

Programlamada kullanılan temel yapılardan biri döngülerdir. İlk yıllarımda Hata yattığımda genellikle döngüleri kullanırken hata yapardım. Bundan dolayı Debug konusunda bu dokümanda inceleyeceğiz. Yapay zekâdan ve internetten faydalanmayı unutmayın

# Döngüler ve Debug

Debug = Hata nerede lan?

Oğuzhan Karagüzel

16.12.2024

## İçindekiler

ŞEKİLLER DİZİNİ.....	2
TABLOLAR DİZİNİ.....	3
GİRİŞ .....	4
C#'TA ÖNERMELER .....	5
BOOL VERİ TİPİ.....	5
MANTIKSAL OPERATÖRLER VE BOOL SONUÇLARI.....	5
Mantıksal Operatörler .....	5
BOOL VE OPERATÖRLERLE ÖRNEK KULLANIM .....	7
Örnek 1: Yaş Kontrolü .....	7
Örnek 2: Kullanıcı Girişi.....	7
C#'TA DÖNGÜ .....	8
WHILE DÖNGÜSÜ .....	8
DEBUG .....	10
SAYAÇLAR VE DİZİLER İLE DÖNGÜ.....	12
FOR DÖNGÜSÜ .....	14
FOREACH DÖNGÜSÜ .....	16
FOR VE FOREACH KARŞILAŞTIRMASI .....	17
FOR VE FOREACH KULLANIMINDA ÖNEMLİ NOKTALAR .....	18
ÖDEV.....	18
TEST .....	18
CEVAP ANAHTARI.....	20

## ŞEKİLLER DİZİNİ

Şekil 1 İnsan Çalışma Döngüsü .....	4
Şekil 2 While Döngüsü .....	8
Şekil 3 "++" operatörü kullanımı.....	9
Şekil 4 "++" operatörü kullanımı programı çıktısı. ....	9
Şekil 5 Kod sırasını değiştirme. ....	10
Şekil 6 Kod sırası değiştirme program çıktısı.....	10
Şekil 7 Debug için hazırlık. ....	11
Şekil 8 Debug süreci.....	12
Şekil 9 Öğrenci notları sıralama programı .....	13
Şekil 10 Öğrenci notları sıralama programı çıktısı .....	13
Şekil 11 Öğrenci notları sıralama programı çıktısı 2. Adım .....	13
Şekil 12 Index out of range exception .....	14
Şekil 13 Array'lerde length (uzunluk) özelliği.....	14
Şekil 14 Dizi uzunluğu ile dizinin son elemanını yazdırmak. ....	14
Şekil 15 For döngüsü ile öğrenci notlarını yazdırmak. ....	15
Şekil 16 For döngüsü ile öğrenci notlarını yazdırma programı çıktısı.....	15
Şekil 17 foreach döngüsü ile öğrenci notlarını yazdırma. ....	17
Şekil 18 foreach döngüsü ile öğrenci notları yazdırma program çıktısı .....	17

## TABLÖLAR DİZİNİ

Tablo 1 Karşılaştırma Operatörleri Tablosu.....	7
Tablo 2 For - Foreach döngüsü karşılaştırması.....	17

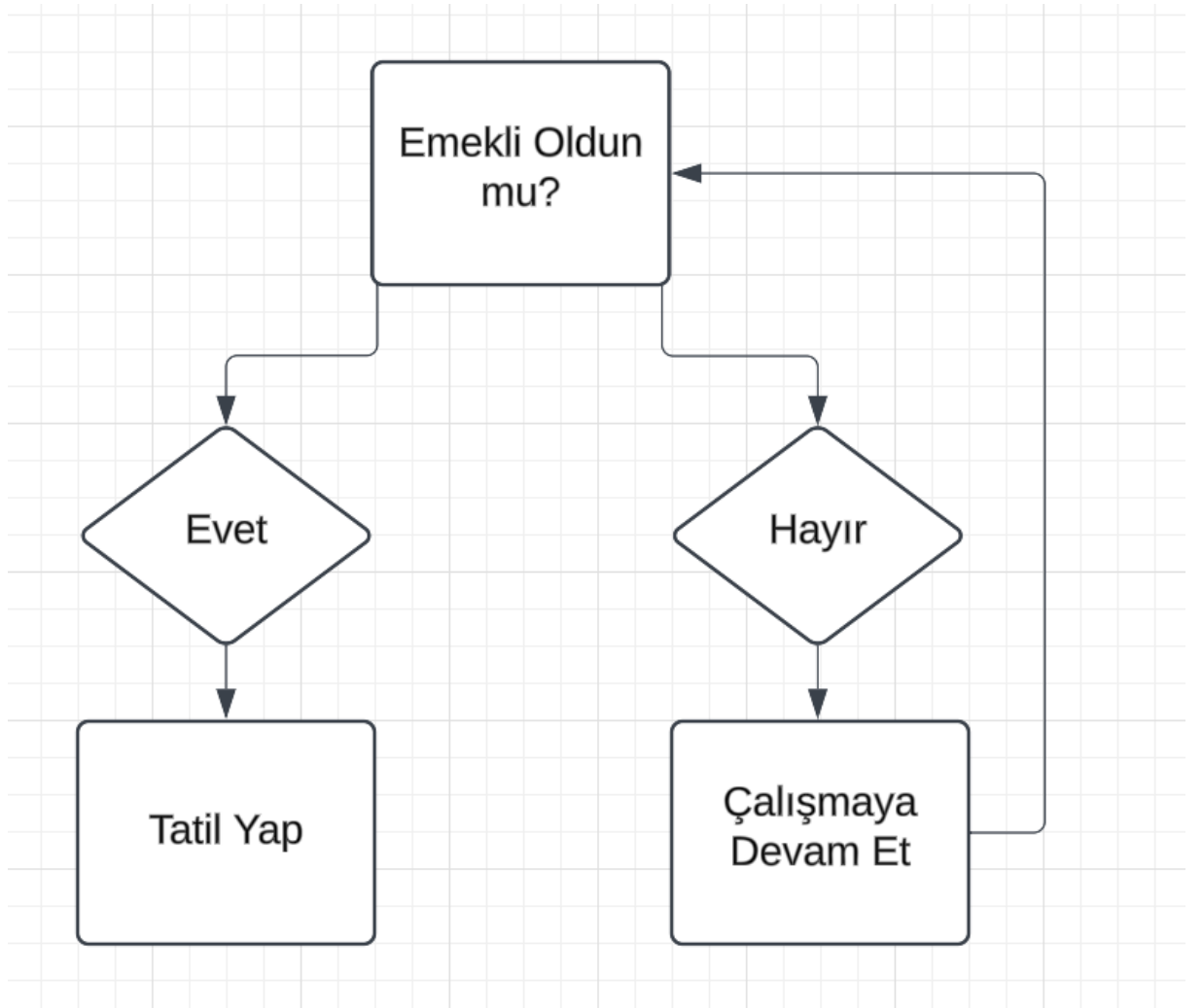
## GİRİŞ

Varsayalım ki belirli bir işlemi tekrar tekrar yapmanız lazım. Örnek: Cilala parlat, Cilala parlat gibi. Peki ne zamana kadar? Bu işlemi ne kadar tekrarlayacaksınız. İstediginizi elde edene kadar. Ya da cevap bulunana kadar.

Cevabın bulunup bulunmadığını ya da istediğinizin olup olmadığını kontrol edebilirsiniz. Cevap Bulundu mu? Evet ya da hayır. İstedğim oldu mu? Evet Ya da Hayır.

Evet ise tekrarlamayı durdur. Hayır ise tekrarlamaya devam et.

Bu şekilde basitçe anlatıldığında sanki bir robota komut veriyormuş gibi bir hissi var. Ancak insan oğlunun hayatı tamamen bu şekilde. Robotlarla oldukça fazla ortak yanımız var.



Şekil 1 İnsan Çalışma Döngüsü

Şimdi bu işleme biraz daha soyut bir biçimde bakalım. Çalışmaya devam et başlı başına çok büyük bir işlem. Kendi içerisinde daha detaylı bir sürü işlem barındırmakta. Her mesai bitiminde Kendinize sorarsınız. Emekli oldum mu? Bunu soru olarak sormak yerine daha önceki derslerden de hatırlayacağınız üzere önerme olarak ta sunabilirsiniz. Emekliyim. Doğru ya da yanlış. Yanlış ise çalışmaya devam et. Doğru ise tatil yap.

Peki neden önerme? Bilgisayar sorulardan veya cevaplarından anlamaz. Bilgisayar önermelerden anlar. Daha doğrusu önermelerden de anlamaz. Ancak önermeleri elektronik devreler kullanıp simüle edebiliyoruz.

Anlayacağınız üzere Döngülerde bakmamız gereken iki durum var. İlk olarak sürekli tekrarlanacak olan işlemi tanımlamak. İkincisi ise önermemizi tanımlamak.

Kod yazmayı bir önceki dersten öğrendik. Bu doküman boyunca kod yazmaya devam edeceğiz. Ancak hadi şu önermeleri tekrar edelim.

## C#'TA ÖNERMELER

**Önermeler**, doğru (**true**) veya yanlış (**false**) şeklinde değerlendirilen ifadelerdir. Bu tür ifadeler, C# dilinde temel olarak **bool** (boolean) veri tipi ile ifade edilir.

### BOOL VERİ TİPİ

C# dilinde **bool** veri tipi yalnızca iki değer alabilir:

- **true** (doğru)
- **false** (yanlış)

**Bool** tipindeki değişkenler genellikle karar mekanizmalarında ve mantıksal ifadelerde kullanılır.

#### Örnek:

```
bool isActive = true; // Bu değişken "doğru" değerini alır.
```

```
bool isCompleted = false; // Bu değişken "yanlış" değerini alır.
```

### MANTIKSAL OPERATÖRLER VE BOOL SONUÇLARI

C# dilindeki **mantıksal operatörler**, bool tipinde bir sonuç döndürür. Bu operatörler, bir veya birden fazla önermeyi birleştirerek **true** ya da **false** sonucunu üretir.

#### Mantıksal Operatörler

##### 1. VE (&&) Operatörü

- İki önermenin **her ikisi de doğru** ise sonuç **true** döner.
- Aksi takdirde **false** döner.

```
bool condition1 = true;
```

```
bool condition2 = false;
```

```
bool result = condition1 && condition2; // false
```

```
Console.WriteLine(result); // Sonuç: false
```

## 2. VEYA (||) Operatörü

- İki önermeden **en az biri doğru** ise sonuç **true** döner.
- Her iki önerme de yanlışsa sonuç **false** olur.

```
bool condition1 = true;
```

```
bool condition2 = false;
```

```
bool result = condition1 || condition2; // true
```

```
Console.WriteLine(result); // Sonuç: true
```

## 3. DEĞİL (!) Operatörü

- Bir önermenin değerini tersine çevirir.
- **true** ise **false**, **false** ise **true** döner.

```
bool isAvailable = true;
```

```
bool result = !isAvailable; // false
```

```
Console.WriteLine(result); // Sonuç: false
```

## Karşılaştırma Operatörleri

C#'ta **karşılaştırma operatörleri** iki değeri karşılaştırır ve bool tipinde bir sonuç döndürür. Bunlar genellikle mantıksal operatörlerle birlikte kullanılır.

Tablo 1 Karşılaştırma Operatörleri Tablosu

Operatör	Anlamı	Örnek	Sonuç
==	Eşittir	5 == 5	true
!=	Eşit Değil	5 != 3	true
>	Büyüktür	7 > 5	true
<	Küçüktür	3 < 8	true
>=	Büyük Eşittir	5 >= 5	true
<=	Küçük Eşittir	3 <= 2	false

## BOOL VE OPERATÖRLERLE ÖRNEK KULLANIM

### Örnek 1: Yaş Kontrolü

Bir kişinin belirli bir yaş aralığında olup olmadığını kontrol eden bir program yazalım:

```
int age = 25;
```

```
bool isAdult = age >= 18 && age <= 65; // Yaş 18 ile 65 arasında mı?
```

```
Console.WriteLine(isAdult); // true
```

### Örnek 2: Kullanıcı Girişi

Kullanıcının hem kullanıcı adı hem de şifreyi doğru girip girmediğini kontrol eden bir program:

```
string username = "oguzhan";
```

```
string password = "12345";
```

```
bool isUsernameCorrect = username == "oguzhan";
```

```
bool isPasswordCorrect = password == "12345";
```

```
Console.WriteLine($"Giriş Yapabilir: { isUsernameCorrect && isPasswordCorrect }");
```



Mantıksal operatörleri kısaca hatırladığımıza göre Döngülere başlayabiliriz. Ancak önceki derslerden bit bazında mantıksal operatörleri de tekrar etmenizi öneririm. İlerleyen derslerde bu operatörleride kullanacağız.

## C#'TA DÖNGÜ

Döngüler, bir işlemi birden fazla kez gerçekleştirmek için kullanılan yapılar olup, tekrarlama koşuluna bağlı olarak çalışır. İlk olarak while döngüsüne inceleyeceğiz. While döngüsü döngü önermesini kontrol eder. Eğer önerme true ise döngüye devam eder. Değilse döngüden çıkılır.

### WHILE DÖNGÜSÜ

**While** döngüsü, belirtilen koşul doğru (**true**) olduğu sürece çalışan bir döngüdür. Koşul sağlanmazsa döngüye hiç girilmez. Döngü çalışırken koşul değişir ise bir sonraki döngü önermesi kontrolünde döngüden çıkılır.

```
while (koşul)
{
    // Tekrar edilecek kodlar
}
```

Görüldüğü üzere syntax oldukça basit.

İlk olarak anahtar kelime “while” ile döngü olduğu belli ediyoruz.

Ardından “( )” ile koşulumuzu döngüye veriyoruz.

“{ }” ile döngümüzün gövdesini. Yani tekrar tekrar çalışacak kodlarımızı yazıyoruz.

Burada dikkat etmemiz gereken nokta şudur. İlk olarak şart kontrol edilecek. True ise döngüye girilecek ve gövde çalıştırılacak. False ise döngüye hiç girilmeyecek. Kod akışında değişir ise ve false olursa bir sonraki kontrolde döngüden çıkılacak.

Hadi inceleyelim.

```
static void Main(string[] args)
{
    int sayac = 0;
    while (sayac < 5)
    {
        Console.WriteLine(sayac);
    }
    Console.Read();
}
```

Şekil 2 While Döngüsü

Eğer şekil 2'deki programı çalıştırırsanız. Programın sonsuza kadar çalıştığını görebilirsiniz. Bunun sebebi şartın her zaman doğru olmasıdır. 0 her zaman 5'ten küçüktür. Programlama dünyasında

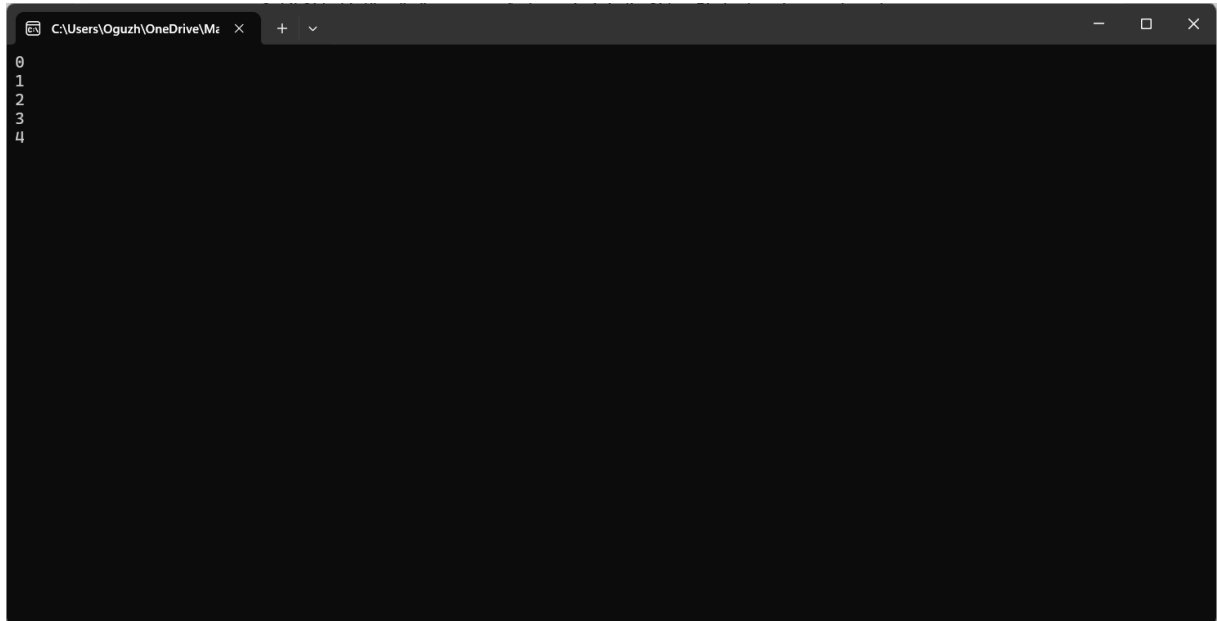
sonsuz döngüler ciddi bir sorundur. Sonsuz döngülere benzer durumlar kullanılır. Ancak istenildiği zaman durdurulabilme şartı ile. Daha basit araçların; Örneğin Arduino gibi araçların programlamalarında sonsuz döngülerin kullanıldığını görebilirsiniz. Aslında yazdığımız her program bir çeşit sonsuz döngü içerisinde çalıştırılır. Tek fark istenildiği zaman durdurulabilir olmasıdır.

Şekil 2’de ki döngünüz amacını fark etmişsinizdir. 0’dan 5’e kadar olan sayıları ekrana yazdırmak. Bu durumda sayıyı her zaman döngü içerisinde arttırmalıyız. Peki nasıl arttıracacağız? Operatörler dersini hatırlamaya çalışın. O dokümanda “++” operatöründen bahsetmiştik. Bu operatör sayıların değerini bir artırır. Hadi bu operatörü deneyelim.

```
static void Main(string[] args)
{
    int sayac = 0;
    while (sayac < 5)
    {
        Console.WriteLine(sayac);
        sayac++;
    }
    Console.Read();
}
```

Şekil 3 “++” operatörü kullanımı

Şekil 3’deki programın çıktısı şu şekildedir;

A screenshot of a Windows console window. The title bar shows the file path 'C:\Users\Oguzh\OneDrive\Mz' and standard window controls. The console output displays the numbers 0, 1, 2, 3, and 4, each on a new line. The cursor is positioned at the end of the line containing the number 4.

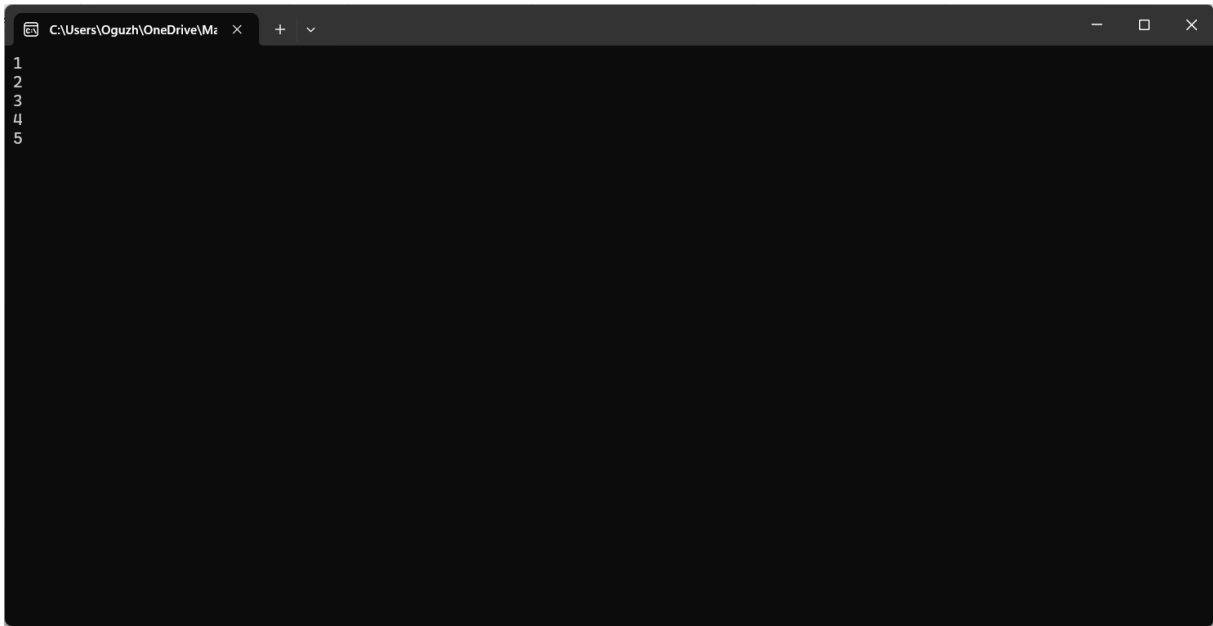
Şekil 4 “++” operatörü kullanımı programı çıktısı.

Gördüğünüz üzere programımız “0, 1, 2, 3, 4” şeklinde bir çıktı verdi. Ardından programımız durdu. Sayı 5’e geldiğinde artık şart sağlanmadığı ve “<” operatörü “false” değer döndürdüğü için döngüden çıkıldı. Peki bundan nasıl emin olabiliyoruz? Döngüden çıkıldığından ya da döngünün içine girildiğinden. Tabi ki yazdığımız kodların çıktılarına bakarak! Doğru ancak her zaman bu şekilde program yazmayacağız. Her zaman anlaşılabilir olmayacak. Bundan dolayı debug yapmayı öğrenmek zorundayız. Hem debug’dan önce bir konuyu tekrar hatırlayalım. Debug için’de önemli bir konu. Kodun senkronizasyonu. Ya da sürekli olarak yukarıdan aşağıya doğru çalışması. Hadi “++” ile konsola yazı yazdırdığımız kodların yerini değiştirelim ve çıktıya bakalım.

```
static void Main(string[] args)
{
    int sayac = 0;
    while (sayac < 5)
    {
        sayac++;
        Console.WriteLine(sayac);
    }
    Console.Read();
}
```

Şekil 5 Kod sırasını değiştirme.

Bu sefer ne olacak peki? Aynı çıktıyı mı alacağız? Bakalım:



Şekil 6 Kod sırası değiştirme program çıktısı

Bu sefer 0'dan 5'e değil de 1'den 5'e kadar 5'te dahil olmak üzere olan sayıları yazdırdık.

Bunun nasıl olduğunu anlamışsınızdır. Henüz basit aşamadaysınız. Ancak yine de programımızı adım adım izlemek, nasıl çalıştığını anlamamızı oldukça kolaylaştıracaktır. Hadi debug yapmayı öğrenelim.

## DEBUG

- **"De"**: Bu ön ek, "çıkarmak" veya "temizlemek" anlamına gelir.
- **"Bug"**: Yazılımda veya donanımda karşılaşılan bir hata veya sorun anlamına gelir.

Birleştirildiğinde **"Debug"**, "hataları temizlemek" veya "hatalardan arındırmak" anlamına gelir.

Debug, yazılım geliştirme sürecinde şu işlemleri içerir:

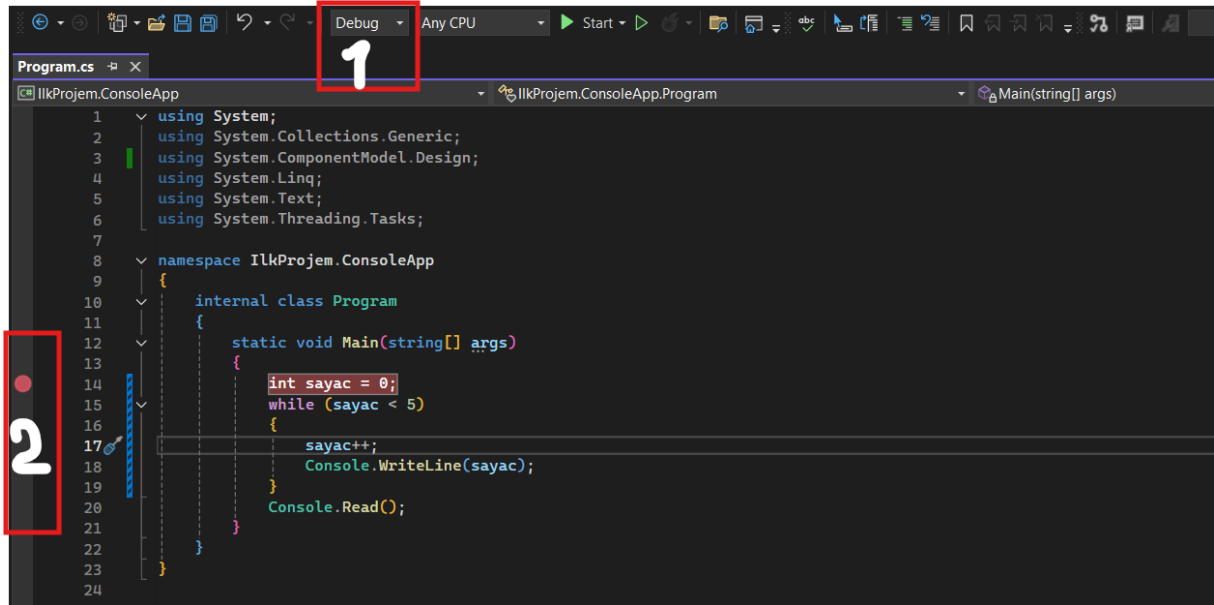
1. **Hataları Tespit Etmek**: Hangi bölümde sorun olduğunu anlamak.

2. **Hatanın Sebebinin Bulmak:** Hatanın neden oluştuğunu analiz etmek.
3. **Hatanın Çözülmesi:** Kod üzerinde değişiklikler yaparak hatayı gidermek.
4. **Test Etmek:** Hatanın gerçekten çözümlü çözülmeyeceğini kontrol etmek.

Debugging genellikle geliştirme araçları (IDE'ler) içindeki **debugger** adı verilen araçlarla yapılır. Bu araçlar, kodu adım adım çalıştırarak değişkenlerin değerlerini ve kodun akışını gözlemlemenize olanak tanır.

Biz de IDE olarak visual studio kullanıyoruz. Visual studio'da debugging oldukça basittir. İlk olarak programımızın debug modda çalıştırıldığına emin olmalıyız. İkinci olarak ise hangi noktada adım adım çalışması istediğimizi işaretlemeliyiz. Tüm programı baştan aşağıya adım adım çalıştırmak oldukça uzun sürecektir. Bundan dolayı şüphelendiğimiz alanı adım adım çalıştırmak bizim için en kısa yol olacaktır.

Hadi debug süreci için hazırlıklarımızı yapalım.

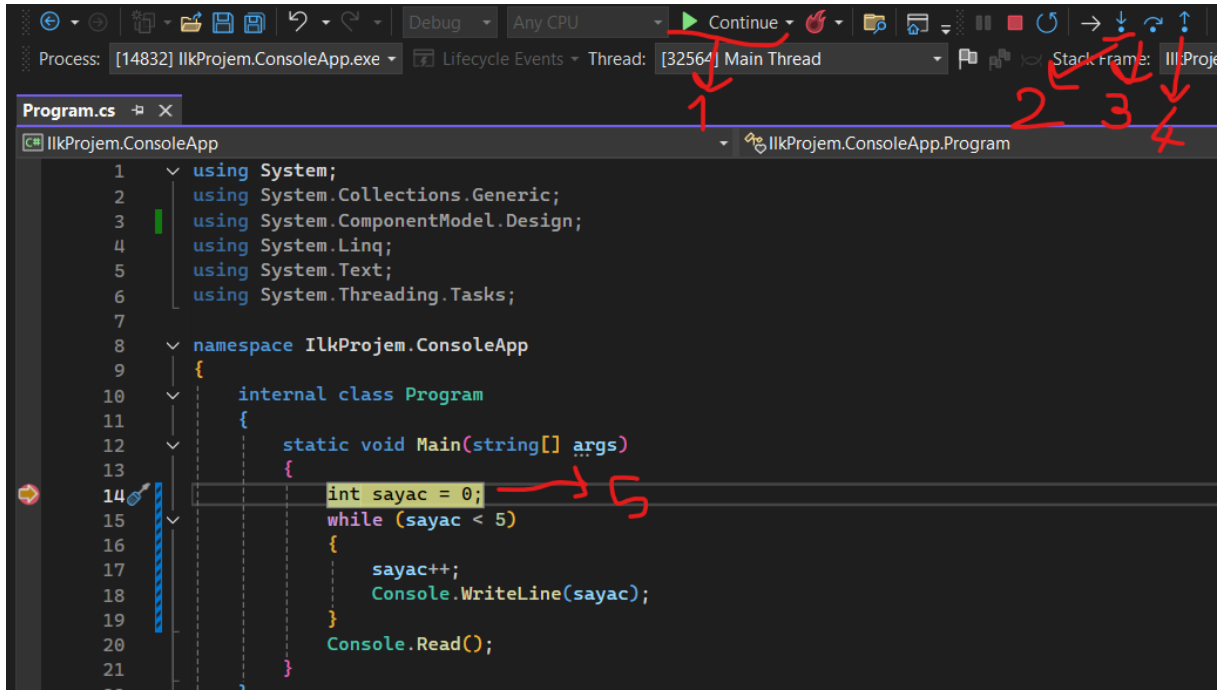


Şekil 7 Debug için hazırlık.

1-) Programımızın debug modda çalıştırılacağını söylüyoruz. Bu sayede eğer programımızın herhangi bir yerinde hata alırsak IDE bize hatayı gösterecek ve hatanın nerede olduğunu anlamış olacağız. Ancak bu hatalar programı çalışmaz ya da çökerten hatalardır. Visual studio'da bu mod varsayılan olarak debug olarak ayarlıdır.

2-) Satır sayısının sol tarafında ki çubuğa Mouse sol tuşu ile tıklayarak bir "breakpoint" ya da kırılmak noktası oluşturuyoruz. Programımız çalıştığında ve kod akışı buraya geldiğinde program burada duracaktır. Burada da eğer program beklediğimiz gibi çalışmıyorsa bu sayede adım adım izleyebiliriz.

Hadi programımızı durdurup bakalım:



Şekil 8 Debug süreci

- 1-) "Continue" tuşuna tıklayarak kırılma sürecini bitirip programın akışını normal devam etmesini sağlayabilirsiniz.
- 2-) "Step into" tuşuna tıklayarak programınızı adım adım izleyebilirsiniz. Ancak bu tuş çağırılan metotlarında içerisine girecektir. Bundan dolayı bu seçeneği bu aşama o kadar sık kullanmayacaksınız. Henüz metotları öğrenmediniz.
- 3-) "Step over" Tuşuna tıklayarak yine aynı şekilde programınızı adım adım ilerletebilirsiniz. Bu sefer çağırılan metotların üzerinden atlanır ve içine girilmez. Henüz metotları öğrenmediğiniz için sıklıkla kullanacağınız yöntem budur. Öğrenseniz de yine sıklıkla bu metodu kullanacaksınız.
- 4-) "Step out" tuşuna tıklayarak içine girdiğiniz metodu bitirip metottan çıkabilirsiniz. Yine şu an sizin için gereksiz.
- 5-) "Breakpoint" programınız tam bu aşamada kesilmiş durumda. Satırımızın renginin değiştiğini görebilirsiniz. Yukarıda bahsettiğim tuşlara tıklayarak sol tarafta görünen oku hareket ettirip programınızı ilerletebilirsiniz. Ya da sol taraftaki oka basılı tutarak yukarı ya da aşağı doğru hareket ettirebilirsiniz. Bu sayede programınızı geri sarabilirsiniz. "sayac" değişkeninin üzerine Mouse ile gelerek o anki değerini görebilir, isterseniz o an yeni değer atayabilirsiniz.

Bu araçları kullanarak döngünün içerisini inceleyebilirsiniz. Her seferinde "sayac" değişkeninin değerini gözlemleyebilirsiniz.

## SAYAÇLAR VE DİZİLER İLE DÖNGÜ

Döngülerin kullanım alanlarının çok büyük çoğunluğunu diziler kapsamaktadır. Bir sayaç ile dizinin tüm elemanları gezilir. İstenilen işlem yapılır ve döngü sonlandırılır. Şimdi var sayalım ki elimizde bir tam sayı dizisi olsun. Bu dizinin elemanları öğrencilerin sınav puanlarını temsil etsin. İlk olarak bu puanları ekrana sıra ile yazdıralım.

“sayac++” kod bloğunun konuma bağlı olarak program akışının nasıl değiştiğini öğrendiğinizi var sayıyorum.

```
static void Main(string[] args)
{
    int sayac = 0;
    int[] ogrenciNotlari = { 54, 65, 34, 82, 73, 45 };
    while (sayac < 5)
    {
        sayac++;
        Console.WriteLine(ogrenciNotlari[sayac]);
    }
    Console.Read();
}
```

Şekil 9 Öğrenci notları sıralama programı

Bu programın çıktısı şu şekildedir.

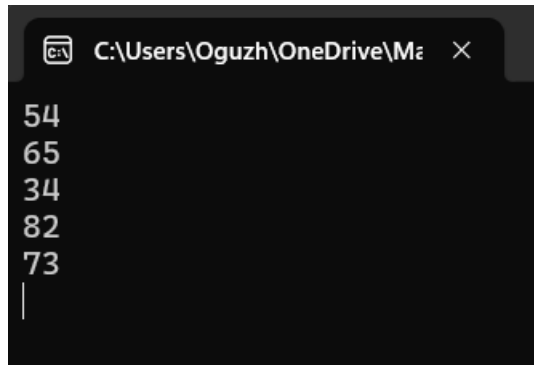


```
C:\Users\Oguzh\OneDrive\Ma... X + v
65
34
82
73
45
```

Şekil 10 Öğrenci notları sıralama programı çıktısı

Gördüğünüz üzere ilk not olan 54 ekrana yazdırılmadı. Burada bir bug bulunmaktadır. Bu bug’ı çözebilmek için debug ile çalıştırıp bulmanız gerekmektedir. Bu size ödevdir.

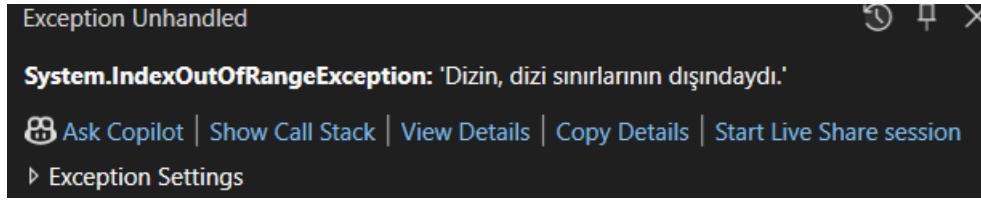
Bu sorunu çözdükten sonra program çıktınız şu şekilde olacaktır.



```
C:\Users\Oguzh\OneDrive\Ma... X +
54
65
34
82
73
|
```

Şekil 11 Öğrenci notları sıralama programı çıktısı 2. Adım

Bu seferde sondaki 45 notu sıralanmamaktadır. Yine debug ile bu sorunu bulmanız gerekmektedir. Bu sorunu çözerken en çok karşılaşacağınız hata şudur.



Şekil 12 Index out of range exception

Bu hata sayacın dizi sınırlarının dışına çıktığını söylemektedir. Peki bu durumu nasıl çözeceğiz? Şu an ki diziye biz oluşturduğumuz için uzunluğunu biliyoruz. Peki ya uzunluğunu bilmediğimiz bir dizi gelirse ne yapacağız? .Net framework sağolsun bize dizinin uzunluğunu söyleyen bir özelliği dizilere eklemiş durumda. Hadi onu kullanalım.

```
int sayac = 0;
int[] ogrenciNotlari = { 54, 65, 34, 82, 73, 45 };
while (sayac < ogrenciNotlari.Length)
{
}
```

Şekil 13 Array'lerde length (uzunluk) özelliği

Bu sayede array'in uzunluğu ne olursa olsun artık o sayı elinizde. Ancak bir sorun var. Eğer sayac'ınızın konumu doğru değilse yine şekil 12'deki hatayı alacaksınız. Bu sorunları teker teker debug ile çözmeni gerekmektedir.

Bir dizinin index'i ile uzunluğu (length) arasında bir fark vardır. "ogrenciNotlari" adlı dizisinin 6 elemanlı olduğunu göreceksiniz. Bu durumda uzunluk 6 olacak. Ancak son elemanın index'inin 5 olduğunu önceki derslerden öğrendik. Uzunluk değerini kullanarak son elemanı yazdırmak istersek şuna dikkat etmemiz gerekecek.

```
int sayac = 0;
int[] ogrenciNotlari = { 54, 65, 34, 82, 73, 45 };
Console.WriteLine(ogrenciNotlari[ogrenciNotlari.Length]); // hatalı. şekil 12'deki hatayı alacaksınız
Console.WriteLine(ogrenciNotlari[ogrenciNotlari.Length - 1]); // doğru. 6 elemanlı bir dizinin son elemanı 5. indistir.
```

Şekil 14 Dizi uzunluğu ile dizinin son elemanını yazdırmak.

Yazılımın başlarında bu alanda oldukça hata yapacaksınız. Bu gibi hataları gidermek için ise bol bol debug kullanacaksınız. Bunları anladığınızı düşünerek diğer iki döngümüzü de inceleyelim.

## FOR DÖNGÜSÜ

**For döngüsü**, belirli bir başlangıç, bitiş ve artış/değişim kriterine göre çalışan bir döngüdür. Genellikle bir dizi, sayı aralığı veya koleksiyon üzerinde işlem yapmak için kullanılır.

### Kullanım Amacı:

- Döngü sayısı önceden bilindiğinde.
- Bir sayılar dizisinde veya dizide indekslere erişmek gerektiğinde.
- Döngü içerisinde koşula bağlı kontrol yapılarında.

### Syntax:

```
for (başlangıç; koşul; artış/azalış)
{
    // Döngüde çalışacak kodlar
}
```

### Her Bileşenin Anlamı:

1. **Başlangıç:** Döngünün başlangıç noktasını tanımlar. Genellikle bir sayaç değişkeni burada tanımlanır. Örneğin: `int i = 0`. Değişkeni isterseniz dışarıda da tanımlayabilirsiniz. Ancak genel olarak for döngüsü içerisinde tanımlanır.
2. **Koşul:** Döngünün devam etmesi için sağlanması gereken şarttır. Şart false olduğunda döngü sona erer. Örneğin: `i < 10`. While döngüsünde de kullandığımız gibi. Bu alana doğrudan false ya da true değeri verebilirsiniz. Ancak bu ya sonsuz döngüye ya da hiç çalışmayan bir kod bloğuna sebep olur.
3. **Artış/Azalış:** Döngü her çalıştırıldığında sayaç değişkeninin nasıl değişeceğini tanımlar. Örneğin: `i++` (1 artır) veya `i--` (1 azalt).

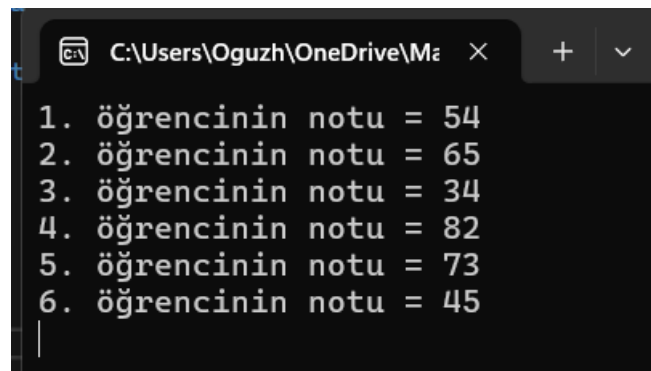
**Örnek:** Bir dizideki tüm elemanları yazdıran for döngüsü:

String interpolasyonunu da kullanalım ve indisimizi de yazdıralım.

```
static void Main(string[] args)
{
    int[] ogrenciNotlari = { 54, 65, 34, 82, 73, 45 };
    for (int i = 0; i < ogrenciNotlari.Length; i++)
    {
        Console.WriteLine($"{i+1}. öğrencinin notu = {ogrenciNotlari[i]}");
    }
    Console.Read();
}
```

*Dikkat*

Şekil 15 For döngüsü ile öğrenci notlarını yazdırmak.



```
C:\Users\Oguzh\OneDrive\M...
1. öğrencinin notu = 54
2. öğrencinin notu = 65
3. öğrencinin notu = 34
4. öğrencinin notu = 82
5. öğrencinin notu = 73
6. öğrencinin notu = 45
```

Şekil 16 For döngüsü ile öğrenci notlarını yazdırma programı çıktısı



### Çalışma Mantığı:

- `int i = 0`: Döngü, `i` değişkeni 0 olarak başlar.
- `i < numbers.Length`: Döngü, `i` değişkeni dizinin uzunluğundan küçük olduğu sürece çalışır.
- `i++`: Her iterasyonda `i` bir artırılır.

Sayaç ifadeleri genel olarak `i` ile başlar. (iterator kelimesinin ilk harfi) birden fazla var ise `j k` ile devam eder.

For döngüsü oldukça basittir. Ancak yine while döngüsünde olduğu gibi indis hatası ile karşılaşabilirsiniz. Bu durumda bu hatayı da doğrudan engelleyebileceğimiz ve rahatlıkla kullanabileceğimiz bir döngü daha bulunmaktadır.

## FOREACH DÖNGÜSÜ

**Foreach döngüsü**, koleksiyonlardaki (diziler, listeler vb.) her bir öğe üzerinde teker teker işlem yapmak için kullanılır. Bu döngü, for döngüsünden farklı olarak, indekslerle uğraşmayı gerektirmez ve her öğeyi otomatik olarak işler.

### Kullanım Amacı:

- Koleksiyonun her elemanını kolayca işlemek.
- Koleksiyonun elemanlarını sırasıyla okumak (değişiklik yapılmaz).
- Daha okunabilir ve güvenli bir yapı sunmak.

### Syntax:

```
foreach (var eleman in koleksiyon)
{
    // Döngüde çalışacak kodlar
}
```

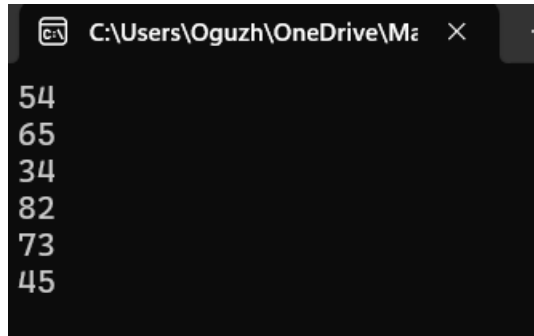
### Her Bileşenin Anlamı:

1. **Var/Tip Belirleme:** Koleksiyondaki elemanın tipini belirtir. Örneğin, bir dizi `int` ise `int` kullanılır. Alternatif olarak, `var` ile tip belirlenir. “`var`” anahtar kelimesini henüz öğrenmediniz. Ancak bu döngüde bu anahtar kelimeyi kullanabilirsiniz. Bu anahtar kelime sizin bilmediğiniz veri tipini kullanmanızı kolaylaştırır. Veri tipi otomatik algılanır. Ve o tipte gibi işlem yapılır. Bu daha ileride detaylı olarak öğreneceğiz.
2. **Eleman:** Döngü sırasında ele alınan mevcut öğeyi ifade eder.
3. **Koleksiyon:** Üzerinde işlem yapılacak koleksiyon.

**Örnek:** Bir dizideki tüm elemanları yazdıran foreach döngüsü:

```
static void Main(string[] args)
{
    int[] ogrenciNotlari = { 54, 65, 34, 82, 73, 45 };
    foreach (int item in ogrenciNotlari)
    {
        Console.WriteLine(item);
    }
    Console.Read();
}
```

Şekil 17 foreach döngüsü ile öğrenci notlarını yazdırma.



54  
65  
34  
82  
73  
45

Şekil 18 foreach döngüsü ile öğrenci notları yazdırma program çıktısı

Döngüde int anahtar kelimesi yerine var anahtar kelimesini de kullanabilirsiniz. Ancak dikkat ettiğiniz üzere indis değerini kullanmak bu döngüde biraz daha zordur. Buna daha sonra değineceğiz.

#### Çalışma Mantığı:

- int item: Her iterasyonda ogrenciNotlari dizisinden sıradaki elemanı alır.
- ogrenciNotlari: İşlem yapılacak koleksiyon.

Şimdi bu iki döngü arasında ki farklara geniş açıdan bakalım.

## FOR VE FOREACH KARŞILAŞTIRMASI

Tablo 2 For - Foreach döngüsü karşılaştırması

Özellik	For Döngüsü	Foreach Döngüsü
Kullanım Alanı	İndekslerle işlem yapmak gerektiğinde.	Koleksiyonun elemanlarını teker teker işlerken.
Performans	Daha hızlıdır; indeks tabanlıdır.	Daha okunaklıdır ancak indeks yoktur.
Okunabilirlik	Daha az okunabilir; indeks ve sayaç gerektirir.	Daha okunabilir; indeks gerekmez.

Özellik	For Döngüsü	Foreach Döngüsü
Değişiklik Yapma	Koleksiyon üzerinde değişiklik yapılabilir.	Koleksiyonun elemanlarını değiştiremez.

## FOR VE FOREACH KULLANIMINDA ÖNEMLİ NOKTALAR

### 1. For Döngüsünde Dikkat:

- Koşul ifadesi dikkatlice belirlenmelidir, aksi halde sonsuz döngüye neden olabilir.
- İndeks dışında değişkenleri güncellemek karmaşık hale gelebilir.

### 2. Foreach Döngüsünde Dikkat:

- Koleksiyonun elemanlarını değiştiremezsiniz; yalnızca okuyabilirsiniz. (gerekli kodu yazdıktan sonra değiştirebilirsiniz. Ancak doğrudan değiştiremezsiniz.)
- Koleksiyonun null olması durumunda çalışmaz, hata verir. (null ifadeyi henüz öğrenmediniz. Buna ileride değineceğiz.)

## ÖDEV

- “++” operatörünün kullanımını araştırınız.
- While döngüsünde size verilen ödevi çözünüz. (O bug’ın çözümünde “++” operatörün kullanımı ile alakası yoktur.)
- “—” operatörünü kullanarak verilen diziyi tersten yazdırınız.
- Öğrendiğiniz bütün döngüleri kullanarak listeyi ekrana yazdırınız.
- Do – While döngüsü bu dokümanda sizlere anlatılmamıştır. Bu döngü çeşidini araştırınız ve hakkında kısa bir doküman oluşturunuz.

## TEST

### 1. While döngüsü ile ilgili aşağıdakilerden hangisi doğrudur?

- A) While döngüsü, belirtilen koşul yanlış (false) olduğu sürece çalışır.
- B) While döngüsü, koşul her zaman doğru (true) ise sonsuz döngüye neden olabilir.
- C) While döngüsü, koşul kontrolü yapılmadan döngü gövdesini çalıştırır.
- D) While döngüsü, yalnızca integer türündeki değişkenlerle kullanılabilir.

### 2. C# dilinde bool veri tipi genellikle hangi tür işlemlerde kullanılır?

- A) Sadece matematiksel hesaplamalarda
- B) Karar mekanizmalarında ve mantıksal ifadelerde
- C) String ifadelerle karşılaştırmalarda
- D) Sadece döngülerin kontrol mekanizmasında

### 3. Debug işlemi sırasında kullanılan "Step Over" seçeneği aşağıdakilerden hangisini yapar?

- A) Programı adım adım çalıştırır ve metodların içine girer.
- B) Programı adım adım çalıştırır, ancak metodların içine girmez.

- C) Tüm kodu baştan sona çalıştırır.
- D) Hataları otomatik olarak düzeltir.

**4. Aşağıdaki mantıksal operatörlerden hangisi iki önermenin her ikisi de doğru olduğunda true döner?**

- A) ||
- B) &&
- C) !=
- D) !

**5. Foreach döngüsü ile ilgili aşağıdakilerden hangisi yanlıştır?**

- A) Koleksiyon elemanlarını sırasıyla işler.
- B) Koleksiyonun elemanlarını değiştiremez.
- C) İndekslerle işlem yapmak için kullanılır.
- D) Daha okunaklı ve güvenli bir yapı sunar.

**Cevap: C**

**6. Bir dizinin uzunluğunu öğrenmek için hangi özellik kullanılır?**

- A) size()
- B) length()
- C) Length
- D) count()

**7. Aşağıdaki durumlardan hangisi sonsuz döngüye neden olabilir?**

- A) Döngü koşulunun her zaman true olması
- B) Döngü gövdesinde koşulun değişmemesi
- C) Döngü başında yanlış bir koşul belirtilmesi
- D) Döngünün hiç çalışmaması

**8. For döngüsünde aşağıdaki ifadelerden hangisi döngünün sonlanmasını kontrol eder?**

- A) Başlangıç ifadesi
- B) Artış/azalış ifadesi
- C) Koşul ifadesi
- D) Döngü gövdesi

**9. Foreach döngüsünde elemanların tipini otomatik belirlemek için hangi anahtar kelime kullanılır?**

- A) dynamic
- B) auto
- C) var
- D) type

**10. Döngülerin çoğunlukla hangi yapı ile kullanıldığı belirtilmiştir?**

- A) Mantıksal operatörler
- B) Diziler
- C) Metotlar
- D) Breakpointler

## CEVAP ANAHTARI

1 – B

2 – B

3 – B

4 – B

5 – C

6 – C

7 – A

8 – C

9 – C

10 - B