



# Operatörler

# Operatör Nedir?

 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 yapılarına ve işlevlerine göre sınıflandırılabilir.

– Yapılarına göre:

- Unary (tek operandı olan) Operatörler
- Binary (iki operandı olan) Operatörler
- Ternary (üç operandı olan) Operatörler



## Unary Operatörler:

- `()` , `[]` , `++` , `--` , `+` , `-` , `!` , `~` , `new` , `checked` , `unchecked` , `typeof` , `sizeof`



## Binary Operatörler

- `*` , `/` , `%` , `+` , `-` , `<` , `<=` , `>` , `>=` , `as` , `is` , `<<` , `>>` , `==` , `!=` , `&` , `^` , `|` , `&&` , `||` , `=` , `*=` , `/=` , `%=` , `+=` , `-=` , `<<=` , `>>=` , `&=` , `^=` , `|=`



## Ternary Operatörler

- `?:`



## İşlevlerine Göre:

- Aritmetik Operatörler
- Karşılaştırma Operatörleri
- Mantıksal Operatörler
- Bitsel Operatörler
- Atama ve İşlemlili Atama Operatörleri
- Özel Amaçlı Operatörler



## Aritmetik Operatörler

- + , - , \* , / , % , ++ , --



## Karşılaştırma Operatörleri

- > , < , <= , >= , == , != , as , is



## Mantıksal Operatörler

- || , && , !



## Bitsel Operatörler

- | , & , ~ , ^ , << , >>



## Atama ve İşlemlili Atama Operatörleri

- = , \*= , /= , %= , += , -= , <<= , >>= , &= , ^= , |=



## Özel Amaçlı Operatörler

- ? : , ( ) , [ ] , + , - , & , \* , -> , . , new , checked , unchecked ,  
typeof , sizeof

# Operatör Önceliği



Bazı ifadelerde birden fazla operatör kullanılmış olabilir. Bu gibi durumlarda operatörlerin belirli bir çalışma sırası olacaktır.

$$- X = 6 + 8 * 10$$



Bu ifadeden iki farklı sonuç elde edilebilir:

$$- X = (6 + 8) * 10 = 140$$

$$- X = 6 + (8 * 10) = 86$$

- ✓ Bu durumda operatör önceliği kavramı devreye girecektir. Bütün dillerde olduğu gibi C# dilinde de operatör önceliği vardır. Mantıksal bir yol izlenerek operatörlere öncelik sırası verilmiştir.
- ✓ Örneğimizi tekrar yorumlarsak çarpma işleminin toplama işlemine göre daha öncelikli olduğunu söylemek mümkündür. Bu yüzden işlem sonucu
  - $X = 6 + (8 * 10) = 86$  olacaktır.





Bütün dillerin kendine ait bir operatör öncelik tablosu bulunmaktadır. Bu sayede doğru bir öncelik sıralaması yapılabilmektedir.



Eğer ifadelerde paranteze alınmış kısımlar varsa matematik kurallarından da hatırlayacağımız gibi öncelik parantez içindeki işlemlerdir.

- $y = ( 7 + ( 6 + 4 ) * 5 ) * 2$

- İşlemi sonucunda y değeri 114 olacaktır.



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.

## Örnek:

–  $a = 8 + 5 - 3$

– İşleminde ilk önce  $(8 + 5)$  sonunda da  $(13 - 3)$  işlemleri gerçekleşecektir.

Toplama ve çıkarma işlemleri için sonuçta bir değişiklik gözlemlenmeyebilir fakat bu soldan sağa işlem önceliği bazı durumlarda farklı sonuçlar verebilir.



Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
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

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



Bir önceki sunuda verilen tablo C# operatörlerinin öncelik ilişkilerini göstermektedir. 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.



Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
using System;

class Operatorler2
{
    static void Main()
    {
        int i = 4 +- 6;
        Console.WriteLine(i);
    }
}
```

# Aritmetik Operatörler



**+ , - , \* ve / operatörleri:**

– Toplama, çıkarma, çarpma ve bölme operatörleridir. İstenilen tipteki sayısal değerlerle kullanılabilir. Bölme işleminde kullanılan değerın tipine göre farklı sonuçlar elde edilebilir.

- `int i = 20 + 30;`
- `float f = 12.34f + 56.78f;`





Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
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);
    }
}
```



## % operatörü:

– Bölümden sonra kalanı bulmak yani “mod” almak amacıyla kullanılır. Tüm sayısal türler için kullanılabilir.

- $x = 10 \% 3 \rightarrow x = 1$
- $y = -7 \% 4 \rightarrow y = -3$
- $f = 2.5f \% 1.2f \rightarrow f = 0.1$



## ++ 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.

```
using System;

class Operatorler4
{
    static void Main()
    {
        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);
    }
}
```



Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
using System;

class Operatorler5
{
    static void Main()
    {
        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);
    }
}
```



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 Operatörleri



## > ve < Operatörleri:

- İki sayısal değerin birbirlerine göre büyüklüğünü ve küçüklüğünü kontrol eder. Bu operatörlerin ürettiği değerler “true” ve “false” değerleridir. < operatörü için eğer birinci operand ikinci operandtan küçük ise “true” aksi durumda “false” üretir. > operatörü için de birinci operand ikinciden küçükse “false” büyükse “true” değerlerini üretir.



## >= ve <= Operatörleri:

- > ve < operatörleriyle tamamen aynıdır. Yalnızca eşitlik söz konusu olduğunda da “true” değeri üretirler.

```
using System;

class Operatorler6
{
    static void Main()
    {
        bool b1 = 20 < 30;
        bool b2 = 30 < 20;
        bool b3 = 20.53 > 20.532;
        bool b4 = 12.3f > 11.9f;
        bool b5 = 15 <= 15;
        bool b6 = 20 >= 30;

        Console.WriteLine(b1);
        Console.WriteLine(b2);
        Console.WriteLine(b3);
        Console.WriteLine(b4);
        Console.WriteLine(b5);
        Console.WriteLine(b6);
    }
}
```





## == ve != Operatörleri:

– İki değerin eşit olup olmadığını karşılaştıran operatörlerdir. == operatörü eşitlik durumunda “true”, != operatörü ise eşitsizlik durumunda “true” üretir.

- `bool b1 = 100 == 100; → true`
- `bool b2 = 100 != 100; → false`
- `bool b3 = 100 == 99; → false`
- `bool b3 = 100 != 99; → true`



## as Operatörü:

– Uygun türler arasındaki dönüşümü sağlar. Yaygın olarak kullanılmaz. as operatörü sonucunda referans türde değer elde edilir. Genellikle object türündeki nesnelere farklı bir türe çevirmek için kullanılır. çevrim işlemi başarısız ise null değeri üretir. Kullanımı:

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



## is Operatörü:

- Çalışma zamanında bir nesnenin türünün operand olarak verilen türle uyumlu olup olmadığını kontrol eder. Kullanımı:
  - `<ifade> is <tür>`
- Çok kullanılan bir yapı değildir. Operandlarla belirtilen türler uyumlu ise “true” aksi halde “false” değeri üretir. Bu operatör kullanıldığında derleyici uyarı mesajları verebilir.



Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
using System;

class Operatorler8
{
    static void Main()
    {
        int i = 10;

        Console.WriteLine(i is int);
        Console.WriteLine(i is double);
        Console.WriteLine(i is object);
    }
}
```

# Mantıksal Operatörler



## && (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:

• <u>1.Operand</u>	<u>2.Operand</u>	<u>Sonuç</u>
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**
- `bool b3 = -12.35f > -12.34f && 0 != 1 ;`  
– **false**



## || (VEYA) Operatörü:

- “true” ya da “false” değerindeki operandları mantıksal VEYA ile işler. Operandlardan biri “true” ise “true” değeri üretir.

Kullanımı:

- <ifade> || <ifade>

- Mantıksal VEYA işleminin doğruluk tablosu:

• <u>1.Operand</u>	<u>2.Operand</u>	<u>Sonuç</u>
True	True	True
True	False	True
False	True	True
False	False	False

- `bool b1 = 25 < 15 || 5 == 50 ;`  
– **false**
- `bool b2 = 25 > 15 || 5 != 5 ;`  
– **true**
- `bool b3 = -12.34f > -12.35f || 0 == 1 ;`  
– **true**





## ! (Değil) Operatörü:

- Tek operand alır ve mantıksal Değil (NOT) işlemini uygular.

Kullanımı:

- ! <ifade>

- Mantıksal DEĞİL işleminin doğruluk tablosu

<u>Operand</u>	<u>Sonuç</u>
True	False
False	True

- `bool b1 = ! (25 < 10) ;`
  - **true**
- `bool b2 = ! (10 != 20) ;`
  - **false**

# Bitsel Operatörler

- ✓ Mantıksal operatörler nesnelere üzerinde karşılaştırma gibi işlemler yaparak direkt doğru ya da yanlış sonuç üreten operatörlerdir.
- ✓ 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.
```



## ~ (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 Operatorler9
{
    static void Main()
    {
        byte b1 = 254;
        byte b2 = (byte)~b1;
        Console.WriteLine(b2);
    }
}
```



## & (Bitsel VE) Operatörü:

- İki operandın bitlerini karşılıklı olarak VE işlemine tabi tutar.

byte a = 153;

byte b = 104;

byte x = (byte)(a & B)

→ x = 8

153 : 1001 1001

104 : 0110 1000

-----

VE 0000 1000



## | (Bitisel VEYA) Operatörü:

- İki operandın bitlerini karşılıklı olarak VEYA işlemine tabi tutar.

byte a = 153;

byte b = 104;

byte x = (byte)(a | b)

→ x = 249

153 :1001 1001

104 :0110 1000

-----

VEYA 1111 1001



## **^ (Bitisel Özel VEYA) Operatörü**

- İki operandın bitlerini karşılıklı olarak Özel VEYA (XOR) işlemine tabi tutar.

byte a = 153;

byte b = 104;

byte x = (byte)(a ^ b)

→ x = 241

153 : 1001 1001

104 : 0110 1000

-----

XOR 1111 0001



Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
using System;

class Operatorler10
{
    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);
    }
}
```





## << (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  
byte c = (byte)(b << 4) // 1111 0000
```



## >> (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
```



Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
using System;

class Operatorler11
{
    static void Main()
    {
        byte b = 0xFF;

        b = (byte) (b << 4) ;
        Console.WriteLine (b) ;

        b = (byte) (b >> 3) ;
        Console.WriteLine (b) ;
    }
}
```

# Atama ve İşlemlili Atama Operatörleri



## = (Atama) Operatörü:

- Bir değişkene herhangi bir değer atamak için kullanılır. Çok bilinen ve kullanılan bir operatördür. Kullanımı ile dikkat edilmesi gereken en önemli nokta = operatörü atama yaptığı değeri üretir.

```
using System;
class Operatorler12
{
    static void Main()
    {
        byte b = 5, a = 2, c;
        (c = a) = b;
        Console.WriteLine(c);
    }
}
```



$\ast=$  ,  $/=$  ,  $\%=$  ,  $\ast=$  ,  $\ast=$  ,  $\ast=$  ,  $\ast=$  ,  $\ast=$  ,  $\ast=$  ,  $\ast=$  gibi operatörler işlemlili atama operatörleri olarak geçer ve işlem sonucunun işlenen operanda atandığı durumlarda kullanılır.

- $x = x + y$ ; ifadesi yerine  $x \ast= y$ ;
- $a = a \ast b$ ; ifadesi yerine  $a \ast= b$ ;
- $b = b \ast n$ ; ifadesi yerine  $b \ast= n$ ;

gibi ifadeler örnek olarak verilebilir.

# Özel Amaçlı Operatörler



Bu grupta değişiklikleri işleri yerine getirmek için şu ana kadarki gördüklerimizden farklı operatörler bulunmaktadır.



## ? : (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”



### () Operatörü (Tür Dönüştürme):

İfade önüne yazıldığında tür dönüştürme operatörü olarak çalışır.



### [] Operatörü (İndeks):

Dizilerde elemanın indeksini belirtmek için kullanılır.



### + ve – (İşaret Operatörleri):

Bir değişkenin içeriğindeki değer negatif ya da pozitif olarak işlem görmesini sağlar. + aynı zamanda stringlerde 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.





## . 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.

- OrnekSinif s = new OrnekSinif();



## checked ve unchecked Operatörleri

## 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.



Aşağıdaki kod parçasını bilgisayarınızda deneyiniz:

```
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());
    }
}
```