'nin son değeri:100  
Devam etmek için bir tuşa basın . . . _
```

99 gösterildikten sonra **i**'nin değeri 100 e arttırılır. Programın devamı **boolean** ifadeye döner. **i**'nin değeri 100'den küçük mü? Hayır. Programın işletimi while ifadesinden sonraki program kodu ile devam eder. Şimdi **i** değişkenin değeri 100'e eşit oldu. En son olarak program "i'nin son değeri 100" yazar.

i'nin değeri 100'e ulaştığında (veya 100 değerinin üzerine ulaştığında) küme parantezleri içerisindeki ifadeler işletilmeyecek.

Aşağıda bu programda kullandığımız while döngüsünün başka bir varyasyonu gösterilmektedir.

```
while ( i < 100 )
{
    i++;
    Console.WriteLine(i);
}
```

i'nin değeri **Console.WriteLine** çağrısından önce arttırıldığı için bu döngü 1 den 100 e kadar olan sayıları gösterir. Döngü **i** değeri 100'e ulaştığında hemen durmaz. WriteLine çağrısından sonra işletim döngünün başına döner ve bundan sonra **boolean** ifade tekrar kontrol edilir.

Not: Küme parantezleri arasında sadece 1 ifade varsa küme parantezlerini kullanmanıza gerek yoktur.

Şimdi **i** değişkenin değerini **Console.WriteLine** çağrısının içinde arttırarak küme parantezlerini kullanma derdinden de kurtulalım.

```
while ( i < 100 )
    Console.WriteLine(i++);
```

Sonek arttırma operatörü değişkenin değerini ekranda gösterildikten sonra arttırır. Bu örnekte program 0'dan 99'a kadar sayıları ekranda gösterir. Eğer önek arttırma operatörünü kullanırsak;

```
while ( i < 100 )
    Console.WriteLine(++i);
```

değişken ekranda gösterilmeden önce arttırılacak ve program ekranda 1'den 100'e kadar olan sayıları gösterecek.

Aynı zamanda **i** değişkenini **boolean** ifadenin içinde de arttırabiliriz. Örneğin;

```
while ( i++ < 100 )
    Console.WtiteLine(i);
```

Sonek artırım operatörü **i** değişkenin değerini **boolean** ifadede kullanıldıktan sonra, **Console.WriteLine** çağrısından önce artırıyor. Program ekranda 1'den 100'e kadar sayıları gösteriyor. Aynı ifadenin önekli versiyonu;

```
while ( ++i < 100 )
    Console.WriteLine(i);
```

Şimdi ise program ekranda 1'den 99'a kadar sayıları görüntülüyor. **i** değişkenin değeri 100'e arttırıldığında **boolean** ifaden yanlış döner ve programın işletimi while döngüsünden bir sonraki ifadede devam eder.

while döngü gövdesindeki bu farkları özellikle vurguluyorum. Problem genellikle **while** döngüsünün başlangıç ve bitiş sınırlarının yönetiminde ortaya çıkar. **boolean** ifadedeki değişkeni nerede ve nasıl değiştireceğini dikkatlice belirlemen gerekiyor. Boolean ifadede kullanılan < yerine <= ve > yerine >= ifadelerini de kullanarak döngünün sonunda ek bir döngü sağlayabilirsin.

Şimdi kira hesabı yapan programımızı tekrar yazalım.

```
using System;

class KiraHesabi
{
```

```

static void Main()
{
    int BaslangicYili = 2006;
    int BitisYili = 2016;
    decimal Kira = 1000;
    decimal ArtisMiktari = 5.0m;

    while (BaslangicYili <= BitisYili)
    {
        Console.WriteLine("{0} {1}", BaslangicYili, Kira);
        Kira *= (1 + ArtisMiktari / 100);
        Kira = Decimal.Round(Kira, 2);
        BaslangicYili++;
    }
}

```

Bu hesaplama biraz deęişik. Kirayı ve yılı ekrana yazdıktan sonra program gelecek yılın kirasını hesaplıyor ve **decimal** olarak tanımlanmış kira deęerinin virgülden sonraki kısmını 2 basamak kalana kadar yuvarlıyor. Programın çıktısı aşıęıdaki gibi olur.

```

C:\WINDOWS\system32\cmd.exe
2006 1000
2007 1050,00
2008 1102,50
2009 1157,62
2010 1215,50
2011 1276,28
2012 1340,09
2013 1407,09
2014 1477,44
2015 1551,31
2016 1628,88
Devan etmek için bir tuşa basın . . . _

```

Program 2016 yılındaki son kirası hesapladıktan sonra, Kira ve BaslangicYili deęerlerini arttırarak bir sonraki döngüye hazırlıyor. Kirayı hiçbir zaman kullanmayacak olmamıza rağmen hesaplıyor. Bu örnekte çok fazla bir yük oluşturmuyor ama daha büyük programlarda problem olabilir.

Programın çıktısına baktığımızda programın 2006 dan 2011 e kadar olanları kiralari hesaplamak için 11 tane **WriteLine** çağırısı yaptığını fark ederiz. Fakat program bu yıllar arasındaki kiralari hesaplamak için 10 kez hesap yapmaya ihtiyaç duyar. Bunun için while döngüsünün dışında fazladan bir **WriteLine** çağırısı yaparak bir çözüm oluşturabiliriz.

```

using System;

class KiraHesabi
{
    static void Main()
    {
        int BaslangicYili = 2006;

```

```

int BitisYili = 2016;
decimal Kira = 1000;
decimal ArtisMiktari = 5.0m;

while (BaslangicYili < BitisYili)
{
    Console.WriteLine("{0} {1}", BaslangicYili, Kira);
    Kira *= (1 + ArtisMiktari / 100);
    Kira = Decimal.Round(Kira, 2);
    BaslangicYili++;
}
Console.WriteLine("{0} {1}", BaslangicYili, Kira);
}
}

```

Yeni program çıktısı aşağıda.

```

C:\WINDOWS\system32\cmd.exe
2006 1000
2007 1050,00
2008 1102,50
2009 1157,62
2010 1215,50
2011 1276,28
2012 1340,09
2013 1407,09
2014 1477,44
2015 1551,31
2016 1628,88
Devam etmek için bir tuşa basın . . . _

```

Dikkat ederseniz **boolean** ifadede ki içindeki \leq karşılaştırmasını $<$ olarak değiştirdim. Böylelikle döngünün içinde en son olarak 2015 yılı için hesap yapılacak. Döngünün altındaki **WriteLine** çağrısı döngünün içinde önceden hesaplanan 2016 yılının kirasını ekrana yazdırıyor.

Bir diğer varyasyon ise **Console.WriteLine** ile döngüden önce ilk yılı ve kirasını ekrana yazdırıp daha sonra bu değerleri döngünün içinde arttırarak yazma işlemi yapmak. Yani hesaplama döngünün başında yapılıyor ve **WriteLine** çağrısı döngünün en altında yapılıyor.

```


using System;

class KiraHesabi
{
    static void Main()
    {
        int BaslangicYili = 2006;
        int BitisYili = 2016;
        decimal Kira = 1000;
        decimal ArtisMiktari = 5.0m;
    }
}

```

```
Console.WriteLine("{0} {1}", BaslangicYili, Kira);
while (BaslangicYili < BitisYili)
{
    BaslangicYili++;
    Kira *= (1 + ArtisMiktari / 100);
    Kira = Decimal.Round(Kira, 2);
    Console.WriteLine("{0} {1}", BaslangicYili, Kira);
}
}
```

Yine çıktısı aşağıda görülüyor.



```
C:\WINDOWS\system32\cmd.exe
2006 1000
2007 1050,00
2008 1102,50
2009 1157,62
2010 1215,50
2011 1276,28
2012 1340,09
2013 1407,09
2014 1477,44
2015 1551,31
2016 1628,88
Devam etmek için bir tuşa basın . . .
```

Bu iki çözüm de ideal değil çünkü ikisi de birbiri ile aynı **WriteLine** çağrısı içermektedir. Birini değiştirdiğinde diğerini de değiştirmek zorundasın.

Biz bu örnekte olduğu gibi döngünün içinde 11 **WriteLine** çağrısı ve 10 kez de hesap yapmak istiyorsak döngünün ortalarında bir yerde döngüden çıkmamız gerekiyor. **break** olarak adlandırılan ifade ile döngünün ortasında döngüden çıkabilirsin. Bu ifade her zaman aşağıdaki şekilde kullanılır.

```
break;
```

break ifadesini noktalı virgül takip eder. Bu ifade kullanıldığında döngüden çıkmaya ve programın döngüden bir sonraki ifadeden devam etmesine neden olur. Genellikle **break** ifadesi bir **if** bloğu ile kullanılır.

```
if ( BaslangicYili == BitisYili )
    break;
```

Kira hesabının bir diğer versiyonu aşağıdaki gibi yazıldığında gereksiz bir hesaplamadan bizi kurtarır.

```
using System;

class KiraHesabi
```

```

{
    static void Main()
    {
        int BaslangicYili = 2006;
        int BitisYili = 2016;
        decimal Kira = 1000;
        decimal ArtisMiktari = 5.0m;

        while (BaslangicYili <= BitisYili)
        {
            Console.WriteLine("{0} {1}", BaslangicYili, Kira);
            if (BaslangicYili == BitisYili)
                break;

            Kira *= (1 + ArtisMiktari / 100);
            Kira = Decimal.Round(Kira, 2);
            BaslangicYili++;
        }
    }
}

```

Şimdi while döngüsündeki **boolean** ifadeyi **true** kullanarak programın başka bir versiyonunu yazalım. Unutmayın ki **break** olmadan bu bir sonsuz döngü olacaktır.

```

using System;

class KiraHesabi
{
    static void Main()
    {
        int BaslangicYili = 2006;
        int BitisYili = 2016;
        decimal Kira = 1000;
        decimal ArtisMiktari = 5.0m;

        while (true)
        {
            Console.WriteLine("{0} {1}", BaslangicYili, Kira);
            if (BaslangicYili == BitisYili)
                break;

            Kira *= (1 + ArtisMiktari / 100);
            Kira = Decimal.Round(Kira, 2);
            BaslangicYili++;
        }
    }
}

```

break gibi while döngüsü içinde kullanılan bir diğer atlama komutu da **continue**'dur. Genellikle **if** karar yapısı ile tetiklenir. Bu ifade döngü içindeki arta kalan ifadenin atlanmasını sağlar. Ve program bir sonraki iterasyondan devam eder.

Örneğin, kira hesabı programımızda sonu 0 ile biten yıllarda kiranın artmamasını yani bir önceki yıl ile aynı kalmasını istediğimizi farz edelim. Yani 2010 yılında ki kiranın 2009'daki kira ile aynı kalmasını istediğimizi düşünelim.

Şimdi bir önceki yazdığımız orijinal koda bir bakalım.

```

while ( BaslangicYili<=BitisYili )
{
    Console.WriteLine("{0} {1}", BaslangicYili,Kira);

if ( BaslangicYili == BitisYili )
    break;

```

```
        Kira *= ( 1 + ArtisMiktari/100);
        Kira = Decimal.Round(Kira,2);
        BaslangicYili++;
    }
```

Bu kodu sihirli 2010 yılı için **if** ifadesi ile yazmanın bir yolu aşağıdaki gibi olabilir.

```
while ( BaslangicYili<=BitisYili )
{
    Console.WriteLine("{0} {1}", BaslangicYili,Kira);
}
if ( BaslangicYili != BitisYili )
{
    Kira *= ( 1 + ArtisMiktari/100);
    Kira = Decimal.Round(Kira,2);
}
BaslangicYili++;
}
```

if ifadesi 2009 yılının kirası gösterildikten sonraki artışları engeller. Bu nedenle de 2009 ile 2010 yılları arasında kira artışı meydana gelmez. Bu işlemi aynı zamandan **if** ile birlikte **continue**

```
while ( BaslangicYili<=BitisYili )
{
    Console.WriteLine("{0} {1}", BaslangicYili,Kira);
    BaslangicYili++;
}
if ( BaslangicYili ==2010 )
    continue;

    Kira *= ( 1 + ArtisMiktari/100);
    Kira = Decimal.Round(Kira,2);
}
```

BaslangicYili değişkenin artırılmasını döngünün başına aldım ve **if** ifadesini bunun devamına yazdım. Eğer BaslangicYili değişkeni 2010'a ulaşırsa **continue** ifadesi döngünün bir sonraki iterasyondan devam etmesini sağlayacak. Döngünün için deki son iki ifade bu şekilde atlanmış oluyor.

while döngüsünün bir diğer çeşidi de do döngüsü. while döngüsünün genel ifadesini bir kez daha hatırlayalım.

```
while ( boolean ifade )
{
    // while döngüsünün gövdesi
}
```

Şimdide do döngüsünün genel ifadesini tanıyalım. do döngüsü aynı zamanda while komutunu da içeriyor.

```
do
{
    // do döngüsünün gövdesi
}
while ( boolean ifade );
```

do döngüsün en sonunda noktalı virgül olduğuna dikkat edin. while döngüsünde olduğu gibi do döngüsünde de **break** ve **continue** ifadelerini kullanabilirsiniz.

do döngüsünün yapısı **boolean** ifadeyi döngünün başında değil de sonunda değerlendirir. do döngüsünde gövde **boolean** ifadeyi dikkate alınmadan en az kez çalışır.

do döngüsü kullanıcıdan giriş alan programlarda kullanışlıdır. Şimdi toplama yapan basit bir program yazalım.

```

using System;

class TopamaProgrami
{
    static void Main()
    {
        string Cevap;

        do
        {
            Console.Write("Birinci sayıyı gir: ");
            double d1 = Double.Parse(Console.ReadLine());

            Console.Write("ikinci sayıyı gir: ");
            double d2 = Double.Parse(Console.ReadLine());

            Console.WriteLine("Toplam : {0}", d1 + d2);

            Console.WriteLine("Başka bir toplama yapmak ister misin?(e/h)");
            Cevap = Console.ReadLine().Trim();

        } while (Cevap.Length > 0 && Cevap.ToLower()[0] == 'e');
    }
}

```

Buradaki **string** tipindeki Cevap değişkeni neden while bloğu içinde tanımlanmadı diye düşünebilirsiniz. Bunun nedeni; değişkenin bloğun altındaki while ifadesinin içindeki boolean ifadenin içinde kullanılmasıdır.

Bu programın anahtar kısmı do döngüsünün altına yakın yerde ortaya çıkar. Buradaki bir komut kullanıcıya bir tane daha hesaplama yapıp yapmayacağını sorar ve string tipindeki Cevap değişkeni bu cevabı saklar. while ifadesinin içindeki **boolean** ifade ilk önce Cevap değişkeninin uzunluğunun 0'dan büyük olup olmadığını kontrol eder. Eğer öyleyse Cevap değişkeninin büyük harflerle yazılmış olmasına karşın değişken küçük harflere çevriliyor ve ilk harfinin 'e' olup olmadığına bakıyor. Eğer ilk karakter 'e' ise programın işletimi döngünün en başına dönecek ve oradan devam edecek.

boolean ifadenin içindeki **Cevap.Length** ifadesi kullanıcının sorulan soruya karşılık olarak basitçe **enter** tuşuna basıp basmadığını kontrol etmek için kullanılıyor. Eğer kullanıcı cevap için sadece **enter**'e basmış ise **Console.ReadLine** boş bir **string** döndürecek ve bu **string**'in ilk karakterine ulamaya çalışmak kural dışı bir durum oluşturacaktır. Kullanıcı 'Y' veya 'y' ile başlayan bir şeyler yazdığında ki yazdıklarında boşluk bıraksa bile program devam edecek. Kullanıcıdan cevabını istediğimizde kullandığımız **Trim()** fonksiyonu boşluklardan kurtulmamızı sağlıyor.

Tabii ki program eğer kullanıcı bizim sorduğumuz iki komuta doğru yanıtları vermez ise yani sayıları istediğimiz ve cevabı istediğimiz komutlara diğer tipler değişkenler yazarsa program hata verecektir. Bu hataları nasıl halledeceğimizi başka bir makalede anlatacağız.

Bazen aşağıdaki gibi hatalar yapabiliriz.

```

while ( i < 100 )
{
    // diğer ifadeler
}
while ( i < 101 )

```

Bu örnekte ilk önce bir while ifadesi yazıyoruz. Daha sonra do döngüsü oluşturduğumuzu düşünüp döngünün sonuna bir while daha yazıyoruz.

```

while ( i < 100 )
{
    // diğer ifadeler
}

```

Yukarıdaki kısımda **i** değişkeni döngü bittiğinde muhtemelen 100 değerini alacak olduğundan ikinci kısımdaki

```
while ( i < 101 )
```

ifadesi sonsuz bir döngü olacak. Bu program asılı kalacakmış gibi görünüyor. Ve muhtemelen bir çıktı da görüntülenemeyecek.

Ne zaman bir konsol programı asılı kalırsa **ctrl+c** tuşlarını kullanarak programı durdurabilirsin. Bundan sonra kaynak kodunu tekrar incelemeli ve sonsuz döngü nerede oluşuyor bulmaya çalışmalısın. Eğer kodunda bununla ilgili bir kanıt bulamıyorsan **while** döngüsünün başına ve sonuna birkaç tane "Burası döngünün girişi", "Döngü bitiyor" gibi ifadeler içeren **WriteLine** çağrısı koyabilirsin. Böylelikle programı çalışırken inceleyebilirsin. Bu şekilde fazladan ihtimalleri sınırlamış olursun.

Döngülerin bir diğer yaygın kullanım alanı da dizilerle birlikte kullanımıdır. Şimdi önceden girilen bir sayı dizisinin sayılarının karelerini ve küplerini bula bir programı while döngüsü ile yapalım.

```
using System;
class KareKub
{
    static void Main()
    {
        int[] SayiDizisi = {10,11,12,13,14,15};

        int i = 0 ;

        while ( i < SayiDizisi.Length )
        {
            Console.WriteLine("{0} {1,6} {2,6} " ,SayiDizisi[i], Math.Pow(SayiDizisi[i],3),
            Math.Pow(SayiDizisi[i],3));
                i++;
        }
    }
}
```

Burada girilen sayı dizisinin içindeki sayıları karelerini ve küplerini alan bir programı yazdık. Dizinin eleman sayısı kadar dönen bir döngü oluşturduk. Programın çıktısı da aşağıdaki gibi oluyor



```
C:\WINDOWS\system32\cmd.exe
10    100    1000
11    121    1331
12    144    1728
13    169    2197
14    196    2744
15    225    3375
Devan etmek için bir tuşa basın . . .
```