Visual Studio.Net -C#

3. HAFTA

Operatörler, Akış kontrol mekanizmaları, Diziler

Operatörler

- Operatörler önceden tanımlanmış birtakım matematiksel ya da mantıksal işlemleri yapmak için kullanılan özel karakterler ya da karakterler topluluğudur.

- Operatörlerin işlem yapabilmek için ihtiyaç duydukları değerlere ise "**operand**" denir.

- **x+y -> "+" operatör, "x" ve "y" operand**
- **++z -> "++" operatör, "z" operand**
-

Operatörler

- Operatörler yapılarına ve işlevlerine göre sınıflandırılabilir

- ***Yapılarına göre***
  - ***Unary (tekil) operatörler***
    - **() , [], ++, _, +, _, !, ~, new, checked , unchecked , typeof, sizeof**
  - ***Binary (İkili) Operatörler***
    - *,/,%, + ,-,<,<=,>,>=, as, is, <<,>>,==,!=,&, ^, | ,&&, ||, = ,*=,/=,%=,+=,-=,<<=, >>=,&=,^=,| =
  - ***Ternary(üçlü) Operatörler***
    - ***? :***

Operatörler

- *İşlevlerine göre*
  - Aritmetik operatörler (++, --, *, /, +, -, %)
  - Karşılaştırma(<,>,!=, ==,as, is)
  - Mantıksal(|| ,&&, !)
  - Bitsel (>>,<<, ~(değil), &, |, ^ (xor-aynı 0,farklı 1))
  - Atama ve işlemli atama(*=, %=, -= vs)
  - Özel amaçlı (? : operatörü (a<b ? min=a : min=b))
    - new operatörü :yeni bir nesne oluşturur.
    - typeof : nesnelerin GetType() metodu gibi bağlı olduğu System türünü verir.
    - Sizeof, (), [],+,-,*,&,->, . , checked, unchecked

Operatör Önceliği

- Bazı ifadelerde birden fazla operatör kullanılmış olabilir. Bu gibi durumlarda operatörlerin belirli bir çalışma sırası olacaktır.
- **Büyükten küçüğe işlem önceliği;**
  - Parantez
  - Üs Alma
  - Çarpma , Bölme
  - Toplama, Çıkarma
    - X = 6 + 8*10
    - = 6 + (8 * 10) = 86
    - y=(7+(6+4)*5)*2 = 114

Operatör Önceliği

- Bazı operatörlerin öncelik sırası aynı olabilir. Çarpma-bölme ya da toplama- çıkarma gibi operatörler aralarında aynı önceliğe sahiptir. Böyle bir durumda sağdan sola ya da soldan sağa öncelik ilişkisi devreye girer.
- **Atama operatörleri hariç bütün binary operatörler soldan sağa önceliğe sahiptir, işlemler bu yönde yapılır.**

```
using System;

class Operatorler1
{
    static void Main()
    {
        int i = 5 * 4 / 6;
        Console.WriteLine(i);

        i = 5 * ( 4 / 6 );
        Console.WriteLine(i);
    }
}
```

Operatör Öncelik Tablosu

Birinci Öncelikliler	x.y (nesne erişim operatörü) , f(x) , dizi[x], x++ , x-- , new , typeof , checked , unchecked
Unary Operatörler	+ , - , ! , ~ , ++x , --x , (tür)x
Çarpma Bölme	* , / , %
Toplama ve Çıkarma	+ , -
Kaydırma Operatörleri	<< , >>
İlişkisel ve Tür Testi	< , > , <= , >= , is , as
Eşitlik Operatörleri	==, !=
Mantıksal VE (AND)	&
Mantıksal Özel Veya (XOR)	^
Mantıksal VEYA (OR)	|
Koşul VE	&&
Koşul VEYA	||
Koşul Operatörü	? :
Atama ve İşlemli Atama	= , *= , /= , %= , += , -= , <<= , >>= , &= , ^= , |=

- Yukarıdan aşağı doğru bir öncelik sıralaması vardır. Aynı satırda belirtilmiş olan operatörler ise aynı önceliğe sahiptirler. Bunlar arasında (atama operatörleri hariç) soldan sağa öncelik ilişkisi söz konusudur.

Operatör Önceliği

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 4 +-6;
            Console.WriteLine(i);
        }
    }
}
```

Çıktı
-2

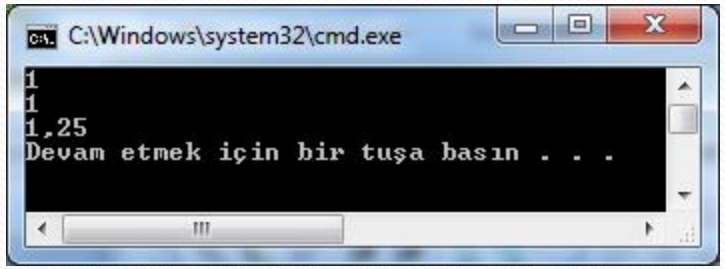
Aritmetik Operatörler

- **+ , -, * ve / operatörleri:**
- Toplama, çıkarma, çarpma ve bölme operatörleridir. İstenilen tipteki sayısal değerlerle kullanılabilir. Bölme işleminde kullanılan değerin tipine göre farklı sonuçlar elde edilebilir.

```
using System;

class Operatorler3
{
    static void Main()
    {
        int i = 5 / 4;
        float f1 = 5 / 4;
        float f2 = 5f / 4f;

        Console.WriteLine(i);
        Console.WriteLine(f1);
        Console.WriteLine(f2);
    }
}
```



Aritmetik Operatörler

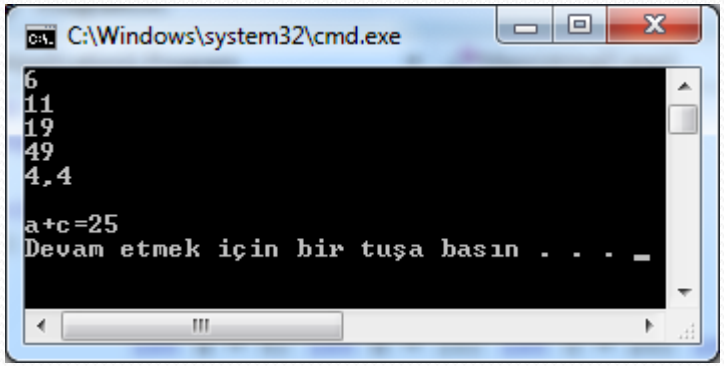
- **% operatörleri:**
  - Bölümden sonra kalanı bulmak yani “mod” almak amacıyla kullanılır Tüm sayısal türler için kullanılabilir.
  - x=10 % 3 -> x= 1
  - y= -7 % 4 -> y= -3
  - f=2.5f % 1.2f ->f = 0.1

- **++ ve -- Operatörleri:**
  - Önüne ya da sonuna geldiği değişkenin değerini 1 arttırır ya da 1 azaltır. İki şekilde kullanılırlar: Önek (prefix) ve Sonek (postfix). Bütün sayısal değerlerde kullanılabilir.

Aritmetik Operatörler

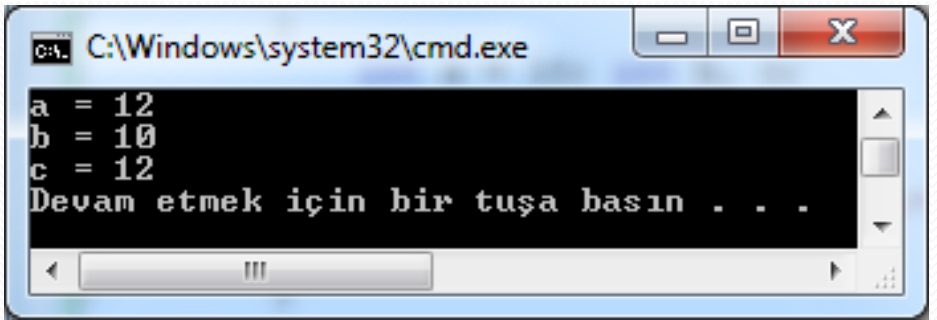
```
using System;

class Program
{
  static void Main(string[] args)
  {
   int a = 5; int b = 10; int c = 20;
   int d = 50; float f = 3.4f;
   a++; ++b; c--; --d;  f++;
   Console.WriteLine("{0}\n{1}\n{2}\n{3}\n{4}\n",a,b,c,d,f);
   Console.WriteLine("a+c={0}",a+c);
  }
}
```



Aritmetik Operatörler

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10; int b, c;
        b = a++; c = ++a;
        Console.WriteLine("a = {0}", a);
        Console.WriteLine("b = {0}", b);
        Console.WriteLine("c = {0}", c);
    }
}
```



Aritmetik Operatörler

- Arttırma ve Azaltma operatörlerinde dikkat edilmesi gereken durumlar şu şekilde özetlenebilir:

```
a = 4
b = a++
    a=5, b=4
```

```
a = 4
b = ++a
    a=5, b=5
```

```
a = 4
b = a--
    a=3, b=4
```

```
a = 4
b = --a
    a=3, b=3
```

Karşılaştırma Operatörü

- İki sayısal değerin birbirlerine göre büyüklüğünü, küçüklüğünü ve eşitliğini kontrol eder. Bu operatörlerin ürettiği değerler “true” ve “false” değerleridir.

= =	Eşittir	!=	Eşit Değildir
<	Küçüktür	>=	Büyük Eşittir
>	Büyüktür	<=	Küçük Eşittir

Karşılaştırma Operatörü

```
using System;
class ilişkisel_operatörler
{
    public static void Main()
    {
        int a = 10, b = 15;
        if(a == b) Console.WriteLine("a,b'ye eşittir");
        if(a < b) Console.WriteLine("a,b'denküçüktür");
        if(a > b) Console.WriteLine("a,b'denbüyüktür");
        if(a != b) Console.WriteLine("a,b'ye eşit değil");
        if(a >= b) Console.WriteLine("a,b'ye eşittir veya büyüktür");
        if(a <= b) Console.WriteLine("a,b'ye eşittir veya küçüktür");
    }
}
```

Operatörler

- **as operatörü** sık kullanılmayan bir operatördür. Uygun türler arası dönüşüm sağlar as operatörü sonucunda referans türde değer elde edilir.

- Genellikle object türündeki nesneleri farklı bir türe çevirmek için kullanılır. Çevirme işlemi başarısız ise null değeri üretir. Kullanımı:

- <referans tipi üretilecek ifade> as <referans tipi>

  Örn:

- ```
  object i="50";
  ```
- ```
  string s=i as string;
  ```

Operatörler

- **is operatörü,** çalışma zamanında bir nesnenin türünün operand ile verilen türe uyumlu olup olmadığını kontrol eder. Sık kullanılmaz**.**

- Kullanımı:
- <ifade>is<tür>

- Örn:
- `int i=50;`
- `bool b1=i is int`

Çalışma sonucu “true” değerini verir.

Mantıksal Operatörler

&	VE	&&	Kısa devre VE
|	VEYA	| |	Kısa devre VEYA
^	XOR	!	Değil

- **&& (VE) Operatörü:**
- "true" ya da "false" değerindeki operandları mantıksal VE ile işler. Operandlardan biri "false" ise "false" değeri üretir.
- **Kullanımı:** <ifade> && <ifade>
- Mantıksal VE işleminin doğruluk tablosu

1.Operand	**2.Operand**	**Sonuç**
True	True	True
True	False	False
False	True	False
False	False	False

- bool b1 = 25 < 15 && 5 == 50 **;**
- **false**
- bool b2 = 25 > 15 && 5 != 50 ;
- **true**

Mantıksal Operatörler

&	VE	&&	Kısa devre VE
|	VEYA	| |	Kısa devre VEYA
^	XOR	!	Değil

- **|| (VEYA) Operatörü:**
- "true" ya da "false" değerindeki operandl arı mantıksal VEYA ile işler. Operandlardan biri "true" ise "true" değeri üretir.
  Kullanımı: <ifade> || <ifade>
- Mantıksal VEYA işleminin doğruluk tablosu:

1.Operand	**2.Operand**	**Sonuç**
True	True	True
True	False	True
False	True	True
False	False	False

- bool b1 = 25 < 15 || 5 == 50 ;
- **false**
- bool b2 = 25 > 15 || 5 != 50;
- **true**

Mantıksal Operatörler

&	VE	&&	Kısa devre VE
|	VEYA	| |	Kısa devre VEYA
^	XOR	!	Değil

- **! (Değil) Operatörü:**
- Tek operand alır ve mantıksal Değil (NOT) işlemini uygular.
  Kullanımı: **! <ifade>**
- Mantıksal DEĞİL işleminin doğruluk tablosu
- **Operand** **Sonuç**
- True False
- False True
  - bool b1 = ! (25 < 10) ;
  - **True**
  - bool b2 = ! (10 != 20) ;
  - **false**

Mantıksal Operatörler

```
using System;
class mantıksal_operatörler
{
    public static void Main()
    {
        bool a = true;
        bool b = false;
        if (a ^ b) Console.WriteLine("a^b doğrudur");
        if (a && b) Console.WriteLine("a ve b doğrudur");
        if (a || b) Console.WriteLine("a veya b doğrudur");
    }
}
```

Bit Tabanlı Operatörler

&	Bit tabanlı VE	>>	Sağa kaydırma
|	Bit tabanlı VEYA	<<	Sola kaydırma
^	Bit tabanlı XOR	~	Birin tümleyeni

- Bitsel operatörler ise sayıları ikilik olarak ele alır ve her bitlerinde işlem gerçekleştirirler.
- Bitsel operatörler tamsayılarda kullanılır, gerçek sayılarda kullanılamaz.
- Bitsel operatörlerde **bool** türünden ifadeler kullanılırsa mantıksal işlemler gerçekleştirir.
- Bu operatörler bitlerin durumlarını öğrenmek veya belirli bitlerin değerlerini değiştirmek için kullanılabilir.
- Örn: bool a= false && false ile bool a= false & false aynıdır.

Bit Tabanlı Operatörler

- **~ (Bitsel DEĞİL) Operatörü:**
- Tek operand alan **~** operatörü bir değer içindeki bitlerin teker teker tersini alır.
- **0000 1111** sayısının bitsel değili **1111 0000** olur.

```
using System;
class Operatorler
{
    static void Main()
    {
        byte b1 = 254;
        byte b2 = (byte)~b1;
        Console.WriteLine(b2);
    }
}
```

Bit Tabanlı Operatörler

- **& (Bitsel VE) Operatörü:**

- İki operandın bitlerini karşılıklı olarak VE işlemine tabi tutar.
- byte a = 153; **153 :1001 1001**
- byte b = 104; **104 :0110 1000**
- byte x = (byte)(a & b) ----------
- ->x=8 **VE 0000 1000**
-

Bit Tabanlı Operatörler

- **| (Bitsel VEYA) Operatörü:**

- İki operandın bitlerini karşılıklı olarak VEYA işlemine tabi tutar.
- byte a = 153; **153 :1001 1001**
- byte b = 104; **104 :0110 1000**
- byte x = (byte)(a | b) ----------
- ->x=249 **VE YA 1111 1001**
-

Bit Tabanlı Operatörler

- **^ (Bitsel Özel VEYA) Operatörü:**

- İki operandın bitlerini karşılıklı olarak Özel VEYA (XOR) işlemine tabi tutar.
- byte a = 153; **153 :1001 1001**
- byte b = 104; **104 :0110 1000**
- byte x = (byte)(a ^ b) ----------
- ->x=241 **XOR 1111 0001**
-

Bit Tabanlı Operatörler

- **^ (Bitsel Özel VEYA) Operatörü:**

```
using System;
class Operatorler
{
    static void Main()
    {
        int x = 195;
        int y = 26;
        x = x ^ y;
        y = y ^ x;
        x = x ^ y;
        Console.WriteLine("x={0}\ny={1}", x, y);
    }
}
```

```
x=26
y=195
```

Bit Tabanlı Operatörler

- **<<(Bitsel Sola Kaydırma) Operatörü:**
- Verilen bir tamsayının bitlerinin istenilen sayıda sola doğru ötelenmesini sağlar.
- İki operandı bulunur. İşlem yapılacak değer ve kaydırılacak bit sayısı. İşlem yapılırken en düşük değerlikli bit sıfır ile tamamlanırken en yüksek değerlikli bitten taşanlar atılır. Kullanımı:
- ifade << öteleme sayısı
- byte b = 0xFF // 1111 1111 (255)
- byte c = (byte)(b<<4) // 1111 0000

Bit Tabanlı Operatörler

- **>> (Bitsel Sağa Kaydırma) Operatörü:**
- Verilen bir tamsayının bitlerinin istenilen sayıda sağa doğru ötelenmesini sağlar. İki operandı bulunur. İşlem yapılacak değer ve kaydırılacak bit sayısı. İşlem yapılırken en yüksek değerlikli bit sıfır ile tamamlanırken en düşük değerlikli bitten taşanlar atılır. Kullanımı:
- ifade >> öteleme sayısı
- byte b = 0xFF // 1111 1111
- byte c = (byte)(b>>4) // 0000 1111

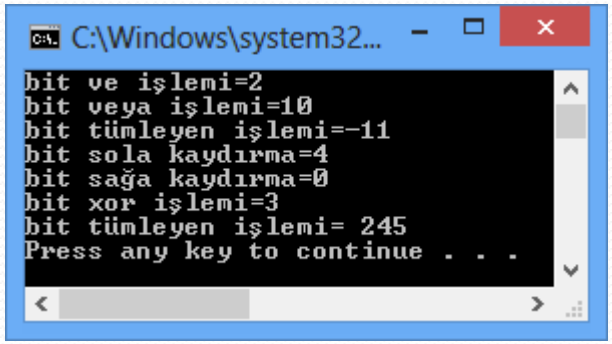
Bit Tabanlı Operatörler

- Örnek

```
using System;
class Operatorler
{
    static void Main()
    {
        byte b = 0xFF; //255
        b = (byte) (b << 4);
        Console.WriteLine(b); //240
        b = (byte)(b >> 3);
        Console.WriteLine(b); //30
    }
}
```

Bit Tabanlı Operatörler

```
using System;
class mantıksal_operatörler
{
    public static void Main()
    {
        int a = 10; int b = 2;
        Console.WriteLine("bit ve işlemi=" + (int)(a & b));
        Console.WriteLine("bit veya işlemi=" + (int)(a | b));
        Console.WriteLine("bit tümleyen işlemi=" + (int)(~a));
        Console.WriteLine("bit sola kaydırma=" + (int)(1 << b));
        Console.WriteLine("bit sağa kaydırma=" + (int)(1 >> b));
        Console.WriteLine("bit xor işlemi=" + (int)(1 ^ b));
        Console.WriteLine("bit tümleyen işlemi= "+(byte)(~a));
    }
}
```



Atama ve İşlemli Atama Opreratörleri

- **= (Atama) Operatörü:**
- Bir değişkene herhangi bir değer atamak için kullanılır. Çok bilinen ve kullanılan bir operatördür.
- İşlemli atama operatörleri, işlem sonucunun işlenen operanda atandığı durumlarda kullanılır

- x = x + y; ifadesi yerine x+=y;
- a = a * b; ifadesi yerine a *= b;
- b = b << n; ifadesi yerine b<<=n;

Özel Amaçlı Operatörler

- **? : (Ternarty) Operatörü:**
- Bir koşulu karşılaştırıp doğru ya da yanlış olduğu durumlar için farklı değerler üretmeyi sağlayan operatördür. Üç tane operanda sahiptir. Kullanımı:

- **koşul ? doğru değeri : yanlış değeri**

- sayi < 10 ? “10dan küçük” : “10 a eşit ya da büyük”

Özel Amaçlı Operatörler

- **() Operatörü (Tür Dönüştürme):**
  - İfade önüne yazıldığında tür dönüştürme operatörü olarak çalışır
- **[] Operatörü (İndeks):**
  - Dizilerde elemanın indeksini belirtmek için kullanılır
- **+, - (İşaret Operatörleri):**
  - Bir değişkenin içeriğindeki değerin negatif ya da pozitif olarak işlem görmesini sağlar. + aynı zamanda stringler de ekleme için de kullanılan bir operatördür.
- **&, *, -> ve sizeof Operatörleri:**
  - C/C++ dilindeki işaretçileri (pointer) C# dilinde de kullanmak mümkündür. Güvensiz (unsafe) kod yazarken bu operatörler de kullanılabilir.

Özel Amaçlı Operatörler

- **. Operatörü:**
  - Bir sınıfın ya da yapının elemanlarına ulaşmak için kullanılır
  - Console.WriteLine(); gibi…

- **new Operatörü:**
  - Yeni bir nesne oluşturmak için kullanılan operatördür.
  - Değer ve referans tiplerden yeni bir nesne oluşturmak için kullanılabilir
  - Örnek : Sinif s = new Sinif();

Özel Amaçlı Operatörler

- **checked ve unchecked Operatörleri**
  - Büyük türün küçük türe dönüşümde taşma meydana geldiğinde uyarı veren (checked) veya taşmayı göz ardı eden (unchecked) oparatörlerdir.

- **typeof Operatörü:**
  - Çalışma zamanında bir tür için System.Type sınıfı türünden bir nesne üretilmesini sağlar. Nesneler için kullanılan GetType() ifadesinin benzer işlemini tür ve ya sınıf tanımlamaları için gerçekleştirir

Özel Amaçlı Operatörler

- **Örnek**

```
using System;

class Operatorler13
{
    static void Main()
    {
        Type t;
        int i = 123;
        float f = 23.45F;

        t = typeof(int);
        Console.Write(t.ToString() + "   ");
        Console.WriteLine(i.GetType());

        t = typeof(float);
        Console.Write(t.ToString() + "   ");
        Console.WriteLine(f.GetType());
    }
}
```

Akış Kontrol Mekanizmaları

- ***Koşul ifadeleri***
  - ***if deyimi***
  - ***switch deyimi***
- ***Döngü yapıları (loops)***
  - ***for döngüsü***
  - ***while döngüsü***
  - ***foreach döngüsü***
- ***Atlama(jump)***
  - ***break, continue, goto, return***

Koşul İfadeleri

- Programlar içersinde bazı durumlarda programın akışının değişmesi ya da farklı işlemlerin yapılması gerekebilir. Bazen de seçimlerin yapılması ya da belirli işlemlerin belli sayıda tekrar edilmesi gerekebilir.

- Bu gibi işlemleri gerçekleştirmek için koşul ifadeleri ve döngü yapıları kullanılır.

- C# dilinde koşula bağlı olarak işlemler gerçekleştirmek için iki farklı deyim kullanılabilir:
  - **if deyimi**
  - **switch deyimi**

Koşul İfadeleri

- Programlar içersinde bazı durumlarda programın akışının değişmesi ya da farklı işlemlerin yapılması gerekebilir. Bazen de seçimlerin yapılması ya da belirli işlemlerin belli sayıda tekrar edilmesi gerekebilir.

- Bu gibi işlemleri gerçekleştirmek için koşul ifadeleri ve döngü yapıları kullanılır.

- C# dilinde koşula bağlı olarak işlemler gerçekleştirmek için iki farklı deyim kullanılabilir:
  - **if deyimi**
  - **switch deyimi**

if Deyimi

- Program akış kontrol deyimlerinin başında gelir. Koşula göre değişik işlemlerin yapılmasını sağlar. Basit kullanımı:

- ***if (koşul)***
- ***Deyim1;***
- ***else***
- ***Deyim2;***
- **ya da**
- ***if (koşul)***
- ***Deyim1;***
- ***else if***
- ***Deyim2;***
- ***else***
- ***Deyim3;***

- ***Eğer deyim kısmında 1 den fazla satır varsa { } program bloğu kullanmak zorunludur.***

if Deyimi

```
using System;
class if_yapısı
{
    public static void Main()
    {
        string s;
        int a, b, c;
        Console.Write("1.sayıyı giriniz : ");
        s = Console.ReadLine();
        a = Int32.Parse(s);
        Console.Write("2.sayıyı giriniz : ");
        s = Console.ReadLine();
        b = Int32.Parse(s);
        Console.Write("3.sayıyı giriniz : ");
        s = Console.ReadLine();
        c = Int32.Parse(s);
        if (a >= b && a >= c) Console.WriteLine("Enbüyük :" + a);
        else if (b >= c) Console.WriteLine("Enbüyük :" + b);
        else Console.WriteLine("Enbüyük :" + c);
    }
}
```

switch Deyimi

- Bir ifadenin alabileceği değişik değerlere göre işlemler gerçekleştirmek için kullanılan deyimdir.
- Karmaşık if ifadelerini daha sade bir şekilde ifade etmeyi de sağlayabilir.

Kullanımı:

```
switch(ifade)
{
case sabit1:
        deyim1;
        break;
case sabit2:
        deyim2;
        break;
default:
        deyim3;
        break;
}
```

switch Deyimi

- Her case ifadesinin bir atlama **(break, goto, return, continue)** ifadesi ile sonlandırmalısınız.
- **continue** deyimi switch ifadesi bir döngünün gövdesinde ise kullanılabilir.
- Bir case içinde hiçbir ifade verilmezse atlama deyimi kullanılması gerekmez
  - case 2: // ifade yok
  - case 3:
  - case 4: ifade….;break,;
  - case 5: ifade…; goto case 4;

Switch Deyimi

- **switch ifadesi kullanırken dikkat edilecek bazı kurallar vardır:**
  - case sözcüğünden sonra gelen ifadeler sabit olmak zorundadır.
  - **case ifadeleri tamsayı, karakter ya da string sabitler olabilir.**
  - default ve case ifadeleri istenilen sırada yazılabilir.
  - Aynı switch bloğu içerisinde birden fazla aynı case ifadesi bulunamaz.
  - default kullanmak zorunlu değildir.
  - Akışı bir case ifadesinden bir başka case ifadesine yönlendirmek için mutlaka goto anahtar sözcüğü kullanılır.

Switch Deyimi

```
using System;
class switch_case_yapısı
{
    public static void Main()
    {
        string s;        int a, b, c;
        Console.Write("Kaçıncı Sınıftasınız : ");
        s = Console.ReadLine();
        a = Int32.Parse(s);
        switch (a)
        {   case (0):
            case (1): Console.WriteLine("Daha Yenisiniz");  break;
            case (2): Console.WriteLine("Mirasçısınız");    break;
            case (3): Console.WriteLine("Ev Sahibisiniz");  break;
            case (4): Console.WriteLine("Misafirsiniz");    break;
            default: Console.WriteLine("Siz okulu uzatmışsınız"); goto case 1;
        }
    }
}
```

Döngüler(Loops)

- Program içerisinde belirli işleri tekrar tekrar yapılmasını sağlayan kod bloklarıdır. Döngüler sonsuz olabileceği gibi belli bir koşul ile kontrol etmek de mümkündür.
- C# dilinde dört tip döngü yapısı bulunur:
  - for
  - while
  - do while
  - foreach

For Döngüsü

- for döngü tanımlamasında iki “;” ile ayrılmış üç ifade bulunur. Bunlardan bazıları boş olabilir fakat mutlaka “;” işaretleri kullanılmalıdır.

- İlk ifade bir defaya mahsus olmak üzere çalıştırılır. Genelde döngü değişkeninin tanımlanması ya da ilk değer ataması için kullanılır.

- İkinci ifade ise döngünün kontrol edildiği kısımdır Buradaki ifade “true” değer ürettiği sürece döngü devam eder.

- Son ifade ise genelde döngü değişkeninin değerinin değiştirildiği kısımdır.

For Döngüsü

- ***for(ifade1;ifade2;ifade3)***
- ***{***
- ***deyimler;***
- ***}***
- Kullanımı:

- *for(;;) Sonsuz döngüdür.*
- *for(int i=1;i<=100;i++);*
- *for(int i=1,j=10;(i<=10)&(j>=1);i++,j--);*

For Döngüsü

```
using System;

class for_1
{
    static void Main()
    {
        Console.Write("Bir sayı girin:");
        int n = Convert.ToInt32(Console.ReadLine());
        int toplam = 0;

        for (int i = 1; i <= n; i++)
            toplam = toplam + i;

        Console.WriteLine("1\'den {0}\'e kadar olan
            sayıların toplamı {1}\'dir.", n, toplam);
    }
}
```

While ve do-while Döngüsü

- ***while (koşul)***
- ***{***
- ***deyimler***
- ***}***
- Belirtilen koşul doğru olduğu sürece çalışmasına devam eder.

- ***do while***
- ***{***
- ***deyimler***
- ***} while(koşul);***

- for ve while döngülerinde koşul sağlanmadığı durumlarda döngü bloğu hiç çalıştırılmaz. Bazı durumlarda ise döngünün en az bir kez çalıştırılması gerekebilir. Bu durumda do while kullanılabilir.

While ve do-while Döngüsü

```
using System;

class while_1
{
    static void Main()
    {
        int i=0;

        while (i < 1000)
        {
            i += 5;
            Console.Write("{0,5}",i);
            if (i % 50 == 0) Console.WriteLine();
        }
    }
}
```

Foreach Döngüsü

- Farklı bir döngü yapısıdır. Koleksiyon tabanlı nesneler içerisinde adım adım dolaşılmasını sağlar.

- Bu döngü elemanlara tek tek ulaşım sağlar fakat ulaşılan elemanlar sadece okunabilir özelliğe sahiptir.

Foreach Döngüsü

- string[ ] isimler ={ "Ozlem","Nesrin", "Ozge", "Fulya" };

//string dizi tanımı

```
foreach (string ss in isimler)  // elemanlar yazdırılıyor
 {
  Console.WriteLine(ss);
 }
```

- foreach (int i in dizi) // i sadece okunur(readonly)

```
 {  Console.WriteLine(i);
    /* i++;*/ //yasak kullanılamaz
 }
```

Jump (atlama) Deyimleri)

- **Break*:*** Çalışan bir döngüden “break” sözcüğü kullanılarak çıkılabilir. Program akışı döngüden sonraki satırlardan devam eder. break sözcüğü sadece döngü ve switch ifadelerinde kullanılabilir.

- **Continue*:*** Döngünün bir sonraki tekrarına geçilmesini sağlar.

- **Goto*:*** Programın etiket ile belirlenmiş herhangi bir kısmına atlamak için kullanılır. Nesneye yönelik programlamaya uygun bir yapı değildir. Switch ifadesindeki kullanımı dışında mümkün olduğunca kullanılmaktan kaçınılmalıdır.

- **Return *:*** Metotların herhangi bir anda sonlandırılması için kullanılır. Metot sonlandırıldıktan sonra programın akışı metodu çağıran fonksiyondan devam eder.

Jump (atlama) Deyimleri)

```
using System;

class break_continue_1
{
    static void Main()
    {
        int sayi, t=0;

        while (true)
        {
            Console.Write("Negatif bir sayı girin:");
            sayi = Convert.ToInt32(Console.ReadLine());
            if (sayi == 0) break;
            if (sayi > 0)
            {
                Console.WriteLine("Pozitif sayı girdiniz,
   tekrar deneyin...");
                continue;
            }
            t = t + sayi;
        }
        Console.WriteLine("Toplam :{0}", t);
    }
}
```

System.Random Sınıfı

- Rastgele sayı üretmek için .NET Framework sınıf kütüphanesindeki **System.Random** sınıfı kullanılır. Rastgele sayı üretmek için öncelikle **Random** sınıfı türünden bir nesne oluşturmak gerekir.

- ***Random rnd=new Random ( );//tanımlama***

- rnd nesnesi oluşturulduktan sonra artık bu nesnenin metotlarına erişebiliriz.

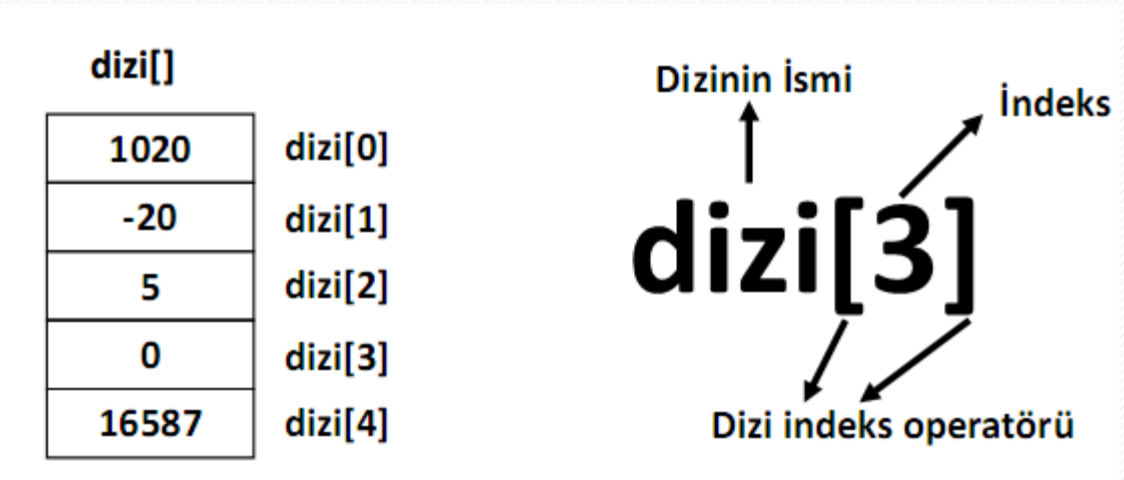
System.Random Sınıfı

- *Int rastgelesayı =rnd.next(10,20);*
  - *10 ile 20 arasında bir sayı üretir.*
- *Int rastgelesayı =rnd.next(50);*
  - *0 ile 50 arasında bir sayı üretir.*
- *Int rastgelesayı =rnd.next();*
  - *Herhangi bir rasgele sayı üretir.*
- *double rastgelesayı =rnd.NextDouble();*
  - *0.0 ile 1 arasında bir sayı üretir.*

Diziler (Arrays)

- Bellekte ard arda yer alan aynı türden nesneler kümesine dizi (array) denilir. Bir dizi içerisindeki bütün elemanlara aynı isimle ulaşılır. Yani dizideki bütün elemanların isimleri ortaktır. Elemanlar arasındaki ayırt edici özellik, bellekteki yeridir.

- 5 elemanlı bir tamsayı dizisi bellekte aşağıdaki gibi yerleştirilir.



Diziler (Arrays)

- C#'da diziler ayrı bir tür olarak tasarlanmıştır. Bütün diziler System.Array sınıfından türetilmiştir. Bir dizi tanımlandığında Array sınıfı türünden bir nesne tanımlanmış olur.

- Dizinin sınırlarını aşan bir indeks kullandığımızda derleme hatası oluşmaz, ancak çalışma zamanında hata oluşur.
- Dizi tanımlama 2 türlü yapılabilir :

```
int[] dizi = new int[19];
```

- veya

```
int[] dizi;
dizi = new int[19];
```

Diziler (Arrays)

- new anahtar sözcüğü ile, dizinin her elemanına temel veri türleri için varsayılan değer, ilk değer olarak verilmektedir.
- Bu ilk değer; referans türleri için null, nümerik türler için 0, bool türü için ise false'tur.

- Dizileri aşağıdaki gibi de tanımlamak mümkündür:

```
string[] dizi1={"Bilgisayar","Mühendisliği","Bölümü"};
int [] dizi2= {5,9,12,56,23};
float [] dizi3={8f,39f,324f,23f,2f};
```

Diziler (Arrays)

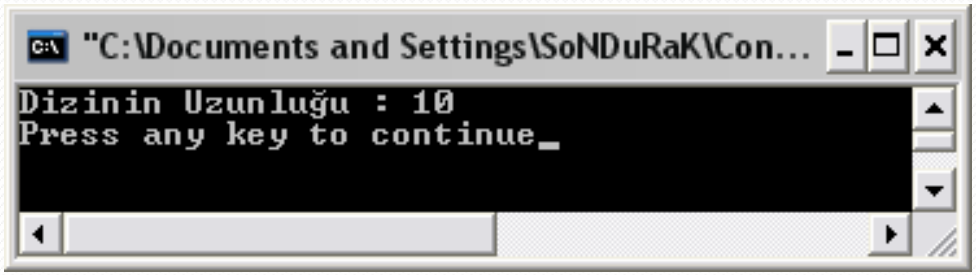
- Dizilerin boyutu C ve C++ dillerinde derleme sırasında bilinmek zorundadır. Böylece derleyici, dizi için bellekte dizinin boyutuna göre yer ayırmaktadır. Ancak C# 'ta ise diziler referans tipi olduğu için dizi boyutları çalışma zamanında belirlenebilir.

```
class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Dizinin boyutunu giriniz=");
            int boyut = Convert.ToInt32(Console.ReadLine());
            bool [] dizi = new bool [boyut];
            for (int i = 0; i < boyut; i++)
                Console.WriteLine(dizi[i]);
            Console.ReadLine();
        }
    }
```

- Bir dizinin boyutu bir kez belirlendikten sonra artık değiştirilemez. Yani dizinin boyutunu dinamik olarak değiştirmemiz mümkün değildir.

Diziler (Arrays)

```
using System;
class Length
{
    static void Main()
    {
        int k = 0;
        int[] dizi = new int[10];
        //diziye ait nesne oluşturuluyor
        k = dizi.Length;
        Console.WriteLine("Dizinin Uzunluğu : " + k);
    }
}
```



Diziler (Arrays)

- *Diziye rastgele değer ataması;Random sınıfı, System içinde tanımlı.*

```
Random rnd = new Random();
    int[] dizi =new int[3];
    dizi[0] = rnd.Next(2,10); // 2 ile 10 arası rastgele sayı,2 dahil.
    dizi[1] = rnd.Next(50);   //0 ile 50 arası
    dizi[2] = rnd.Next();  //rastgele integer değer

foreach(int i in dizi)
{
    Console.WriteLine(i);
}
```

```
for (k=0;k<3;k++) //0 10 arası değer
 {
dizi[k] = rnd.Next(10);
Console.WriteLine(dizi[k]);
 }
```

Çok Boyutlu Diziler (Multidimensional *Arrays)*

- İki türlü çok boyutlu dizi bulunur. Her boyutta eşit miktarda elemanın olduğu düzenli diziler (matrisler) ve her boyutta farklı sayıda eleman barındıran düzensiz diziler(jagged arrays). Aslında düzensiz diziler elemanları da dizi olan dizilerdir.

- Çok boyutlu dizi tanımı için tanımlama esnasında [ ] içine verilecek boyut kadar ‘,’ değeri eklenir. “[, ,]” şeklinde yazılır.

Matris Dizileri

- **int [,] dizi = {{1,2},{3,4},{5,6}};**
- Şeklinde tanımlanan bir 3x2'lik dizinin elemanları:

dizi[0,0]=1
dizi[0,1]=2
dizi[1,0]=3
dizi[1,1]=4
dizi[2,0]=5
dizi[2,1]=6

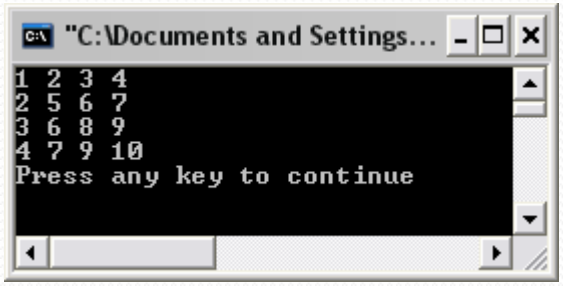
dizi

[0,0]	[0,1]
[1,0]	[1,1]
[2,0]	[2,1]

- int[,] dizi1=new int[2,3];
  dizi1[0,0]=1;
  dizi1[2,1]=7;

Çok Boyutlu Diziler

```
using System;
class diziler
{
  public static void Main()
  {    int[,] dizi = new int[4, 4]; int k=1;
       for (int i=0;i<4;i++)
          for(int j=i;j<4;j++) { dizi[i,j]=k;  dizi[j,i]=k;  k++;  }
       k=1;
       foreach(int x in dizi)
       {
          if (k%4==0)
          {  Console.Write(x+" ");  Console.WriteLine();    }
          else Console.Write(x+" "); k++;
       }
   }
}
```



Düzensiz Diziler(Jagged Arrays)

- Dizilerin elemanları da farklı diziler olabilir. Bu durumda tanımlanan dizinin birbirinden farklı boyutlarda alt elemanları olabilir. Bu gibi yapılara düzensiz diziler ismi verilir.

dizi[0][0]	dizi[0][1]	dizi[0][2]	dizi[0][3]	dizi[0][4]
dizi[1][0]	dizi[1][1]	dizi[1][2]		
dizi[2][0]	dizi[2][1]	dizi[2][2]	dizi[2][3]	

- Düzensiz dizilerde boyut çift [][] yapısıyla veriliyor. İlk [] satır, ikinci [] ifade düzensiz yapan sütuna aittir.

```
int [][] dizi= new int[3][];
dizi[0]=new int[5];
dizi[1]=new int[3];
dizi[2]=new int[4];
```

Düzensiz Diziler(Jagged Arrays)

- Düzensiz dizilerde eleman sayısı sabit olmadığı için dizi elemanlarını taramak için Array sınıfının metotları ve ya özellikleri kullanılır. **Length** özelliği bir dizinin eleman sayısını verir.

```
int [][] dizi= new int[3][];
dizi[0]=new int[5];
dizi[1]=new int[3];
dizi[2]=new int[4];
Console.WriteLine(dizi.GetLength(0)); // dizinin satır sayısı
Console.WriteLine(dizi[2].GetLength(0));//2.satırdaki sütun
                                        sayısı
```

Düzensiz Diziler(Jagged Arrays)

```
using System;

class dizi_5
{
    static void Main()
    {
        int[][] dizi = new int[3][];

        dizi[0] = new int[] { 1, 2 };
        dizi[1] = new int[] { 3, 4, 5, 6, 7 };
        dizi[2] = new int[] { 8, 9, 0 };

        for (int i = 0; i < dizi.GetLength(0); i++)
         for (int j = 0; j < dizi[i].GetLength(0); j++)
         Console.WriteLine("dizi[{0}][{1}]={2}",i,j,dizi[i][j]);
    }
}
```

System.Array Sınıfı

- Dizi nesneleri **System.Array** sınıfından türemişlerdir. Dolayısıyla bu sınıfın bazı özellik ve metotlarını bünyelerinde barındırırlar.
- **Özellikler:**

IsFixedSize	Dizinin eleman sayısının sabit olup olmadığını verir. (Boolean)
IsReadOnly	Dizideki elemanların sadece okunabilir olup olmadığını verir. (Boolean)
Length	Dizideki eleman sayısını verir.
Rank	Dizinin boyutunu verir.

System.Array Sınıfı

- **Metotlar:**

BinarSearch	Tek boyutlu dizide binary search algoritmasına göre arama yapar.
Clear	Dizinin elemanlarını varsayılan değere çeker.
Clone	Dizinin bit bit kopyasını çıkarır.
Copy	Dizinin bir bölümünü başka bir diziye kopyalar.
CopyTo	Bir dizinin belirlenen bir kısmını başka bir diziye kopyalar.
GetLength	Dizideki eleman sayısını verir.
GetValue	Dizideki ilgili eleman değerini verir.
IndexOf	Dizi içindeki bir değerin ilk görüldüğü indeksi verir.
Reverse	Diziyi tersine çevirir.
SetValue	Bir dizinin bir elemanına değer atar.
Sort	Bir boyutlu dizilerde sıralama yapar.
CreateInstance	Yeni bir dizi nesnesi oluşturur.

System.Array Sınıfı

- int[ ] dizi ={ 1, 22, 34 };

```
Console.WriteLine(dizi.GetValue(2)); // 34 değerini verir
Console.WriteLine(dizi[2]); // 34 değerini verir
```

- int v = dizi.GetLength(0);//satır
- int y = dizi.GetLength(1);//sütun

- **SetValue**: Metodu ilgili dizinin belirtilen indekslerindeki elemanın değerini değiştirmek için kullanılır.

- dizi.SetValue(20,0);//(değer,1.index)
- dizi.SetValue(20,0,0);//(değer,1.index,2.index)

- **GetValue**: Metodu belirtilen indekslerdeki elemanın değerini verir.

CreateInstance( )

- CreateInstance metodu ile aşağıdaki gibi dizi nesnesi oluşturulabilir:
  - Array dizi=Array.CreateInstance(typeof(int),5);
  - 5 elemanlı int türünden bir dizi tanımlanır.
- İlk parametre her zaman Type türünden olmalıdır. Yukarıdaki gibi bir kullanımda, typeof() operatörü yardımıyla int türünün type sınıfındaki karşılığı elde edilmiş olmaktadır. Örn:
  - CreateInstance(Type,int,int,int);
  - 3 boyutlu bir dizi oluşturulmakta.
  - CreateInstance(Type,int[]);
    Çok boyutlu diziler oluşturmak için kullanılır. Çok boyutlu dizilerin her birinin kaç elemanlı olduğunu belirlemek için int türden bir dizi metoda parametre olarak gönderilir.
  - Array dizi=Array.CreateInstance(typeof(int),3,5);
  - int tipinde 2 boyutlu bir dizi tanımlanır.

CreateInstance( )

```
using System;

class dizi_6
{
    static void Main()
    {
        Array dizi = Array.CreateInstance(typeof(int), 5, 4, 3);
        Random r = new Random();

        for (int i=0; i< dizi.GetLength(0); i++)
            for (int j=0; j< dizi.GetLength(1); j++)
                for (int k = 0; k < dizi.GetLength(2); k++)
                {
                    dizi.SetValue(r.Next(10, 100), i, j, k);
                    Console.WriteLine("dizi[{0},{1},{2}]={3,3}",
                                      i,j,k,dizi.GetValue(i,j,k));
                }
    }
}
```

CopyTo ( )

- CopyTo metodu ile bir dizinin tamamı, başka bir dizinin istenilen yerine kopyalanabilir. Örn:

- **int[ ] dizi1= {1,2,3,4,*5,6,7*};**
- **int dizi2=new int[10];**
- **dizi1. CopyTo(dizi2,3);**

- dizi1'in elemanlarını dizi2'nin 3. indisinden itibaren kopyalanır.

Copy ()

- Aynı işlem Copy() metodu kullanılarak da gerçekleştirilebilir.
- Copy (Array dizi1,Array dizi2,int uzunluk);

- Bu şekildeki bir kullanım, uzunluk kadar elemanı dizi1'den dizi2'ye kopyalar. Kopyalama işlemi 0. indekten başlar.

  - Copy (Array dizi1, int x, Array dizi2, int y, int uzunluk);
  - dizi1'in x. İndeksinden sonraki uzunluk kadar elemanı dizi2'nin y. İndeksinden sonrasına kopyalar.
  - *Array.Copy(dizi1,2,dizi2,3,1); // dizi1'in 2. indisinden itibaren, 1 elemanı, dizi2'nin 3. indis elemanından itibaren kopyalanır.*

Örnek

```
using System;

class dizi_7
{
    static void Main()
    {
        int[] dizi1 = { 1, 2, 3, 4, 5, 6, 7 };
        int[] dizi2 = new int[10];
        int[] dizi3 = new int[10];

        dizi1.CopyTo(dizi2, 2);
        foreach (int i in dizi2)
            Console.Write(i);

        Console.WriteLine();

        Array.Copy(dizi1, 2, dizi3, 5, 3);

        foreach (int i in dizi3)
            Console.Write(i);
    }
}
```

Array.Sort ( )

- Dizileri küçükten büyüğe sıralamaya yarar. Sort() metodunun birçok kullanım biçimi vardır. Temelde aşağıdaki şekilde kullanımı yaygındır. Örn:
- Array.Sort(Array);

  - *Array Dizi=Array.CreateInstance(typeof(string),3);*
  - *Dizi.SetValue("Ali",0);*
  - *Dizi.SetValue("Veli",1);*
  - *Dizi.SetValue("Sami",2);*
  - *Array.Sort(Dizi);*
  - *foreach(string s in array) Console.WriteLine(s);*

Array.Sort ( )

```
using System;

class dizi_8
{
    static void Main()
    {
        string[] dizi = { "Zeynep", "Fatma", "Ali", "Yılmaz",
   "Gökhan", "Osman", "Feride" };

        Console.WriteLine("Dizinin Elemanları\n==================");
        foreach (string s in dizi)
            Console.WriteLine(s);

        Array.Sort(dizi);
        Console.WriteLine();

        Console.WriteLine("Sıralanmış Dizi\n==============");
        foreach (string s in dizi)
            Console.WriteLine(s);

    }
}
```

Array.BinarySearch

- Sıralı dizilerde arama işlemi yapar. **BinarySearch** metodunun iki önemli kullanım biçimi vardır.

- **BinarySearch(Array dizi, object nesne);**
  - Bu yapı, dizi içerisinde nesneyi arar; eğer aranan nesne bulunursa bulunduğu index ile geri dönülür. Eğer bulunamazsa negatif bir sayıyla geri dönülür.

- **BinarySearch(Array dizi, int baslangic, int uzunluk, object nesne);**
  - Bu yapı ise, nesneyi başlangıç indeksinden itibaren uzunluk kadar eleman içerisinde arar.

Array.BinarySearch

```
using System;

class dizi_9
{
    static void Main()
    {
        string[] dizi = { "Zeynep", "Fatma", "Ali", "Yılmaz",
   "Gökhan", "Osman", "Feride" };

        Console.Write("Aranacak İsmi Giriniz :");
        string isim = Console.ReadLine();

        Array.Sort(dizi);
        int indeks = Array.BinarySearch(dizi, isim);

        if (indeks < 0)
            Console.WriteLine("Aranan isim dizide bulunamadı!");
        else
            Console.WriteLine("Aranan isim dizinin {0}. elemanında
   bulundu...",indeks);
    }
}
```

Diğer Metotlar

- **Clear()** metodu, belirtilen indis aralığındaki elemanları sıfırlar. Bir dizi elemanını , 0, Boşluk (Null) ya da false yapar.
  - Array.Clear(dizi,1,3);// 1. elemandan itibaren 3 elemanı sıfır yapar.
  - Array.Clear(dizi,0,2);// Sıfırıncı elemandan itibaren 2 elemanı sıfır yapar.
- **Reverse()** metodu, dizinin tümünü yada belirtilen indis aralığındaki elemanları ters çevirir.
  - Array.Reverse(dizi); veya
  - Array.Reverse (dizi,1,3);
    - Alttaki gibi bir kullanımda ise,1. elemandan itibaren ilk 3 elemanı ters çevirir.

Diğer Metotlar

- Diziler tıpkı değişkenler gibi birbirine atanabilirler. Örnek:
- **int**[] a={1,6,7,12};
- **int**[] b=a;
- Burada b dizisine a dizisi adres gösterme yoluyla aktarılmıştır, dolayısıyla da a dizisinin bir elemanında yapılan bir değişiklik b dizisini, b dizisinin bir elemanında yapılan bir değişiklik a dizisini etkileyecektir. Çünkü diziler birbirlerine adres gösterme yoluyla atanır. Ayrıca aşağıdaki gibi bir kullanım da doğrudur:
- **int**[] a={1,6,7,12};
- **int**[] b={1,5}; b=a;
- Burada b dizisi sahip olduğu elemanları kaybedip a dizisinin adresine yönlendi. Bu ve benzer şekilde diziler birbirlerine atanabilir. Yani aynı adrese yönlendirilebilirler.

Visual Studio.Net -C#
4. HAFTA
Metotlar ve Fonksiyonlar

ÖRNEK UYGULAMALAR

Örnek 1: Operatör

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int a = 2; int b = 3; int c = 6; int d = 1;
        /*  (a < b) = bu ifadenin doğru (true) olduğunu biliyoruz
            (c < d) = bu ifadenin yanlış (false) olduğunu biliyoruz   */
        Console.WriteLine(" (a<b)&&(c<d) --> " + ((a < b) && (c < d)));
        Console.WriteLine(" (a<b)||(c<d) --> " + ((a < b) || (c < d)));
        Console.WriteLine(" ! (a<b) -->" + (!(a < b)));
        Console.WriteLine(" (a<b)&(c<d) --> " + ((a < b) & (c < d)));
        Console.WriteLine(" (a<b)|(c<d) --> " + ((a < b) | (c < d)));
        Console.WriteLine(" (a<b)^(c<d) --> " + ((a < b) ^ (c < d)));
    }
}
```

Sonuç:

(a<b) && (c<d) --> False

(a<b) || (c<d) --> True

! (a<b) -->False

(a<b) & (c<d) --> False

(a<b) | (c<d) --> True

(a<b) ^ (c<d) --> True

Örnek 2: Operatör

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int a = 2; int b = 3; int c = 6;
        Console.WriteLine(" (a & b) --> " + (a & b));
        Console.WriteLine(" (a | b) --> " + (a | b));
        Console.WriteLine(" (a ^ b) --> " + (a ^ b));
        Console.WriteLine(" ( ~a ) --> " + (~a));
        Console.WriteLine(" ( ~b ) --> " + (~b));
        Console.WriteLine(" ( ~c ) --> " + (~c));
    }
}
```

Sonuç:

```
(a & b) --> 2
(a | b) -->  3
(a ^ b) --> 1
( ~a ) --> -3
( ~b ) --> -4
( ~c ) --> -7
```

Örnek 3: while döngü

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int i = 0; //döngü kontrol değişkeni
        while (i < 5)
        {
            Console.WriteLine("i = " + i);
            i++;
        }
    }
}
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
```

Örnek 4: do-while döngü

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        int i = 0; //döngü kontrol değişkeni
        do
        {
            Console.WriteLine("i = " + i);
            i++;
        } while (i < 0);
    }
}
```

- i = 0

Örnek 5 : do-while ve switch

```
using System;

class do_while_1
{
    static void Main()
    {
        int secim;

        do
        {
            Console.WriteLine("İşlemler\n========");
            Console.WriteLine("1 - Toplama");
            Console.WriteLine("2 - Çıkarma");
            Console.WriteLine("3 - Çarpma");
            Console.WriteLine("4 - Bölme");
            Console.WriteLine("0 - Çıkış\n");
            Console.Write("İşleminizi Seçin:");
            secim = Convert.ToInt32(Console.ReadLine());
            switch (secim)
            {
                case 1: Console.WriteLine("Toplama işlemi seçildi."); break;
                case 2: Console.WriteLine("Çıkarma işlemi seçildi."); break;
                case 3: Console.WriteLine("Çarpma işlemi seçildi."); break;
                case 4: Console.WriteLine("Bölme işlemi seçildi."); break;
                case 0: Console.WriteLine("Çıkış seçildi."); break;
                default: Console.WriteLine("Yanlış seçim."); break;
            }
        }
        while(secim != 0);
    }
}
```

Örnek 6: for döngü

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        for (int i = 0, j = 0; i < 20; i++, j++)
        {
            i *= j;
            Console.WriteLine("i = " + i + "  j = " + j);
        }
    }
}
```

Örnek 7: for döngü

```
using System;

class for_2
{
    static void Main()
    {
        string s;

        for (s = Console.ReadLine(); s != "Çıkış";
             s = Console.ReadLine())

        Console.WriteLine(s);
    }
}
```

Örnek 8: for döngü

```
using System;

class for_3
{
    static void Main()
    {
        int i=0, a, n;

        Console.Write("Bir Sayı Girin :");
        n = Convert.ToInt32(Console.ReadLine());

        Console.Write("Artım Miktarı  :");
        a = Convert.ToInt32(Console.ReadLine());

        for (; i < n; )
        {
            Console.Write("{0} ", i);
            i += a;
        }
    }
}
```

Örnek 9: for döngü

```
using System;

class for_4
{
    static void Main()
    {
        int k, t, toplam, n1, n2;

        Console.Write("Aralık başlangıcı :");
        n1 = Convert.ToInt32(Console.ReadLine());

        Console.Write("Aralık Sonu       :");
        n2 = Convert.ToInt32(Console.ReadLine());

        if (n1 <= n2)
        {
            for (k = n1; k <= n2; k++)
            {
                toplam = 0;
                for (t = 1; t <= k; t++) { if (k % t == 0) toplam = toplam + t; }
                if (toplam == k + 1) Console.WriteLine(k);
            }
        }
        else
        {
            Console.WriteLine("Geçerli bir aralık giriniz...");
        }
    }
}
```

Örnek 10: for döngü

```
using System;

class for_5
{
    static void Main()
    {
        int sayi;

        Console.Write("Bir tamsayı giriniz :");
        sayi = Convert.ToInt32(Console.ReadLine());

        for (int bit = 32; bit >= 1; bit--)
        {
            Console.Write("{0}", (sayi >> bit - 1) & 1);
        }
        Console.WriteLine();
    }
}
```

Örnek 11: Diziler

```
using System;

class dizi_3
{
    static void Main()
    {
        Random r = new Random();

        int[] dizi = new int[20];
        char[] chr = new char[20];

        for (int i = 0; i < 20; i++)
        {
            dizi[i] = r.Next(1, 51);
            chr[i] = (char)r.Next(20, 126);
        }
        for (int i = 0; i < 20; i++)
        {
            Console.Write("{0,2}. değer {1,2} -> ", i, dizi[i]);
            for (int j = 0; j < dizi[i]; j++)
                Console.Write(chr[i]);
            Console.WriteLine();
        }
    }
}
```

Örnek 12:Matris Dizileri

```
using System;

class dizi_4
{
    static void Main()
    {
        int[,] dizi = { { 1, 2 }, { 3, 4 }, { 5, 6 } };

        for (int i=0; i<3; i++)
            for (int j =0; j < 2; j++)
            {
                Console.WriteLine(dizi[i, j]);
            }
    }
}
```

Örnek 13 : Düzensiz Diziler

```
using System;

class dizi_6
{
    static void Main()
    {
        int[][] dizi = new int [3][];

        dizi[0] = new int[] {1,2};
        dizi[1] = new int[] {3,4,5,6,7};
        dizi[2] = new int[] {8,9,0};

        foreach (int[] boyut in dizi)
        {
            foreach (int eleman in boyut)
            {
                Console.Write("{0,3}", eleman);
            }
            Console.WriteLine();
        }
    }
}
```

Örnek 14: diziler

```
int[,] d = {{2,3},{4,5},{6,7}}; // 3x2 lik int dizi.
d[2,1]=33; //doğru
d[2][1]=33; //hata
Console.WriteLine(d.GetValue(2,1)); //  7 değerini yazacak
int[][] dd = new int[2][];  // şekildeki düzensiz dizi oluşturuluyor.
dd[0] = new int[2];  //ilk satırın 2 sutunlu olacağı
dd[1] = new int[1];  //ikinci satırın 1 sütunlu olacağı
dd[0][0] = 23;
dd[0][1] = 234;
dd[1][0] = 44;
Random rnd = new Random();
int[] dizi =new int[3];
dizi[0] = rnd.Next(2,10); // 2 ile 10 arası rastgele sayı
dizi[1] = rnd.Next(50);   //0 ile 50 arası
dizi[2] = rnd.Next();  //rastgele integer değer
Console.WriteLine(dizi.GetValue(2));
//dizi değişkeninin 2. indis elemanını (3.eleman) yazar.
Console.WriteLine(dd[1][0]);
```

23	234
44	

Örnek 15: foreach döngü

```
string[] isimler ={ "ali", "ahmet", "selda", "canan", "melike" };
Console.WriteLine("aranan isim=");

string aranan=Console.ReadLine();

foreach(ss in isimler)   {

 if (aranan.Equals(isimler[0]))
  Console.WriteLine("aranan isim bulundu...");
   else Console.WriteLine("isim yok");
   }
   Console.ReadKey();
```