2019 – 2020

Nesne Tabanlı Programlama I

Ders Notları

Öğr. Gör. Hakan Can ALTUNAY
ÇARŞAMBA TİCARET BORSASI MESLEK YÜKSEKOKULU

Nesne Tabanlı Programlama I – 29.09.2017

- Hocamız sıfırdan C# dilini önce teorik olarak, sonra da konsol uygulamaları ile öğretecek.
- Nesne Tabanlı Programlama 2'de Python ile devam edeceğiz.
- C#'ın avantajlarından bahsettik.
- Fonksiyonun ne olduğundan ve nasıl çalıştığından kısaca bahsettik.
- C#'daki blok yapısından kısaca bahsettik.
- Bool en fazla while döngüsünde kullanıldığından bahsettik.
- Aynı kod bloğu içerisinde aynı değişken birden fazla kez tanımlanamaz.

Değişken nedir?

Değişkenler, programın her çalıştırılmasında farklı değerler alabilmesini sağlayan yapılara denir. Bir değişkeni tanımlayabilmek için;

1) Rakam kullanılabilir fakat ilk karakter rakam ile başlayamaz.
2) Programlama dillerindeki for, if, while, foreach gibi herhangi o programlama diline ait özel terimler kullanılamaz.
3) Boşluk karakteri kullanılamaz.
4) Alt tire hariç hiçbir özel karakter kullanılamaz.
5) İngiliz alfabesindeki 26 harf kullanılabileceği gibi C#'ın bazı sürümlerinde Türkçe karakterler kullanılabilir.
6) Aynı isimli değişkenler iki kez tanımlanamaz.

Veri Türleri

Sayısal	Alfa Sayısal	Mantıksal	Nesne	Tarih ve Zaman
int byte float double	char string	bool	Object	DateTime

- Veri türlerini niteleyen sıfatlar;
  - ✓ long → uzun
  - ✓ short → kısa
  - ✓ signed → işaretli
  - ✓ unsigned → işaretsiz

Değişkenlerin Tanımlanması

Değişken Tipi	Değişken Adı	Başlangıç Değeri
`int`	a	
`int`	a	5
`double`	b	6.4
`char`	c	`'1'`
`string`	bolum	`"bilgisayar"`

Kaçış Dizileri

C# dilinde komut yazarken sıklıkla kullanılan ve genel olarak adına kaçış dizisi adı verilen karakter dizileri kullanılır.

\n	Alt satıra geçer.
\t	TAB tuşu özelliğini yerine getirir.
\0	Null (Boş)
\"	Çift tırnak işaretinin ekrana basar.
\r	Satırbaşı yapmak için kullanılır.
\b	Geri alma işlemini yapar. (Backspace)
****	\ işaretinin ekrana basılmasını sağlar.
\a	**1)** "Bip" veya sistemdeki hata. **2)** Sistemden uyarı sesinin çıkartılmasını sağlar.

Değişkenlerin Yaşam Alanı

Bir programda pek çok değişken kullanılabilir. Bu değişkenlerin geçerlilik alanları kullanıldıkları blok ile sınırlıdır. C# dilinde bloklar süslü parantez **"**{ }**"** işaretleri ile gösterilir.

```
class Bloklar
{
    static void Main()
    {
        int a = 3;
        int a = 5;
        Console.ReadKey();
    }
}
```

Yandaki kod bloğunda aynı blok içerisinde a değişkeni iki kez tanımlandığından program derlemede hata verecektir.

Derleyicide hata vermemesi için Main() ana kod bloğunun içerisine iki farklı kod bloğu eklenerek değişkenlerin tanımlandığı alanlar birbirinden ayrılır ve hatanın önüne geçilir.

```
class Bloklar
{
    static void Main()
    {
        {
            int a = 3;
        }
        {
            int a = 5;
        }
        Console.ReadKey();
    }
}
```

Sabitler

Programın içerisinde bir kere tanımlandıktan sonra değişmeyen değişkenlere sabit değişken denir. Bir değişkeni sabit yapmak için C# dilinde değişken türünün başına "const" ifadesi eklenir.

`const double pi = 3.14;` → "const" ifadesi değişken türünün başında olduğu sürece değiştirilemez.

Bir değişkenin sabitlenmesinden sonra kısaca tür dönüşümü işine baktık. Tür dönüşümü işini hocamız `Convert.To` şekliyle anlattı. Ek olarak Convert işlemlerini Parse ile yapmamızda herhangi bir sakınca olmadığını söyledi.

Nesne Tabanlı Programlama I – 06.10.2017

C# Dilinde Veri Giriş İşlemleri

C# dilinde veri girişi için Read() ve ReadLine() komutları kullanılır. Read() komutu tek karakterlik bilgi okurken ReadLine() metodu ise satır satır okuma işlemi yapılır. Read() ile okunan bilgi int olarak, ReadLine() ile okunan bilgi ise string olarak okunur. *(Read() → int , ReadLine() → string)*

Soru: Klavyeden vize ve final bilgilerinin girilerek öğrencinin ortalama notunu hesaplayınız.

```
// Mantık hatasından dolayı bu koddaki ortalama daima tamsayı çıkacaktır.
int vize, final;
double ortalama;
vize = Convert.ToInt32(Console.ReadLine());
final = Convert.ToInt32(Console.ReadLine());
ortalama = (vize + final) / 2;
Console.Write("Ortalama: " + ortalama);
Console.ReadKey();
```

C# Dilinde Veri Çıkış İşlemleri

C# dilinde ekrana bilgi yazdırmak için Console sınıfına ait Write() ve WriteLine() metotları kullanılır. Bu metotların her ikisi de ekrana yazdırılacak bilgiyi string tipte yazdırır. Bu metotlar ile;

1. Tırnak işareti içerisindeki değeri ekrana doğrudan yazdırabiliriz.
   `Console.Write("ABC");`
2. Herhangi bir değişkendeki değeri ekrana yazdırabiliriz.
   `Console.Write(a);`
3. Blokları *({ })* kullanarak değişkenlerden alınacak değerleri sırayla aktarmak şartıyla formatlı şekilde ekrana bilgi yazdırırız.
   `Console.Write("({0} + {1}) / 2", vize, final);`

C# Dilinde Kullanılan Operatörler

1) Matematiksel Operatörler	
İşlem	**Anlamı**
+	Toplama
-	Çıkarma
*	Çarpma
/	Bölme
%	Mod alma
++	Bir arttır
--	Bir azalt

Bölme işleminde sonucun ondalık çıkabileceği durumlarda yanlışa düşmemek için sonuç değişkeni ondalık olarak tanımlanır ve bölme işlemine girecek olan sayılardan herhangi biri veya her ikisi ondalık olarak yazılmalıdır.

```
int a = 8 / 3;
Console.Write(a);
//Sonuç: 2
```

```
double a = 8 / 3;
Console.Write(a);
//Sonuç: 2
```

```
double a = 8.0 / 3;
Console.Write(a);
//Sonuç: 2.66666667
```

Arttırma ve eksiltme operatörlerinde aşağıdaki örnek dikkatle incelendiğinde;

```
{
   int x, y;
   x = 5;
   y = 3;
   x = y++;
   Console.ReadKey();
}
```

x = y++ satırında öncelikle y'nin değeri x'e aktarılır, sonrasında da y'nin değeri 1 arttırılır. Sonuç olarak da x = 3 ve y = 4 olur. Komut satırı x = ++y şeklinde olsaydı öncelikle y'nin değeri 1 arttılır sonra da y'nin değeri x'e aktarılırdı. Sonuç olarak da y = 4 ve x = 4 olurdu.

Soru: Aşağıdaki programın ekran çıktısını yazınız. *(Nesne 1 Çıkmış Vize Sorusu - 2016)*

```
int a;
a = 3;
Console.WriteLine(a++ * ++a); // 3 * 5 = 15
a = 3;
Console.WriteLine(++a * ++a); // 4 * 5 = 20
a = 3;
Console.WriteLine(a++ * a++); // 3 * 4 = 12
a = 3;
Console.WriteLine(++a * a++); // 4 * 4 = 16
Console.ReadKey();
```

Soru: Aşağıdaki programın ekran çıktısını yazınız. *(Çıkmış Sınav Sorusu)*

```
int a, b;
a = 3;
b = 2;
b--;
a = b++;
Console.WriteLine(a); // a = 1
Console.Write(b); // b = 2
Console.ReadKey();
```

2) Karşılaştırma Operatörleri	
<	Küçüktür
>	Büyüktür
>= veya =>	Büyük eşit
<= veya =<	Küçük eşit
==	Eşit ise
!=	Eşit değilse

3) Mantıksal Operatörler	
&&	VE
||	VEYA
!	DEĞİL
??	NULL
?:	Şart *(Koşul)*

Mantıksal operatörde bulunan ?? için aşağıdaki örneği inceleyin;

```
string mesaj = "Merhaba Dünya!";
string sonuc = mesaj ?? "Mesaj yok!";
Console.WriteLine(sonuc); // Merhaba Dünya!
Console.ReadKey();
```

```
string mesaj = null;
string sonuc = mesaj ?? "Mesaj yok!";
Console.WriteLine(sonuc); // Mesaj yok!
Console.ReadKey();
```

Yukarıdaki örnekte öncelikle mesaj değişkenine "Merhaba Dünya" tanımlanıyor. Sonrasında da sonuç değişkeninde "mesaj değişkeni NULL değilse mesaj değişkenini göster, eğer ki mesaj değişkeni NULL ise "Mesaj yok!" göster." şeklinde işlem yapmaktadır.

Mantıksal operatörde bulunan ?: için aşağıdaki örneği inceleyin;

```
int x;
string durum;
x = 3;
durum = x > 5 ? "5'ten büyüktür." : "5'ten küçüktür.";
Console.Write(durum); // 5'ten küçüktür.
Console.ReadKey();
```

Not: Mantıksal operatörlerde işlem önceliği VE, VEYA, ?: şeklindedir.

4) Aktarma ve İşlemli Aktarma Operatörleri	
=	Eşit
+=	Önce topla, sonra aktar
-=	Önce çıkar, sonra aktar
*=	Önce çarp, sonra aktar
/=	Önce böl, sonra aktar
%=	Önce mod al, sonra aktar

5) Birleştirme Operatörleri	
+	Birleştir
+=	Önce birleştir, sonra aktar

İkinci birleştirme operatörünün kullanım şekli için aşağıdaki örneği inceleyebilirsin;

```
string yazi1 = "Merhaba";
string yazi2 = " Dünya!";
yazi1 += yazi2;
Console.WriteLine(yazi1); // Merhaba Dünya
Console.ReadKey();
```

Nesne Tabanlı Programlama I – 13.10.2017

C# Dilinde Kontrol (Şart) İfadeleri

Program içerisinde herhangi bir koşulun doğru olup olmadığına göre farklı işlemler yapılması gerekebilir. Bu durumda kontrol deyimleri kullanılır. C# dilinde if ve switch olmak üzere iki adet kontrol ifadesi vardır. if yapısında şartı, switch yapısında değişkeni kontrol ederiz.

1. If Yapısı ve Kullanım Türleri

1. Kullanım Türü	2. Kullanım Türü	3. Kullanım Türü

`if (şart)`
`{`
`  // şart doğru ise`
`  // yapılacak işlemler`
`}`
	`if (şart)`
	`{`
	`  // şart doğru ise`
	`  // yapılacak işlemler`
	`}`
	`else`
	`{`
	`  // şart yanlış ise`
	`  // yapılacak işlemler`
	`}`
		`if (şart1)`
		`{`
		`  // şart1 doğru ise`
		`  // yapılacak işlemler`
		`}`
		`else if (şart2)`
		`{`
		`  // şart1 yanlış ise`
		`  // şart2 kontrol edilir`
		`  // doğru ise yapılacak`
		`  // işlemler`
		`}`
		`else`
		`{`
		`  // şartların hiçbiri`
		`  // sağlanmıyorsa`
		`  // yapılacak işlemler`
		`}`

2. Switch Yapısının Kullanımı

```
switch(değişken)
{
   case 1: değer1; break;
   case 2: değer2; break;
   case 3: değer3; break;
   default: değerN; break;
}
```

If ve Switch Case ile İlgili Notlar

1) if yapılarında, şarttan sonra yapılacak işlem sayısı 1 ise, blok kullanmaya gerek yoktur.

2) if yapısında else if *(değilse eğer)* bloğu istenildiği kadar değil ihtiyaç duyulduğu kadar kullanılır.

3) if, else if, else yapısındaki else bloğu yukarıdaki şartların hiçbiri sağlanmıyorsa anlamına gelir.

4) switch case yapısında her case bloğunda mutlaka break; komutu kullanılmalıdır.

5) switch case yapısında default kullanmak zorunlu değildir.

6) switch case yapısında aynı değere sahip birden fazla case olamayacağı gibi kullanılan bu değerler sabit olmalıdır.

7) Herhangi bir case bloğundan başka bir case bloğuna “goto” anahtar sözcüğü ile gidilir. Bu durumda break komutu kullanılmaz.

Örnek 1: Klavyeden girilen iki sayıdan hangisinin büyük sayı olduğunu ekrana yazdıran programı C# Konsol kullanarak hazırlayınız. *(Eşit olma şartını şimdilik sorgulamıyoruz.)*

```
int a, b;
a = int.Parse(Console.ReadLine());
b = int.Parse(Console.ReadLine());
if (a > b)
   Console.Write("A sayısı daha büyüktür!");
else
   Console.Write("B sayısı daha büyüktür!");

Console.ReadKey();
```

Örnek 2: Klavyeden girilen not bilgisi 50'den az ise bütünleme notunu isteyen, bütünleme notu 50'nin altında ise "KALDI", değilse "GEÇTİ" mesajını ekrana yazdıran programı hazırlayınız. *(Vize Çıkmış Soru)*

```
int not, butNot;
not = int.Parse(Console.ReadLine());
if (not < 50)
{
   Console.Write("Bütünleme notunu giriniz!");
   butNot = int.Parse(Console.ReadLine());

   if (butNot < 50)
      Console.Write("KALDI");
   else
      Console.Write("GEÇTİ");
}
else
   Console.Write("GEÇTİ");

Console.ReadKey();
```

Örnek 3: Klavyeden girilen ay bilgisine göre hangi mevsimde olunduğunu yazdıran programı hazırlayın.

```
int ay = int.Parse(Console.ReadLine());
switch(ay)
{
   case 12:
   case 1:
   case 2: Console.Write("Kış"); break;
   case 3:
   case 4:
   case 5: Console.Write("İlkbahar"); break;
   case 6:
   case 7:
   case 8: Console.Write("Yaz"); break;
   case 9:
   case 10:
   case 11: Console.Write("Sonbahar"); break;
   default: Console.Write("Yanlış değer girdiniz!"); break;
}

Console.ReadKey();
```

C# Dilinde Döngü İfadeleri

C# dilinde, belirli sayıda ya da bir koşula bağlı olarak, bir takım ifadeleri tekrar tekrar çalıştırmak için dört farklı döngü ifadesi kullanılır.

1. For Döngüsü

Belirli sayıda tekrar eden işlemler için kullanılır. Genel kullanımı aşağıdaki gibidir:

1. Kullanım	2. Kullanım	3. Kullanım

`for (Döngü değişkeni = Başlangıç; Bitiş; Artış)`
`{`
`  // Yapılacak işlemler`
`}`
	`int a;`
	`for (a=1; a <= 10; a++)`
	`{`
	`  Console.Write(a);`
	`}`
		`for (int a=1; a<= 10 ; a++)`
		`{`
		`  Console.Write(a);`
		`}`

Örnek: 1 ile 10 arasındaki çift sayıları ekrana yazdıran for döngüsünü hazırlayınız.

```
for (int i = 2; i <= 10; i += 2)
{
   Console.WriteLine(i);
}
```

2. While Döngüsü

Şart sağlandığı sürece döngü içerisindeki işlemlerin tekrar edilmesini sağlar.

```
while (şart)
{
   // yapılacak işlemler
}
```

3. Do While Döngüsü

Tekrar etmesi düşünülen ifadeler şart sağlandığı sürece do while bloğunda çalıştırılır. Bu yapıda şart ne olursa olsun döngü içerisindeki işlemler en az bir kere çalıştırılır.

```
do
{
   // yapılacak işlemler
}
while (şart);
```

Örnek: Klavyeden sıfır sayısı girilene kadar girilen bütün sayıların toplamını ekrana yazdıran programı C# dilinde hazırlayınız. *(Vize Çıkmış Soru)*

```
int toplam = 0;
int giris;

do
{
   giris = int.Parse(Console.ReadLine());
   toplam += giris;
} while (giris != 0);

Console.WriteLine(toplam);
Console.ReadKey();
```

4. For Each

Her biri anlamına gelen for each döngüsü bir dizi, liste veya koleksiyondaki her bir değere ulaşılarak işlem yapılmasını sağlar. For each döngüsündeki veri tipi dizideki elemanların veri tipleri ile uyumlu olmalıdır. Genel kullanımı aşağıdaki gibidir:

```
foreach (int dizinin_elemani in dizi_degiskeni)
{
   // yapılacak işlemler
}
```

Örnek kullanım için aşağıdaki örneğe bakabilirsin;

```
int [] sayilar = { 3, 6, 7, 2, 1 };
foreach (int eleman in sayilar)
{
   Console.WriteLine(eleman);
}
```

5. Break ve Continue Komutları

Bazı durumlarda döngülerin sonlandırılması ya da sonraki değerden devam etmesi gerekebilir. Bu gibi durumlarda break ve continue komutları kullanılır.

Break komutu ile döngüyü istediğiniz yerde keserek döngünün gereksiz yere çalışmasının önüne geçer ve döngünün performansını arttırmış oluruz. Continue komutu ile de döngü yapısı içerisinde herhangi bir değerin döngüye dahil edilmemesi gerektiğinde bir sonraki değerden başlamak üzere kullanılır ve döngünün daha az çalışması sağlanır.

Break ve Continue komutları hem şart yapısı içerisinde hem döngü yapısı içerisinde hem de döngü ve şartın birlikte bulunduğu yapılarda kullanılır.

```
for (int i = 1; i <= 5; i++)
{
   if (i == 3)
   {
      break;
   }
   Console.Write(i + " "); // 12
}
Console.ReadKey();
```

```
for (int i = 1; i <= 5; i++)
{
   if (i == 3)
   {
      continue;
   }
   Console.Write(i + " "); // 1245
}
Console.ReadKey();
```

Ödev: Klavyeden girilen değere göre dört işlem yapan programı C# dilinde hazırlayınız. *(1-Toplama, 2-Çıkarma, 3-Çarpma, 4-Bölme, önce sayılar girilecek sonra yapılacak olan işlem seçilecek.)*

```
double sayi1, sayi2;
Console.Write("1. sayıyı giriniz: ");
sayi1 = double.Parse(Console.ReadLine());
Console.Write("2. sayıyı giriniz: ");
sayi2 = double.Parse(Console.ReadLine());
Console.Write("~ İşlem seçiniz ~\n1) Toplama\n2) Çıkarma\n3) Çarpma\n4) Bölme\nİşlem: ");
int islem = int.Parse(Console.ReadLine());
switch (islem)
{
   case 1: Console.Write("Sonuç: " + (sayi1 + sayi2)); break;
   case 2: Console.Write("Sonuç: " + (sayi1 - sayi2)); break;
   case 3: Console.Write("Sonuç: " + (sayi1 * sayi2)); break;
   case 4: Console.Write(String.Format("Sonuç: {0:0.##}", (sayi1 / sayi2))); break;
   default: Console.Write("Hatalı giriş yaptınız!"); break;
}
Console.ReadKey();
```

Nesne Tabanlı Programlama I – 20.10.2017

- Bugün 8 tane uygulamalı örnek yapacağız.
- **Soru 1:** Klavyeden taban uzunluğu ve yükseklik değeri girilen bir üçgenin alanını hesaplayan programı hazırlayınız.

```
double taban, yukseklik, alan;
Console.Write("Taban sayısını girin: ");
taban = Double.Parse(Console.ReadLine());
Console.Write("Yükseklik değerini girin: ");
yukseklik = Double.Parse(Console.ReadLine());
alan = (taban * yukseklik) / 2;
Console.Write("Sonuç: {0}", alan);
Console.ReadKey();
```

- **Soru 2:** Elektrik tüketim miktarı ve birim fiyat klavyeden girilmektedir. Aylık tüketim bedelini "aylık bedel = tüketim miktarı * birim fiyat" formülüne göre hesaplayarak ekrana yazdıracak programı C# dilinde hazırlayınız.

```
double birimFiyat, aylikBedel, tuketimMiktari;
Console.Write("Tüketim miktarını girin: ");
tuketimMiktari = double.Parse(Console.ReadLine());
Console.Write("Birim fiyatı giriniz: ");
birimFiyat = double.Parse(Console.ReadLine());
aylikBedel = (tuketimMiktari * birimFiyat);
Console.Write(aylikBedel);
Console.ReadKey();
```

- **Soru 3:** Klavyeden girilen iki tam sayıdan ilk girilen sayının, ikinci girilen sayıya tam olarak bölünüp bölünemediği ekrana yazdıran programı hazırlayınız.

```
Console.Write("1. sayıyı giriniz: ");
int sayi1 = int.Parse(Console.ReadLine());
Console.Write("2. sayıyı giriniz: ");
int sayi2 = int.Parse(Console.ReadLine());
if (sayi1 % sayi2 == 0)
    Console.Write("{0} sayısı {1} sayısına tam bölünür.", sayi1, sayi2);
else
    Console.Write("{0} sayısı {1} sayısına tam bölünmez.", sayi1, sayi2);
Console.ReadKey();
```

- **Soru 4:** Klavyeden girilen iki tam sayıda büyük olan sayıdan küçük olan sayıyı çıkartarak ekrana yazdıran programı hazırlayınız.

```
Console.Write("1. sayıyı giriniz: ");
int sayi1 = int.Parse(Console.ReadLine());
Console.Write("2. sayıyı giriniz: ");
int sayi2 = int.Parse(Console.ReadLine());
if (sayi1 > sayi2)
    Console.Write(sayi1 - sayi2);
else
Console.Write(sayi2 - sayi1);
Console.ReadKey();
```

- **Soru 5:** Satıldığı adet ve birim fiyatı klavyeden girilen bir ürünün; satış fiyatı 1000 TL'nin üstünde ise "Devam etmek istiyor musunuz?" şeklinde bir soru soran, bu soruya verilecek cevaba göre "İşlem tamamlandı." veya "İşlem iptal edildi." şeklinde uyarı mesajı veren, satış fiyatı 1000'in altında kalırsa "İşlem Onaylandı" yazan C# programını hazırlayınız.

```
int adet, toplamFiyat, birimFiyat;
char cevap;
Console.Write("Adet değerini giriniz: ");
adet = int.Parse(Console.ReadLine());
Console.Write("Birim fiyatı giriniz: ");
birimFiyat = int.Parse(Console.ReadLine());
toplamFiyat = adet * birimFiyat;
Console.Write("Satış fiyatı: " + toplamFiyat);
if (toplamFiyat > 1000)
{
    Console.Write("Devam etmek istiyor musunuz? (E / H)");
    cevap = char.Parse(Console.ReadLine());
       if (cevap == 'e' || cevap == 'E')
          Console.Write("İşlem onaylandı!");
       else
          Console.Write("İşlem iptal edildi!");
}
else
    Console.Write("İşlem onaylandı!");
```

- **Soru 6:** Klavyeden girilen herhangi bir tamsayının üç basamaklı olup olmadığını kontrol ederek;
    1. üç basamaklı değilse "Üç basamaklı sayı giriniz" şeklinde uyarı veren,
    2. girilen sayı üç basamaklı ise girilen sayının rakamlarının farklı olup olmadığını,

  ekrana yazdıran programı hazırlayınız. *(Çıkmış sınav sorusu)*

```
int x;
string metin;
do
{
   Console.Write("Üç basamaklı bir sayı giriniz:");
   x = int.Parse(Console.ReadLine());
   metin = x.ToString();
} while (metin.Length != 3 || x < 0);
if (metin[0] != metin[1] && metin[0] != metin[2] && metin[1] != metin[2])
   Console.Write("Rakamları birbirinden farklıdır.");
else
   Console.Write("Rakamları birbirinden farklı değildir.");
Console.ReadKey();
```

- **Soru 7:** Aşağıdaki ekran çıktısını veren programı hazırlayınız.

```
*
**
***
****
*****
```

```
for (int i = 1; i <= 5; i++)
{
   for (int j = 1; j <= i; j++)
      Console.Write("*");
   Console.WriteLine();
}
```

- **Soru 8:** Klavyeden girilen herhangi bir tamsayının asal sayı olup olmadığını kontrol eden C# programını hazırlayınız. *(Çıkmış sınav sorusu)*

```
int sayi; bool asal = true;
Console.Write("Bir sayı giriniz: ");
sayi = int.Parse(Console.ReadLine());
for (int i = 2; i < sayi; i++)
{
    if (sayi % i == 0)
       asal = false; break;
}
if (sayi < 2)
   Console.Write("Sayı asal değildir");
else if (asal)
   Console.Write("Sayı asaldır.");
else
   Console.Write("Sayı asal değildir.");
```

Nesne Tabanlı Programlama I – 27.10.2017

- **Soru 9:** Klavyeden girilen uzunluğu belirsiz herhangi bir tamsayının basamaklarında bulunan rakamları toplayarak ekrana yazdıran programı C# dilinde hazırlayınız.

```
int toplam = 0; // Toplam sıfırdan başlar.
string sayi;
Console.Write("Sayıyı giriniz: ");
sayi = Console.ReadLine(); // Girilen sayı string şeklinde.
for (int i = 0; i < sayi.Length; i++) // Lenght 1'den başlar.
    toplam += int.Parse(sayi[i].ToString());
Console.Write("Sayının rakamları toplamı: {0}", toplam);
```

- **Soru 10:** Klavyeden girilen herhangi bir metindeki sesli harf, sessiz harf, rakam ve özel karakterlerin sayısını ekrana yazdıran programı hazırlayınız. *(Çıkmış Final Sorusu)*

```
// string sesliharfler = "a,e,ı,i,o,ö,u,ü";
// string sessizharfler = "b,c,ç,d,f,g,h,j,k,l,m,n,p,r,s,ş,t,v,y,z";
string sesliharfler = "aeıioöuü";
string sessizharfler = "bcçdfghjklmnprsştvyz";
string rakamlar = "1234567890";
int seslisay = 0, sessizsay = 0, rakamsay = 0;
Console.Write("Metin giriniz: ");
string metin = Console.ReadLine();
metin.ToLower();
foreach (char karakter in metin)
{
    foreach (char sesli in sesliharfler)
    {
       if (karakter == sesli)
          seslisay++;
    }
    foreach (char sessiz in sessizharfler)
    {
       if (karakter == sessiz)
          sessizsay++;
    }
    foreach (char rakam in rakamlar)
    {
       if (karakter == rakam)
          rakamsay++;
    }
}
```

```
Console.WriteLine("Metin uzunluğu: " + metin.Length);
Console.WriteLine("Sesli harflerin uzunluğu: " + seslisay);
Console.WriteLine("Sessiz harflerin uzunluğu: " + sessizsay);
Console.WriteLine("Rakamların sayısı: " + rakamsay);
Console.WriteLine("Özel karakter sayısı: " + (metin.Length - (seslisay + sessizsay
+ rakamsay)));
Console.ReadKey();
```

C# Dilinde Diziler

Değişkenlerde aynı anda tek bir değer saklanabilir. Fakat bazı durumlarda birden fazla verinin saklanması gerekebilir. Bu durumda aynı veri türünde verilerin saklanabileceği dizi tanımlaması yapılır. Dizi tanımlaması yapılırken mutlaka dizinin kaç elemanlı olacağı belirtilir. Diziler hem eleman sayısı olarak hem de ilk değer ataması yapılarak tanımlanabilirler.

Dizinin Eleman Sayısı Verilerek Tanımlama	Diziye Eleman Atamasıyla Tanımlama
Dizinin_veri_tipi + [] + dizi_adı = new dizi_veri_tipi + [];	Dizi_tipi + [] + dizi_adi = {… , … , …}
`int [] sayilar = new int[5];`	`int [] sayilar = { 3, 4, 6, 7, 8 };`

Dizinin elemanlarına ulaşmak için index değerlerini kullanırız. İlk elemanın index'i her zaman sıfırdır.

Index	0	1	2	3	4
Değer	3	4	6	7	8

Dizilerin herhangi bir elemanına ulaşmak için;

```
Console.Write(sayilar[3]); // Sayılar dizisinin 4. elemanını ekrana yazdırır.
```

Dizilere değer atamak için köşeli parantezler içerisine dizinin kaçıncı elemanına değer atılacaksa index numarası olarak o yazılır.

```
sayilar[0] = 10; // Sayılar dizisinin 0. index numarasına 10 değerini atar.
```

Örnek: 5 elemanlı, elemanları da sırasıyla 3, 5, 10, 6, 8 olan sayılar dizisinin elemanlarına ulaşılmak istenmektedir. Gerekli olan yapıyı C# dilinde hazırlayınız.

```
int[] sayilar = { 3, 5, 10, 6, 8 };
Console.Write("Görmek istediğiniz eleman sırası: ");
int deger = int.Parse(Console.ReadLine());
if (deger > 0 && deger <= 5)
   Console.Write("Dizinin {0}. elemanı: {1}", deger, sayilar[deger - 1]);
else
   Console.Write("Yanlış değer girdiniz!");
Console.ReadKey();
```

C# Dizi Sınıfına Ait Metotlar

```
int [] sayilar = { 5, 10, 15, 2, 3 };
Console.WriteLine("Eleman sayısı: {0}", sayilar.Length); // Dizinin uzunluğunu verir
Console.WriteLine("Boyutu: {0}", sayilar.Rank); // Dizinin boyutunu verir
Console.WriteLine("Ortalama: {0}", sayilar.Average()); // Dizi ortalamasını verir
Console.WriteLine("En büyük eleman: {0}", sayilar.Max()); // Dizi max. değer verir
Console.WriteLine("En küçük eleman: {0}", sayilar.Min()); // Dizi min. değer verir
Console.WriteLine("Elemanların toplamı: {0}", sayilar.Sum()); // Dizi toplamı verir
```

C# Array Sınıfına Ait Metotlar

Sort() metodu: Dizi sınıfına ait olan metot sayısal veri tipine sahip ise küçükten büyüğe doğru, alfasayısal veri tipine sahip dizilerde ise A'dan Z'ye sıralama yapar.

```
int[] sayilar = { 5, 10, 15, 2, 3 };
Array.Sort(sayilar);
foreach (int s in sayilar)
   Console.Write(s + " "); // 2 3 5 10 15
Console.ReadKey();
```

Reverse() metodu: Bu metot sayesinde dizinin mevcut sırası terslenir. Burada dikkat edilecek olan şey kesinlikle sıralama yapılmaz.

```
int[] sayilar = { 5, 10, 15, 2, 3 };
Array.Reverse(sayilar);
foreach (int s in sayilar)
   Console.Write(s + " "); // 3 2 15 10 5
Console.ReadKey();
```

Clear() metodu: Dizinin belirli bir index değerinden başlayarak istenilen sayıda elemanın silineceğini (default değerini) gösterir. Clear metodunda üç farklı parametre vardır. Birinci parametre dizinin adı, ikinci parametre silme işleminin başlanacağı index numarası, üçüncü parametre ise silinecek eleman sayısıdır.

```
int[] sayilar = { 5, 10, 15, 2, 3 };
Array.Clear(sayilar, 2, 2);
foreach (int s in sayilar)
   Console.Write(s + " "); // 5 10 0 0 3
Console.ReadKey();
```

IndexOf() metodu: Bu metot dizinin elemanları içerisinde arama yapmak için kullanılır. Aranan eleman bulunursa bulunan değerin index değeri geriye döndürülür, bulamazsa geriye -1 değerini döndürür.

```
int [] sayilar = { 5, 10, 15, 2, 3 };
Console.Write(Array.IndexOf(sayilar, 3)); // 4
Console.ReadKey();
```

Nesne Tabanlı Programlama I – 03.11.2017

C# Dilinde Listeler

Kullanımı diziler ile neredeyse aynıdır. En temel fark dizilerin kaç elemanlı olacağını belirtmek zorunlu iken listelerde böyle bir zorunluluk yoktur. Listelerde gerek görüldükçe eleman eklenebilir. Ayrıca diziler tek tip elemanları saklarlar. Listelerde ise tanımlama bilgisi "object" olursa farklı veri türlerinden elemanlar listeye eklenebilir. Bir liste aşağıdaki gibi tanımlanır:

`List <int> listeAdi = new List<int>();`	`List` → Liste oluşturma komutu `<int>` → Listenin veri tipi `listeAdi` → Listenin adı `new` → Oluşturma komutu

Bir listeye veri eklemek için "Add" komutu kullanılır. Listeye eklenmiş veriler ise [ ] parantezleri içerisine yazılacak index numarasına göre çekilebilir. Dizilerde olduğu gibi ilk verinin index değeri her zaman sıfırdır.

```
List <int> liste = new List<int>(); // Liste adında sayısal bir liste oluşturulur.
liste.Add(5); // Listenin 0. index numarasına 5 sayısal değeri eklenir.
liste.Add(10); // Listenin 1. index numarasına 10 sayısal değeri eklenir.
liste.Add(15); // Listenin 2. index numarasına 15 sayısal değeri eklenir.
Console.WriteLine(liste[1]); // Ekrana 10 değeri yazdırılır.
```

Farklı veri türlerini bir arada listeye eklemek için "object" metodu aşağıdaki gibi kullanılır;

```
List <object> liste = new List<object>(); // Her türü içerebilen bir liste oluşturulur.
liste.Add(5); // Dizinin 0. index numarasına 5 sayısal değeri eklenir.
liste.Add("C#"); // Dizinin 1. index numarasına "C#" string değeri eklenir.
liste.Add('A'); // Dizinin 2. index numarasına 'A' char değeri eklenir.
liste.Add(true); // Dizinin 3. index numarasına 'true' bool değişkeni eklenir.
```

Listelerde silme işlemi Remove() ve RemoveAt() metotları ile yapılır. Remove() metodunda silinmesi istenen değer aynen yazılır. Silme işlemi gerçekleşirse geriye "true" değer döndürür, silme işlemi gerçekleşmezse geriye "false" değerini döndürür. RemoveAt() metodunda ise index numarasını kullanarak silme işlemi yapılır. Bu metotta geriye değer döndürülmez.

```
List <object> liste = new List<object>(); // Her türü içerebilen bir liste oluşturulur.
liste.Add(5); // Dizinin 0. index numarasına 5 sayısal değeri eklenir.
liste.Add("C#"); // Dizinin 1. index numarasına "C#" string değeri eklenir.
liste.Add('A'); // Dizinin 2. index numarasına 'A' char değeri eklenir.
liste.Add(true); // Dizinin 3. index numarasına "true" bool değişkeni eklenir.
liste.Add(true); // Dizinin 4. index numarasına "true" bool değişkeni eklenir.
liste.Remove(true); // İlk "true" bool değerini bulur ve siler.
liste.RemoveAt(0); // Sıfırıncı index numarasını listeden siler.
```

Belirli bir index değerinden başlayarak istenen sayıda eleman silmek için RemoveRange() metodu kullanılır. Bu komutun iki tane parametresi vardır. İlk parametre silinmeye başlanacak olan index numarasını, ikinci parametre ise silinecek eleman sayısını gösterir.

```
liste.RemoveRange(0, 2); // 0. index'ten başlar ve sonraki 2 index'i siler.
```

Dizilerde olduğu gibi listelerde de küçükten büyüğe sıralama işi için Sort() metodu, listeyi tersine çevirmek için Reverse() metodu kullanılır.[1]

```
List <int> liste = new List<int>(); // Liste adında sayısal bir liste oluşturulur.
liste.Add(7); // Listenin 0. index numarasına 7 sayısal değeri eklenir.
liste.Add(3); // Listenin 1. index numarasına 3 sayısal değeri eklenir.
liste.Add(15);// Listenin 2. index numarasına 15 sayısal değeri eklenir.
liste.Add(4); // Listenin 3. index numarasına 4 sayısal değeri eklenir.
liste.Sort(); // Listede küçükten büyüğe doğru sıralama yapılır.
liste.Reverse(); // Listedeki index numaralarını sondan başa sıralar.
```

C# Dilinde Fonksiyonlar *(Metotlar)*

Belirli bir işi gerçekleştirmek için yazılan alt programlara metot adı verilir. C# dilinde her programda sadece bir tane Main() ana metodu vardır. Main() metodu dışındaki hiçbir metot tek başına çalıştırılamaz. Metotlar mutlaka bir sınıf (class) içerisinde tanımlanmalıdır. Metot kullanılarak;

1) Yazılan programın okunabilirliği arttırılır.
2) Tekrarlar önlenerek kaynak kodun büyümesi engellenir.
3) Oluşan hataların tespiti ve giderilmesi kolaylaşır.

Metotların ana metot (`Main()`) içerisinden çağırılması bir sonraki sayfada gösterilmiştir:

[1] Yaptığım denemeler sonucunda farklı veri tiplerinde Sort() metodunun işe yaramadığını gördüm.

```
static void Main(string[] args)
{
   void Metot1()
   {
      // yapılacak işlemler
   }
   void Metot2()
   {
      // yapılacak işlemler
   }
   Metot1(); // Metod1'in çağrılması
   Metot2(); // Metod2'in çağrılması
}
```

Bir metodun genel yapısı aşağıdaki gibidir;

`public void yaz (string metin)`	`public` → Erişim belirteci `void` → Geri dönüş tipi `yaz` → Metot adı `(string metin)` → Parametre(ler)

C# Dilinde Erişim Belirteçleri

Metotlar public, internal, protected ve private olarak tanımlanabilirler. Erişim belirteci kullanılmazsa varsayılan değer olarak private kabul edilir.

- Public olarak tanımlanan metot tanımlı olduğu sınıf *(class)* dışındaki sınıflardan da çağrılabilir. Ayrıca farklı projelerdeki sınıflar da dahil olmak üzere bütün sınıflardan erişilebilir.
- Internal metotlara aynı Assembly *(Makine dili)* içerisindeki projelere ait sınıflardan erişilebilir.
- Protected metotlara sadece bir alt sınıftan erişilebilir.
- Private metotlara ise sadece tanımlı olduğu sınıf içerisinden erişilebilir.

Bu metotların kullanılması için tanımlı olduğu sınıflardan nesne türetilmesi gerekir. Nesne türetme işlemi aşağıdaki gibi yapılır;

```
SinifAdi.MetotAdi
```

Fakat erişim ifadesinden sonra "static" sözcüğü kullanılırsa metot doğrudan kullanılabilir.

C# Dilinde Metotların Geri Dönüş Tipleri

Metotlar çağrıldıkları diğer metotlara değer döndürebilirler. Döndürdükleri değerin tipi bu kısma yazılır. Eğer metot herhangi bir değer döndürmeyecekse "void" ifadesi yazılır.

C# Dilinde Parametreler

Bazı durumlarda metotların çalışabilmesi için birtakım değerlere ihtiyaç duyarlar. Bu değerlere parametre adı verilmektedir.

Örnek 1: Çağrıldığında ekrana "Merhaba!" yazan programı geriye değer döndürmeyecek bir metot tanımlayarak hazırlayınız.

```
static void Main(string[] args)
{
   yaz();
}
static void yaz()
{
   Console.WriteLine("Merhaba!");
}
```

Örnek 2: Klavyeden girilen iki adet tamsayının toplamını bularak ekrana yazdıran programı geriye değer döndürmeyen fonksiyonu tanımlayarak hazırlayınız.

```
static void topla(int a, int b)
{
   Console.Write("Sonuç: " + (a + b));
}

static void Main(string[] args)
{
   Console.Write("1. sayıyı giriniz: ");
   int x = int.Parse(Console.ReadLine());
   Console.Write("2. sayıyı giriniz: ");
   int y = int.Parse(Console.ReadLine());
   topla(x, y);
}
```

Örnek 3: Klavyeden girilen iki adet tamsayının toplamını bularak ekrana yazdıran programı geriye değer döndürecek şekilde tanımlayarak hazırlayınız.

```
static void Main(string[] args)
{
   Console.Write("1. sayıyı giriniz: ");
   int x = int.Parse(Console.ReadLine());
   Console.Write("2. sayıyı giriniz: ");
   int y = int.Parse(Console.ReadLine());
   Console.Write("Sonuç: " + topla(x, y));
}

static int topla(int a, int b)
{
   return a + b;
}
```

Örnek 4: Klavyeden girilen bir tamsayının negatif, pozitif veya sıfıra eşit olma durumunu inceleyecek programı "static" sözcüğü kullanmadan ve geriye değer döndürmeyen bir fonksiyon tanımlayarak C# dilinde hazırlayınız.

```
class Program
{
    static void Main(string[] args)
    {
        Console.Write("Sayıyı giriniz: ");
        int x = int.Parse(Console.ReadLine());
        Program pr = new Program();
        pr.Denetle(x);
        Console.ReadKey();
    }

    void Denetle(int sayi)
    {
        if (sayi < 0)
            Console.Write("Sayı negatif.");
        else if (sayi > 0)
            Console.Write("Sayı pozitif.");
        else
            Console.Write("Sayı sıfıra eşit.");
    }
}
```

Yukarıdaki programda metot "static" olarak tanımlanmadığından dolayı bu metoda doğrudan erişim mümkün değildir. Metoda ulaşabilmek için ait olduğu sınıftan bir nesne türetilmelidir. Daha sonra o türetilen nesne üzerinden metoda erişilebilir.

Örnek 5: Klavyeden girilen bir tamsayının tek veya çift olduğunu bulan programı hazırlarken public erişim belirtecini static tanımlayarak hazırlayınız.

```
class Program
{
   static void Main(string[] args)
   {
      Console.Write("Sayıyı giriniz: ");
      int x = int.Parse(Console.ReadLine());
      if (Metod.TekCift(x))
         Console.WriteLine("Sayı çift sayıdır.");
      else
         Console.WriteLine("Sayı tek sayıdır.");
   }

   class Metod
   {
      public static Boolean TekCift(int sayi)
      {
         if (sayi % 2 == 0)
            return true;
         else
            return false;
      }
   }
}
```

Nesne Tabanlı Programlama I – 10.11.2017 *(Vize Öncesi Son Ders)*

C# Dilinde Hazır Metotlar

1. C# Dilinde String Fonksiyonlar

1) Trim() Metodu → Metinsel bir ifadenin başındaki ve sonundaki boşlukların atılması için kullanılır. TrimStart() ve TrimEnd() olmak üzere iki farklı türü vardır. TrimStart() metodunda metnin başındaki boşluklar, TrimEnd() metodunda ise metnin sonundaki boşluklar atılır.

```
string a;
a = " Merhaba ";
Console.Write(a.Trim());
Console.ReadKey();
```

```
string a;
a = "   Gökhan   ";
Console.WriteLine(">>" + a.Trim() + "<<");
Console.ReadKey();
```

2) Remove() Metodu → Metindeki belirtilen karakterlerden sonraki karakterlerin silinmesi için kullanılır. Ayrıca istenirse belirtilen karakterden sonra kaç karakterin silineceği de parametre olarak belirlenebilir.

```
string a;
a = "Merhaba";
Console.Write(a.Remove(3)); // 3. index'ten sonrasını siler.
// Console.Write(a.Remove(3,2)); // 3. index'ten itibaren 2 karakter siler.
Console.ReadKey();
```

3) SubString() Metodu → Metindeki belirtilen karakterden sonrasının alınmasını sağlar. Ayrıca istenirse belirtilen karakterden sonra kaç karakterin alınacağı da parametre olarak verilebilir.

```
string a;
a = "Merhaba";
Console.Write(a.Substring(3)); // haba
// Console.Write(a.Substring(3,2));
Console.ReadKey();
```

4) **Split() Metodu** → Bu metot, belirtilen karaktere göre metinsel ifade parçalara bölünebilir. Bu metot geriye string dizisi döndürür. Aşağıdaki kod bloğunda meyveler değişkeni string dizisi türünde tanımlanmıştır. Bu değişkende de meyve isimleri virgül ile ayrılmıştır. Tek tek meyvelere erişebilmek için Split() metodundan virgül işareti kullanılarak meyveler string'i parçalara ayrılmıştır. Bu sayede tüm meyve isimleri ayrılarak meyve dizisinin elemanlarını oluşturmuştur.

```
string meyveler = "Elma, Armut, Kayısı, Erik";
string [] meyve = meyveler.Split(',');
for (int i = 0; i < meyve.Length; i++)
{
   meyve[i] = meyve[i].Trim();
   Console.WriteLine(meyve[i]);
}
```

5) **String.Compare() Metodu** → Metoda verilen iki string değerin alfabetik olarak karşılaştırılması yapılır. Sırasıyla -1, 0 ve 1 değerleri döndürülür. İlk string alfabetik olarak geride ise -1, iki string birbirine eşit ise 0, birinci string alfabetik olarak önde ise 1 değeri döndürülür.

```
Console.Write(String.Compare("abc", "xyz")); // -1
Console.ReadKey();
```

6) **Equals() Metodu** → Metoda gönderilen iki string değer birbirine eşit ise true değer, değil ise false değeri döndürülür.

```
Console.Write(String.Equals("Samsun", "Samsun")); // True
```

2. C# Dilinde Matematiksel *(Math)* Metotları

1) **Abs() Metodu** → Parametre olarak girilen sayının mutlak değerini döndürür.

```
Console.Write(Math.Abs(-5)); // 5
```

2) **Ceiling() Metodu** → Parametre olarak girilen ondalık sayıyı bir üstteki tamsayıya yuvarlar.

```
Console.Write(Math.Ceiling(10.3)); // 11
```

3) **Equals() Metodu** → Object veri tipinde verilen iki parametreyi karşılaştırarak eşit olmaları durumunda true değer, olmamaları durumunda da false değer döndürür.

```
Console.Write(Math.Equals(10, 10)); // True
```

4) **Min() ve Max() Metotları** → Parametre olarak girilen iki değerin büyük olanını bulan Max() fonksiyonu, küçük olanı bulan ise Min() fonksiyonudur.

```
Console.Write(Math.Min(10, 20)); // 10
Console.Write(Math.Max(10, 20)); // 20
```

5) **Round() Metodu** → Parametre olarak girilen ondalıklı ifadenin ondalık kısmını en yakın tamsayıya yuvarlar. Ayrıca, istenirse sayının ondalık kısmında kaç basamağın gösterileceğini parametre olarak girebiliriz.

```
Console.Write(Math.Round(10.7542,2)); // 10,75
```

6) **Sqrt() Metodu** → Parametre olarak girilen sayının karekökünü alır.

```
Console.Write(Math.Sqrt(64)); // 8
```

3. C# Dilinde DateTime Sınıfı Metotları

- `DateTime.Now` → Tarih ve saat bilgisini verir. → *(10.11.2017 19:48:34)*
- `DateTime.Now.ToLongDateString()` → Uzun formatlı tarih verir. → *(10 Kasım 2017 Cuma)*
- `DateTime.Now.ToShortDateString()` → Kısa formatlı tarih verir. → *(10.11.2017)*
- `DateTime.Now.ToLongTimeString()` → Uzun formatlı zaman bilgisini verir. → *(19:48:47)*
- `DateTime.Now.ToShortTimeString()` → Kısa formatlı zaman bilgisini verir. → *(19:48)*
- `DateTime.Now.DayOfWeek` → Bulunduğun günü verir. → *(Cuma)*
- `DateTime.Now.DayOfYear` → Yılın kaçıncı günü bilgisini verir. → *(314)*
- `DateTime.Now.Hour` → Sistemdeki saati verir. → *(19)*
- `DateTime.Now.Month` → Sistemdeki ay bilgisini verir. → *(11)*
- `DateTime.Now.Year` → İçinde bulunduğumuz yılı verir. → *(2017)*
- `DateTime.Today` → Bugünün tarihini verir. → *(10.11.2017 00:00:00)*
- `DateTime.Now.AddDays(1)` → Bugünün tarihine gün ekler. → *(11.11.2017 19:48:47)*
- `DateTime.Now.AddHours(2)` → Bugünün tarihine saat ekler. → *(11.11.2017 19:48:47)*
- `DateTime.Now.Day` → İçinde bulunduğumuz günü verir. → *(10)*

Örnek: Klavyeden girilen doğum tarihi bilgisine göre kişinin kaç gün yaşadığını bulan programı C# dilinde hazırlayınız.

```
DateTime dogumTarihi;
int yasadigiGun;
int fazlaGun;
yasadigiGun = 0;
Console.Write("Doğum tarihi 24.12.2000 veya 24/12/2000 formatında girin: ");
dogumTarihi = Convert.ToDateTime(Console.ReadLine());
fazlaGun = (DateTime.Today.Year - dogumTarihi.Year) / 4;
yasadigiGun = (DateTime.Today.Year - dogumTarihi.Year) * 365 +
(DateTime.Today.DayOfYear - dogumTarihi.DayOfYear) + fazlaGun;
Console.Write("Yaşadığı gün: " + yasadigiGun);
```

- TimeSpan() metodu tarihler arasındaki işlemi gün, saat, dakika, saniye ya da sanise cinsinden döndürür. Yukarıdaki örneğin TimeSpan() ile çözümü aşağıda gösterilmiştir;

```
DateTime dogumTarihi;
TimeSpan yasadigiGun;
Console.Write("Doğum tarihi 24.12.2000 veya 24/12/2000 formatında girin: ");
dogumTarihi = Convert.ToDateTime(Console.ReadLine());
yasadigiGun = (DateTime.Today - dogumTarihi);
Console.Write("Yaşadığı gün: " + yasadigiGun);
```

C# Dilinde Random Sınıfı

Rastgele sayı üretmek için kullanılan bir sınıftır. Sayı üretebilmek için Random sınıfından bir nesne türetilmesi gerekir. Random sınıfına ait Next() metodu ile sayılar üretilmeye başlar. Next metodunun üç farklı kullanımı vardır.

1) Next metodu aşağıdaki gibi kullanılırsa negatif olmayan rastgele sayılar üretir.

```
Random s = new Random();
Console.Write(s.Next());
```

2) Parantez içerisindeki sayı üst sınır olmak üzere *(üst sınır hariç)* negatif olmayan sayı üretir.

```
Random s = new Random();
Console.Write(s.Next(15));
```

3) Parantez içerisinde alt ve üst sınır parametre olarak girilir. Alt sınır dahil, üst sınır hariç olacak şekilde negatif olmayan sayılar üretir.

```
Random s = new Random();
Console.Write(s.Next(5, 15));
```

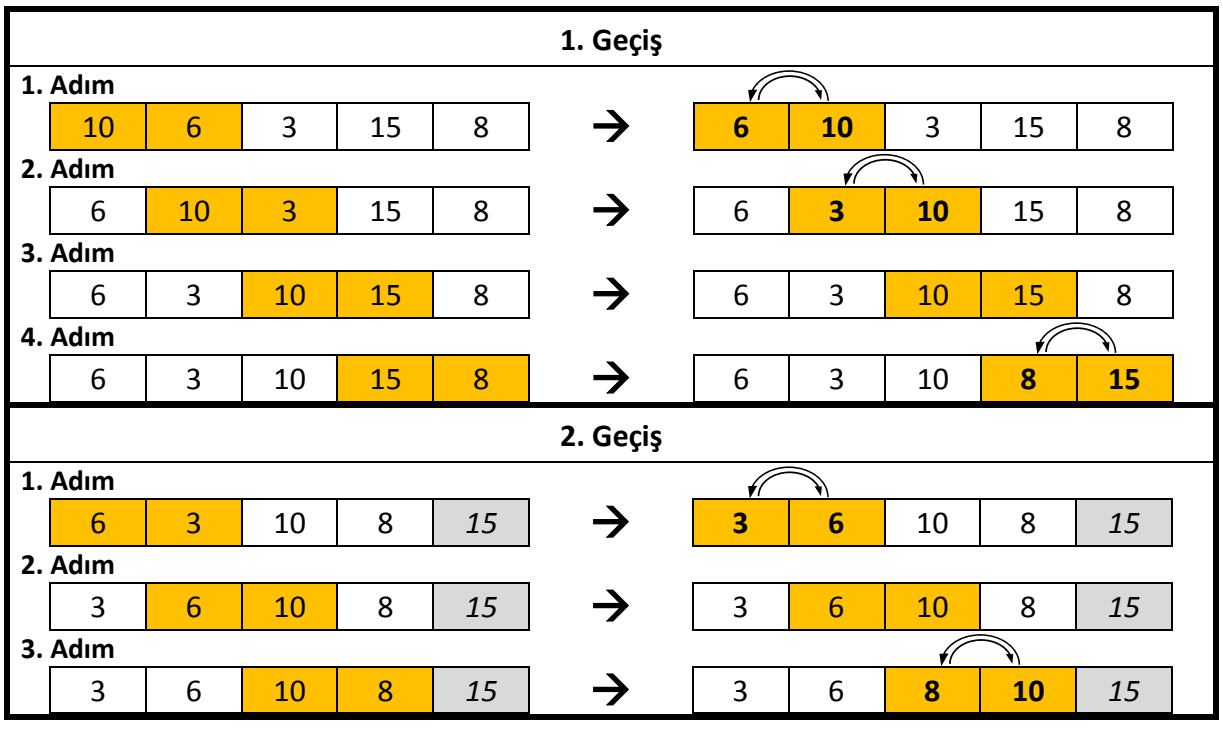
Nesne Tabanlı Programlama I – 01.12.2017 *(Vize Sonrası İlk Ders)*

C# Dilinde Sıralama Algoritmaları

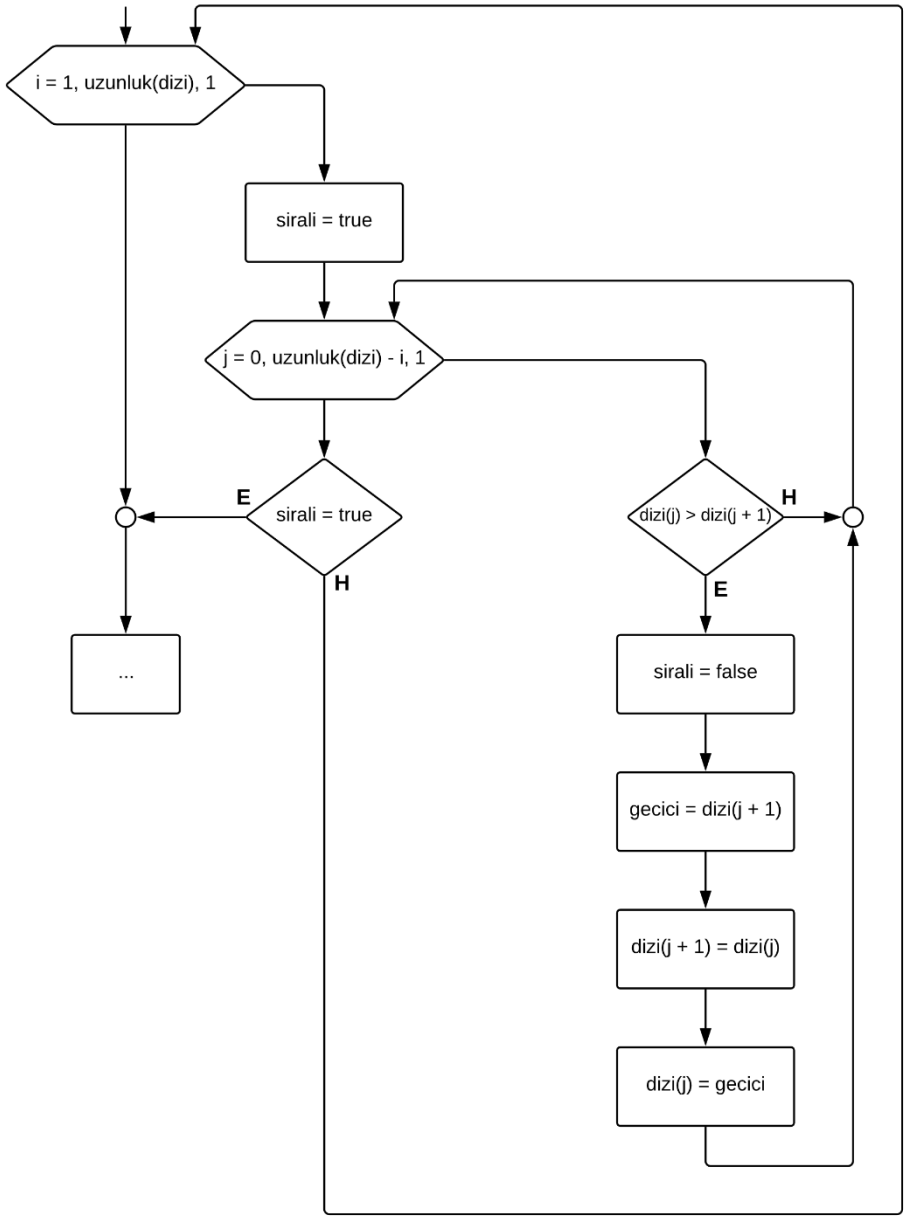
1) Kabarcık Sıralama (Bubble Sort) Algoritması

Her defasında listenin iki elemanını karşılaştırarak gerekliyse yer değişikliği yapılır. Bu işlem hiçbir yer değişikliği yapılmadığı ana kadar devam eder. Her geçişte en büyük sayı listenin en sağına aktarılmış olur. Kabarcık sıralaması algoritmasında yapılan işlem listeyi küçükten büyüğe doğru sıralamaktır.

Örnek: Liste = { 10, 6, 3, 15, 8 } listesini kabarcık sıralaması algoritmasını kullanarak çözünüz.



Algoritmanın çalışma prensibi dikkatle incelendiğinde ikinci geçişin sonunda sıralamanın gerçekleştiği görülmektedir. Kodlama yapılırken sıralamanın gerçekleşip gerçekleşmediğini kontrol ederek gereksiz adımların önüne geçebiliriz. Kod bloğundaki kabarcık sıralama metodunda sıralı isimli değişken bu amaçla kullanılmıştır. Listenin küçükten büyüğe sıralanması için uygulanan kabarcık sıralama metodu aşağıda akış diyagramı çizilerek gösterilmiştir.



Yukarıdaki akış diyagramının kodlamasını C# dilinde yazalım;

```
static void Main(string[] args)
{
   int [] sayilar = { 10, 6, 3, 15, 8 };
   diziYaz(sayilar);
   kabarcikSirala(sayilar);
   diziYaz(sayilar);
   Console.ReadKey();
}

public static void kabarcikSirala(int [] dizi)
{
   int gecici;
   bool sirali;

   for (int i = 1; i < dizi.Length; i++)
   {
      sirali = true;
      for (int j = 0; j < dizi.Length - i; j++)
      {
```

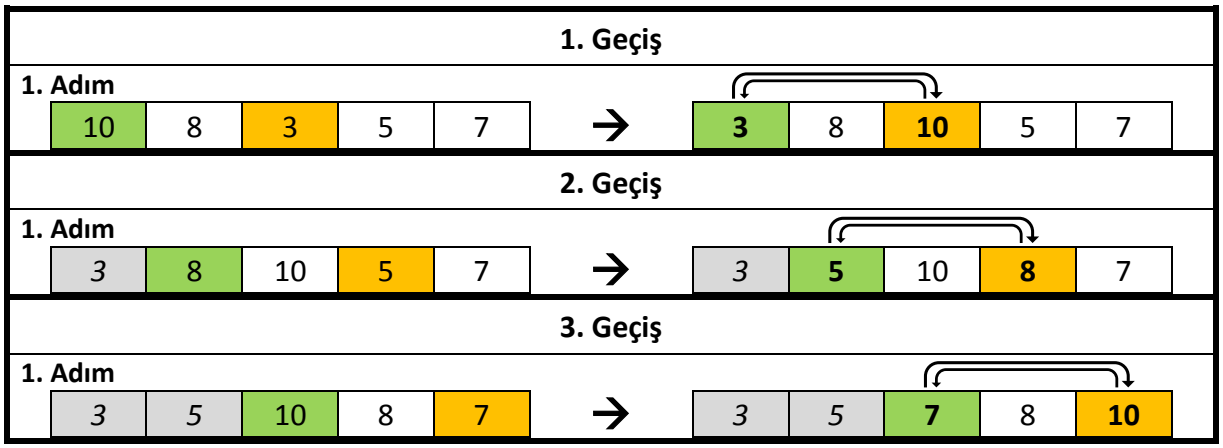
```
            if (dizi[j] > dizi[j + 1])
            {
               sirali = false;
               gecici = dizi[j + 1];
               dizi[j + 1] = dizi[j];
               dizi[j] = gecici;
            }
         }
         if (sirali)
            break;
      }
}

public static void diziYaz(int [] dizi)
{
   for (int i = 0; i <= dizi.Length - 1; i++)
      Console.Write(dizi[i] + " ");
   Console.WriteLine();
}
```

2) Seçmeli Sıralama (Selection Sort) Algoritması

Seçmeli sıralama algoritması listeden sırasıyla en küçük elemanın seçilerek yerine koyulması esasına dayanır. Bu algoritma başlangıçta listenin ilk elemanını en küçük değer olarak kabul eder.

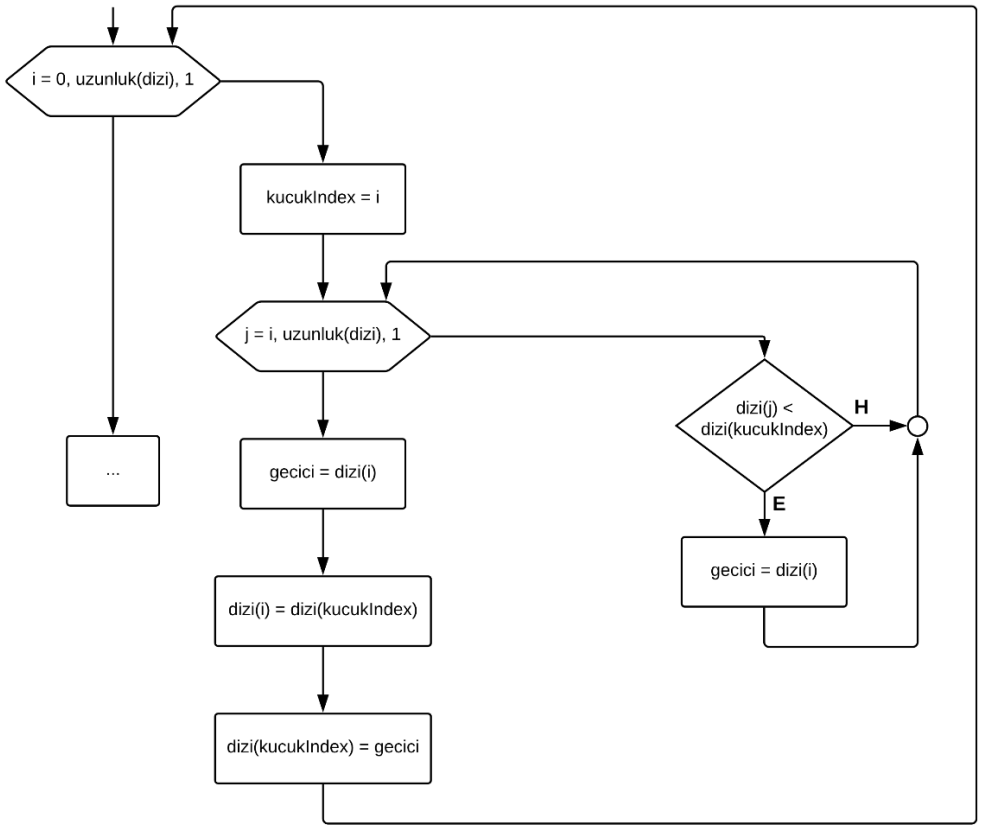
Örnek: Liste = { 10, 8, 3, 5, 7 } listesini seçmeli sıralama algoritmasını kullanarak çözünüz.



4. Geçiş										
1. Adım										
	3	*5*	*7*	8	10	→	*3*	*5*	*7*	8	10

4. geçişte görüldüğü üzere yer değişikliğine gerek duyulmamıştır.

Yukarıda bahsettiğimiz işi bir de akış diyagramına dökelim.



Yukarıdaki yaptığımız akış diyagramını bir de C# dilinde yazalım;

```
public static void secmeliSirala(int [] dizi)
{
    int gecici, kucukIndex, i, j;
    for (i = 0; i < dizi.Length; i++)
    {
       kucukIndex = i;
       for (j = i; j < dizi.Length; j++)
       {
          if (dizi[j] < dizi[kucukIndex])
             kucukIndex = j;
       }
       gecici = dizi[i];
       dizi[i] = dizi[kucukIndex];
       dizi[kucukIndex] = gecici;
    }
}

public static void diziYaz(int [] dizi)
{
   for (int i = 0; i < dizi.Length; i++)
      Console.Write(dizi[i] + " ");
   Console.WriteLine();
}
```

```
static void Main(string[] args)
{
   int [] sayilar = { 10, 8, 3, 5, 7 };
   diziYaz(sayilar);
   secmeliSirala(sayilar);
   diziYaz(sayilar);
   Console.ReadKey();
}
```

3) Eklemeli Sıralama (Insertion Sort) Algoritması

Basit bir algoritma mantığı olan eklemeli sıralama algoritması verimsiz bir sıralama yöntemidir. Çalışma mantığı olarak listenin ikinci elemanı ile ilk elemanı birinci adımda karşılaştırılır. Gerekli ise yer değişikliği yapılır. Bir sonraki adımda ise listenin üçüncü elemanı ile kendinden önceki elemanlar tek tek karşılaştırılır ve gerekli ise yer değişikliği yapılır. Bu işlem son elemana kadar aynen tekrar edilir. Eklemeli sıralama algoritması büyük veritabanlarında tercih edilen bir yaklaşım değildir.

Örnek: Liste = { 2, 5, 5, 8, 10, 7, 4, 12, 9 } listesini eklemeli sıralama algoritmasını kullanarak çözünüz.

1. Adım

2	5	5	8	10	7	4	12	9

2. Adım

2	5	5	8	10	7	4	12	9

3. Adım

2	5	5	8	10	7	4	12	9

4. Adım

2	5	5	8	10	7	4	12	9

5. Adım

2	5	5	8	10	7	4	12	9
2	5	5	7	8	10	4	12	9

6. Adım

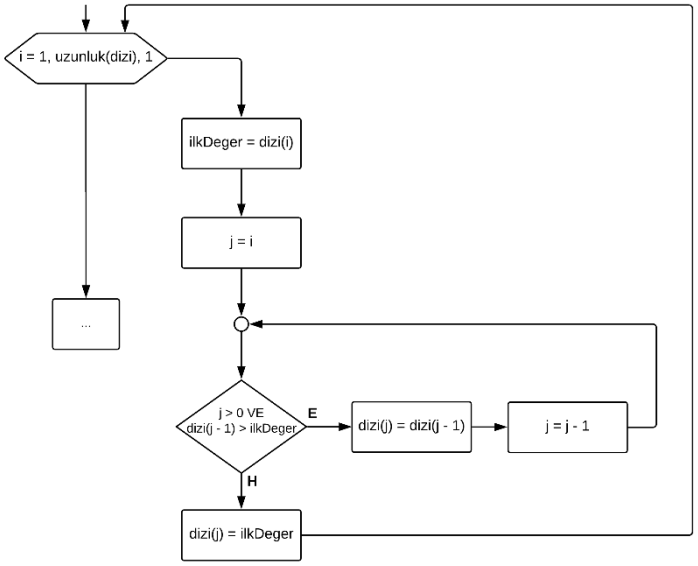
2	5	5	7	8	10	4	12	9
2	4	5	5	7	8	10	12	9

7. Adım

2	4	5	5	7	8	10	12	9

8. Adım

2	4	5	5	7	8	10	12	9
2	4	5	5	7	8	9	10	12



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir;

```
public static void eklemeliSiralama(int[] dizi)
{
   int i, j, ilkDeger;
   for (i = 1; i < dizi.Length; i++)
   {
      ilkDeger = dizi[i];
      j = i;
      while (j > 0 && dizi[j - 1] > ilkDeger)
         {
            dizi[j] = dizi[j - 1];
            j--;
         }
         dizi[j] = ilkDeger;
   }
}

public static void diziYaz(int[] dizi)
{
   for (int i = 0; i < dizi.Length; i++)
      Console.Write(dizi[i] + " ");
   Console.WriteLine();
}
```

```
static void Main(string[] args)
{
   int[] sayilar = {2,5,5,8,10,7,4,12,9};
   diziYaz(sayilar);
   eklemeliSiralama(sayilar);
   diziYaz(sayilar);
   Console.ReadKey();
}
```

4) Kabuk Sıralama (Shell Sort) Algoritması

Bu algoritmada karşılaştırma işlemi belirlenmiş olan atlama sırasına göre daha uzaktaki elemanların karşılaştırılması esasına dayanır. Başlangıçta atlama sayısı eleman sayısının yarısı alınarak hesaplanır. Atlama sayısı her çevrimde *(her geçişte)* bir önceki atlama sayısının yarısı olacak şekilde hesaplanır. Son geçişte atlama sayısı daima 1'dir. Çok sık kullanılan bir algoritmadır.

1. Adım

Atlama sayısı: 9 / 2 = **4**

6	9	5	7	3	4	2	1	8	→	3	4	2	1	6	9	5	7	8

2. Adım

Atlama sayısı: 4 / 2 = **2**

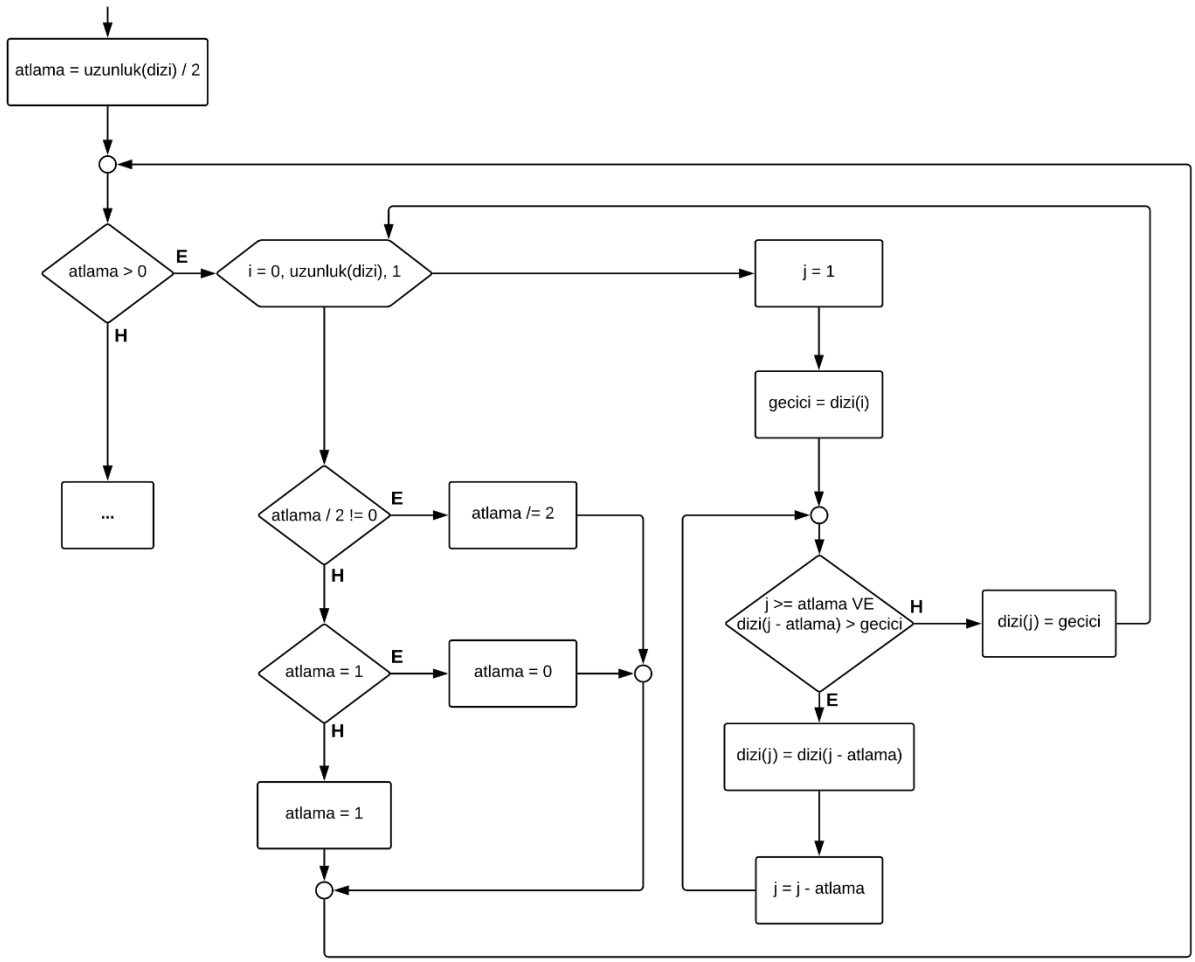
3	4	2	1	6	9	5	7	8	→	2	1	3	4	5	7	6	9	8

3. Adım

Atlama sayısı: 2 / 2 = **1**

2	1	3	4	5	7	6	9	8	→	1	2	3	4	5	6	7	8	9

Yukarıda bahsettiğimiz işi bir de akış diyagramına dökelim.



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir:

```
class Program
{
    static void Main(string[] args)
    {
        int [] sayilar = { 6, 9, 5, 7, 3, 4, 2, 1, 8 };
        diziYaz(sayilar);
        kabukSiralama(sayilar);
        diziYaz(sayilar);
        Console.ReadKey();
    }

    public static void kabukSiralama(int [] dizi)
    {
        int i, j, atlama, gecici;
        atlama = dizi.Length / 2;
        while (atlama > 0)
        {
            for (i = 0; i < dizi.Length; i++)
            {
                j = i;
                gecici = dizi[i];
                while (j >= atlama && dizi[j - atlama] > gecici)
                {
                    dizi[j] = dizi[j - atlama];
                    j = j - atlama;
                }
                dizi[j] = gecici;
            }
            if (atlama / 2 != 0)
                atlama /= 2;
            else if (atlama == 1)
                atlama = 0;
            else
                atlama = 1;
        }
    }

    public static void diziYaz(int [] dizi)
    {
        for (int i = 0; i < dizi.Length; i++)
            Console.Write(dizi[i] + " ");
        Console.WriteLine();
    }
}
```

C# Dilinde Arama Algoritmaları

Listelerden herhangi bir elemanın aranması için kullanılan algoritmalardır. Listeler üzerinde arama yapabilmek için kullanılan üç arama algoritması bulunur. Arama işlemi index numarasına göre yapılır.

1. Doğrusal Arama (Lineer Search) Algoritması

Listedeki tüm elemanlar sırasıyla taranır. Aranan eleman bulunduğunda algoritma sonlanır. Eleman sayısı fazla olan listelerde bu algoritmayı kullanmak performans açısından verimsizdir. Listedeki tüm elemanlara sırasıyla bakıldığından listenin sıralı olması gerekli değildir.

1. Adım

Aranan sayı: **3**

5	1	3	7	6	2

→ 5 ≠ 3 olduğu için sonraki elemana geçilir.

2. Adım

Aranan sayı: **3**

5	**1**	3	7	6	2

→ 1 ≠ 3 olduğu için sonraki elemana geçilir.

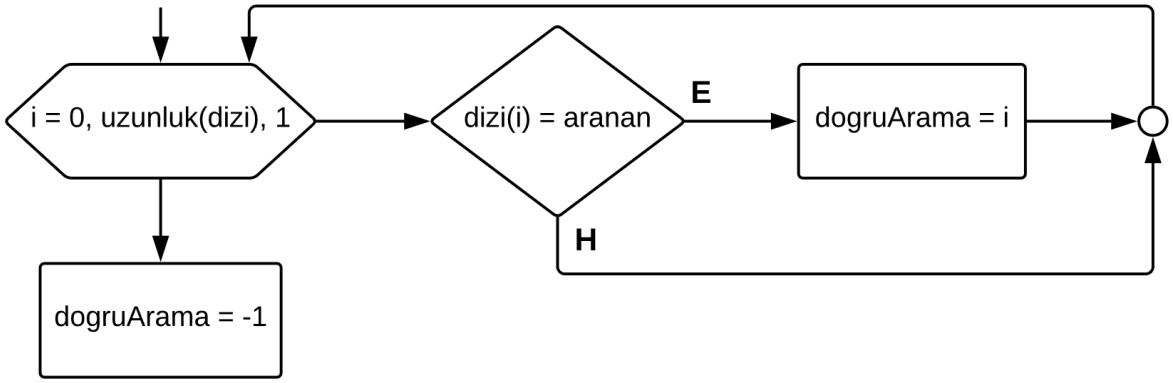
3. Adım

Aranan sayı: **3**

5	*1*	**3**	7	6	2

→ 3 = 3 olduğu için algoritma sonlanır.

Yukarıda bahsettiğimiz işi bir de akış diyagramına dökelim.



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir.

```
class Program
{
    static void Main(string[] args)
    {
        int [] liste = { 5, 1, 3, 7, 6, 2 };
        int aranan = 3;
        int indis = dogrusalArama(liste, aranan);

        if (indis == -1)
            Console.Write("Eleman bulunamadı!");
        else
            Console.Write("Elemanın indisi: " + indis);

        Console.ReadKey();
    }

    public static int dogrusalArama (int [] dizi, int aranan)
    {
        for (int i = 0; i < dizi.Length; i++)
        {
            if (dizi[i] == aranan)
                return i;
        }
        return -1;
    }
}
```

Nesne Tabanlı Programlama I – 15.12.2017

2. İkili Arama (Binary Search) Algoritması

İkili arama algoritması sıralı listeler üzerinde çalışmaktadır. Aranan eleman dizinin ortasındaki eleman ile karşılaştırılır. Ortadaki eleman ile aranan eleman eşitse algoritma sonlandırılır.

a. Aranan eleman, ortadaki elemandan küçükse listenin sol tarafı dikkate alınır.
b. Aranan eleman, ortadaki elemandan büyükse listenin sağ tarafı dikkate alınır.

Bu işlemler eleman bulunana kadar bu işlem devam eder.

Örnek: Liste = { 2, 5, 6, 8, 10, 15, 17, 18, 20, 25, 30 } listesi ve aranan sayımız 25 olsun.[2]

1. Geçiş

Aranan sayı: **25**

Elemanlar	2	5	6	8	10	**15**	17	18	20	**25**	30
Index No.	0	1	2	3	4	5	6	7	8	9	10

$\text{orta} = \frac{0 + 10}{2} = \mathbf{5}$ → 15 ≠ 25 ve 25 > 15 olduğu için listenin sağ tarafı dikkate alınır.

2. Geçiş

Aranan sayı: **25**

Elemanlar	17	18	**20**	**25**	30
Index No.	0	1	2	3	4

$\text{orta} = \frac{0 + 4}{2} = \mathbf{2}$ → 20 ≠ 25 ve 25 > 20 olduğu için listenin sağ tarafı dikkate alınır.

3. Geçiş

Aranan sayı: **25**

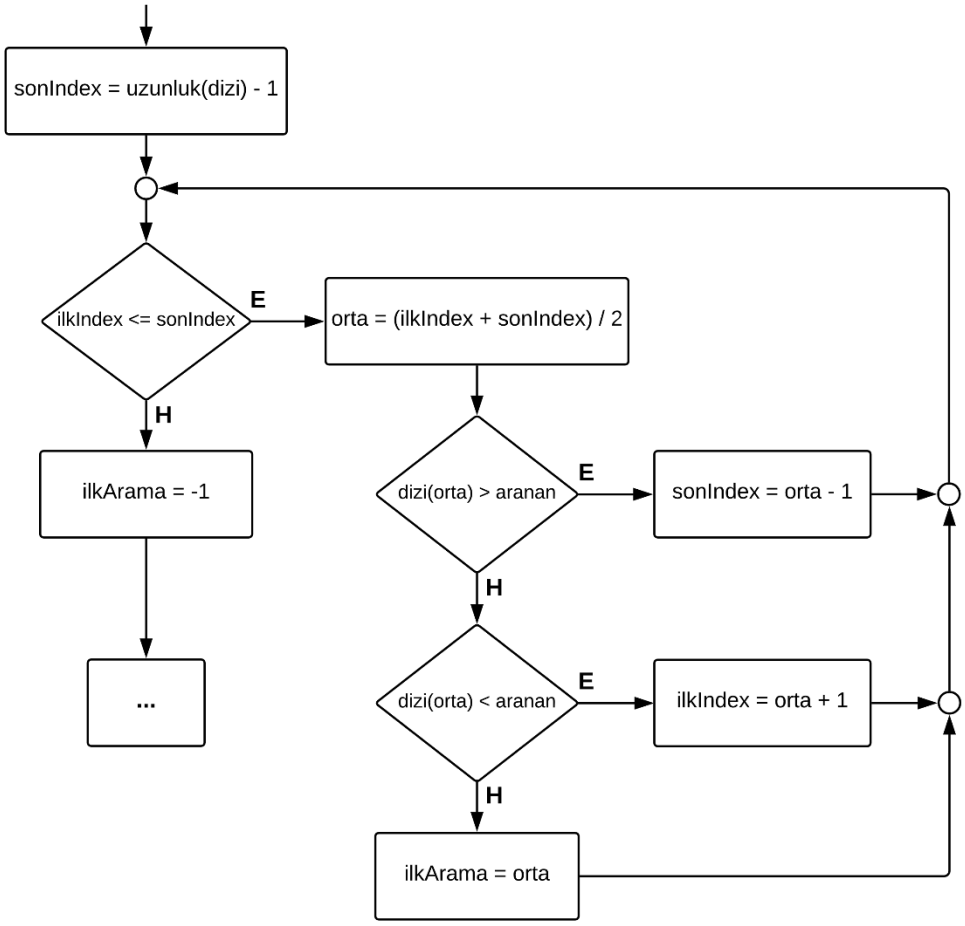
Elemanlar	**25**	30
Index No.	0	1

$\text{orta} = \frac{0 + 1}{2} = 0{,}5 \cong \mathbf{0}$ → 25 = 25 olduğu için algoritma sonlanır.

Not: Üçüncü geçişte görüldüğü üzere değer 0,5 çıkmasına rağmen sonucu sıfır olarak kabul ettik. Bunun nedeni hem tek hem de çift sayıda elemana sahip dizilerde doğru sonuca ulaşabilmek içindir. Yani, eğer ki index numarası bölümündeki orta adı verdiğimiz sonuç virgüllü çıkarsa bir alt değere yuvarlıyoruz.

Yukarıda bahsettiğimiz işi bir de bir sonraki sayfada akış diyagramına dökelim.

[2] İkili sıralamada ortanca değeri bulmak için $\text{orta} = \frac{\text{İlk terim + Son terim}}{2}$ formülü kullanılır.



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir.

```
class Program
{
    public static int ikiliArama(int [] dizi, int aranan)
    {
        int ilkIndex = 0;
        int sonIndex = dizi.Length - 1;
        int orta;
        while (ilkIndex <= sonIndex)
        {
            orta = (ilkIndex + sonIndex) / 2;
            if (dizi[orta] > aranan)
                sonIndex = orta - 1;
            else if (dizi[orta] < aranan)
                ilkIndex = orta + 1;
            else
                return orta;
        }
        return -1;
    }
    static void Main(string[] args)
    {
        int [] liste = { 2, 5, 6, 8, 10, 15, 17, 18, 20, 25, 30 };
        int aranan = 25;
        int index = ikiliArama(liste, aranan);
        if (index == -1)
            Console.WriteLine("Aranan değer bulunamadı!");
        else
            Console.WriteLine("Aranan elemanın index değeri: " + index);
        Console.ReadKey();
    }
}
```

3. Ara Değer Arama (Interpolation Search) Algoritması

Sıralı listelerde arama yapan bir algoritmadır. İkili arama algoritması ile aynı mantıkta çalışır. Tek farkı orta eleman değerinin aşağıdaki formüle göre hesaplanmasıdır.

$$\text{orta} = \text{ilkIndex} + \frac{\big((\text{aranan} - \text{dizi}[\text{ilkIndex}]) * (\text{sonIndex} - \text{ilkIndex})\big)}{(\text{dizi}[\text{sonIndex}] - \text{dizi}[\text{ilkIndex}])}$$

Örnek: Liste = {2, 3, 5, 8, 11, 15, 18, 22, 25, 30} listesi ve aranan değerimiz 22 olsun.

1. Geçiş

Aranan sayı: **22**

Elemanlar	2	3	5	8	11	15	**18**	22	25	30
Index No.	0	1	2	3	4	5	6	7	8	9

$\text{orta} = 0 + \frac{((22-2)*(9-0))}{(30-2)} = \frac{180}{28} = 6{,}42 \cong \mathbf{6}$ → 18 ≠ 22 ve 22 > 18 olduğu için listenin sağ tarafı dikkate alınır.

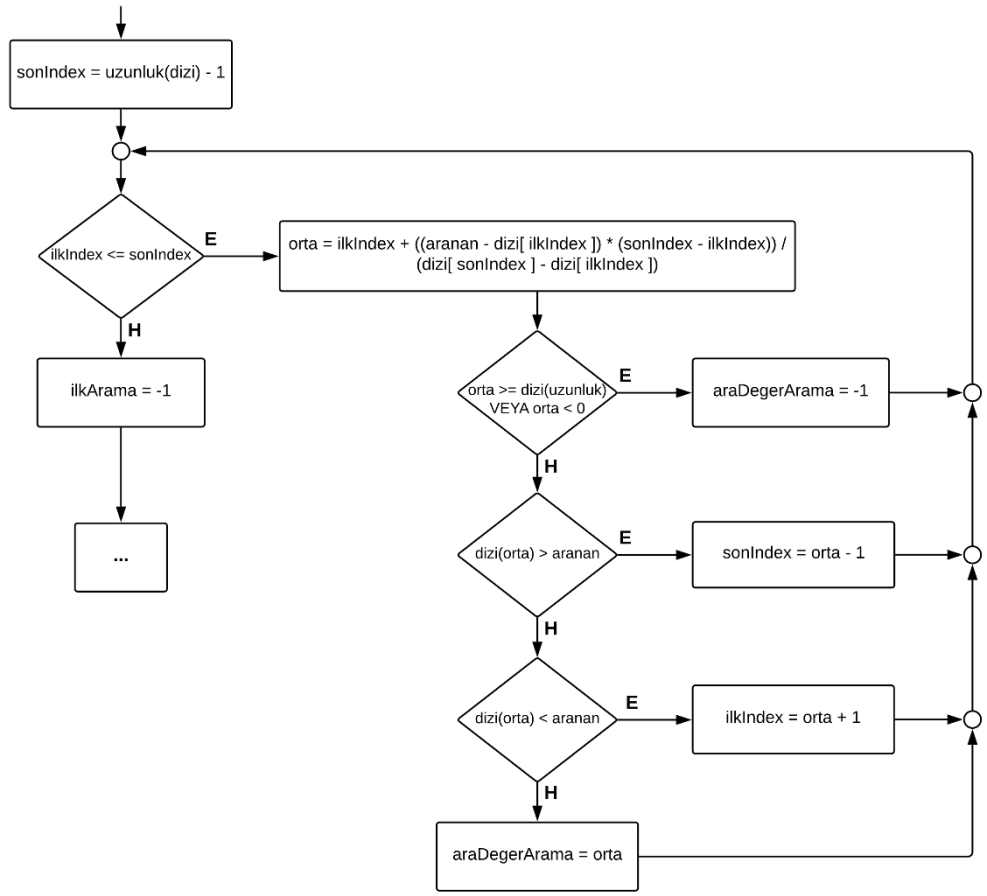
2. Geçiş

Aranan sayı: **22**

Elemanlar	**22**	25	30
Index No.	7	8	9

$\text{orta} = 7 + \frac{((22-22)*(9-7))}{(30-22)} = 7 + \frac{0}{8} = \mathbf{7}$ → 22 = 22 olduğu için algoritma sonlanır.

Yukarıda bahsettiğimiz işi bir de akış diyagramına dökelim.



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir.

```
class Program
{
    public static int aradaDegerArama(int [] dizi, int aranan)
    {
        int ilkIndex = 0;
        int sonIndex = dizi.Length - 1;
        int orta;
        while (ilkIndex <= sonIndex)
        {
            orta = ilkIndex + ((aranan - dizi[ilkIndex]) * (sonIndex - ilkIndex)) /
(dizi[sonIndex] - dizi[ilkIndex]);
            if (orta >= dizi.Length || orta < 0)
                return -1;
            else if (dizi[orta] > aranan)
                sonIndex = orta - 1;
            else if (dizi[orta] < aranan)
                ilkIndex = orta + 1;
            else
                return orta;
        }
        return -1;
    }
    static void Main(string[] args)
    {
        int [] liste = { 2, 3, 5, 8, 11, 15, 18, 22, 25, 30 };
        int aranan = 22;
        int index = aradaDegerArama(liste, aranan);
        if (index == -1)
            Console.WriteLine("Aranan değer bulunamadı!");
        else
            Console.WriteLine("Aranan elemanın index değeri: " + index);
        Console.ReadKey();
    }
}
```

C# Dilinde Dosyalama İşlemleri

Dosyalama işlemleri klasör ve dosya işlemleri olmak üzere ikiye ayrılır. Bu işlemler yapılırken programın başına "`using`" anahtar sözcüğünden sonra "`System.IO`" ön işlemcisi eklenmelidir. Klasör işlemlerinde sıklıkla "`Directory`" sınıfı, dosya işlemlerinde ise "`File`" sınıfı tercih edilir. Her iki sınıfta da kullanılan metoda bağlı olarak verilen parametre değeri ve bu parametrenin oluşturacağı yol daima string şeklinde ve (`@"..."`) olarak verilmelidir.

1. Klasör İşlemleri

a. Klasör Oluşturmak

Directory sınıfını kullanarak klasör oluşturmak için "CreateDirectory" komutunu kullanırız.

```
Directory.CreateDirectory(@"C:\Ornek"); // Ornek adında bir klasör oluşturur.
```

b. Klasör Silmek

Directory sınıfını kullanarak klasör silmek için "Delete" komutunu kullanırız.

```
Directory.Delete(@"C:\Ornek"); // Ornek adındaki klasörü siler.
```

Yukarıdaki komut satırında klasörün bulunamaması veya klasörün içi dolu olma durumunda hata ile karşılaşılabilir. Bu durumda "false" komutu eklenerek silme işlemi iptal edilir.

```
Directory.Delete(@"C:\Ornek", false);
// Ornek klasörünün içerisi boş olmadığı için işlem iptal edilir.
```

Eğer ki içi dolu olan bir klasör silinecekse “true” komutu eklenir. Böylece ilgili klasörün içi boş da olsa dolu da olsa her şekilde silinecektir.

```
Directory.Delete(@"C:\Ornek", true);
```

c. Klasör Kontrolü

Klasörün bulunup bulunmadığını kontrol eden bu metot eğer ki klasörü bulursa “true”, bulamazsa “false” değerini geri döndürür.

```
Directory.Exists(@"C:\Ornek");
```

d. Klasör Taşımak

Bir klasörü başka bir klasörün içerisine taşımak için kullanılır. En az iki parametre içermelidir.

```
Directory.Move(@"C:\Users\Ornek", @"C:\Desktop\Deneme");
//      Taşınacak klasörün yolu , Taşınacak yerin yolu
```

e. Klasör Listelemek

Bir klasörün içerisindeki klasörleri listelemek için kullanılır.

```
Directory.GetDirectories(@"C:\");
```

```
string [] klasorler = Directory.GetDirectories(@"C:\");
foreach (string klasor in klasorler)
{
   Console.WriteLine(klasor);
}
```

2. Dosya İşlemleri

a. Dosya Oluşturmak

Dosya oluşturmak için kullanılır. Dosya uzantısı değiştirilerek farklı tipte dosyalar oluşturulabilir.

```
File.Create(@"C:\Ornek\deneme.txt");
```

b. Dosya Silmek

```
File.Delete(@"C:\Ornek\deneme.txt");
```

c. Dosya Kontrolü

Dosyanın bulunup bulunmadığını kontrol eden bu metot eğer ki dosyayı bulursa true, bulamazsa false değerini geri döndürür.

```
File.Exists(@"C:\Ornek\deneme.txt");
```

d. Dosya Taşımak

Bir dosyayı başka bir klasörün içerisine taşımak için kullanılır. En az iki parametre içermelidir.

```
File.Move(@"C:\Ornek\deneme.txt", @"C:\Masaustu\Proje.txt");
//      Taşınacak dosyanın yolu , Taşınacak yerin yolu
//      Deneme.txt dosyası taşınan klasöre Proje.txt adıyla taşınır.
```

e. Dosya Kopyalamak

```
File.Copy(@"C:\Ornek\deneme.txt", @"C:\Masaustu\Proje.txt");
```

Hedef adreste aynı isme sahip dosya varsa hata ile karşılaşılır. Aynı isimli dosya bulunduğunda verileri üzerine kaydetmek için “true” komutu eklenir, işlemden vazgeçmek için “false” komutu eklenir.

```
File.Copy(@"C:\Ornek\deneme.txt", @"C:\Masaustu\Proje.txt", true);
```

Kaynaklar:

1- Örneklerle Algoritma ve C# Programlama, Erhan Arı, Seçkin Yayıncılık