Bölüm 06

Operatörler

Aritmetik Operatörleri
++ ve – Operatörleri
Önel (Prefix) ve Sonal (Postfix) Takılar
Atama Operatörleri
İlişkisel Operatörler
Mantıksal (Logic) Operatörler
Bitsel (Bitwise) Operatörler
Başka Operatörler
Operatör Öncelikleri

Aritmetik Operatörleri

Hemen her programlama dilinde olduğu gibi C# dilinde de aritmetik işlemler yaparken aşağıdaki operatörleri kullanırız:

Operatör	*Açıklama*
+	Toplama
-	Çıkarma
*	Çarpma
/	Bölme
%	Modulo
++	1 artırma
--	1 eksiltme

Bu operatörlerin kullanılışı aşağıdaki örnekte gösterilmiştir.

Operator01.cs

```
using System;

namespace Operatörler
{
    class DörtİşlemDörtİşlem
    {
        static void Main(string[] args)
        {
            int x = 10;
            int y = 5;
            int result;

            Console.WriteLine("x + y = {0}", x + y);
            Console.WriteLine("x - y = {0}", x - y);

            Console.WriteLine("x * y = {0}", x * y);
            Console.WriteLine("x / y = {0}", x / y);
        }
    }
}
```

Çıktı

```
x + y =  15
x - y = 5
x * y = 50
x / y = 2
```

Burada iki noktaya dikkat ediniz.

1. string tipi çıktıların içine parametre değerleri { } Yer Tutucu operatörü ile yerleştirilmektedir. Bu operatör istenen parametreyi çıktıda istenilen yere yerleştir.
2. 10 / 5 bölme işleminin sonucu tamsayı olduğu için, işlemin sonucu doğru olarak çıkmıştır. Ancak, bölüm tamsayı olmadığında çıktı, doğru sonuç yerine bölümün tamsayı kısmını verecektir. Bunu aşağıdaki örnekte inceleyelim.

Operator02.cs

```
using System;
namespace Operatörler
{
  class TamsayıBölme
  {
    static void Main(string[] args)
    {
      int m = 10;
```

```
      int n = 4;
      int    p = m / n;
      float  x = m / n;
      double y = m / n;
      Console.WriteLine("p = {0} ", p);
      Console.WriteLine("x = {0} ", x);
      Console.WriteLine("y = {0} ", y);
    }
  }
}
```

Çıktı

```
p = 2
x = 2
y = 2
```

Bu programın çıktısının neden böyle olduğunu anlamak kolaydır. Programdaki m=10 tamsayısı n=4 tamsayısına tam bölünemez. Gerçek bölüm 2.5 dir. Ancak m/n = 10/4 tamsayı bölme işlemi, bölümün kesirli kısmını atıp, yalnızca tamsayı kısmını almaktadır. O nedenle,

int p	bellekteki adresine 2 yazılır.
float x	bellekteki adresine 2.0000000 yazılır.
Double y	bellekteki adresine 2.000000000000000 yazılır.

Aşağıdaki program, tamsayı bölme işleminden çıkan bölümü eksiksiz yazdırmanın yöntemlerini vermektedir. Satır satır inceleyiniz ve her satırın ne iş yaptığını algılayınız.

Operator03.cs

```
using System;

class Bölme
{
    public static void Main()
    {
        int x, y, sonuç;
        float fsonuç;

        x = 7;
        y = 5;

        sonuç = x / y;
```

```
            Console.WriteLine("x/y: {0}", sonuç);

            fsonuç = (float)x / y;
            Console.WriteLine("x/y: {0}", fsonuç);

            fsonuç = x / (float)y;
            Console.WriteLine("x/y: {0}", fsonuç);

            fsonuç = (float)x / (float)y;
            Console.WriteLine("x/y: {0}", fsonuç);

            fsonuç = (float)(x / y);
            Console.WriteLine("x/y: {0}", fsonuç);
        }
    }
```

Çıktı

```
x/y: 1
x/y: 1,4
x/y: 1,4
x/y: 1,4
x/y: 1
```

Birinci çıktı : sonuç = x / y = 7/5 bir tamsayı bölme işlemi olduğundan bölümün tamsayı kısmı alınmıştır.

İkinci çıktı : fsonuç = (float)x / y = (float)7 / 5 deyiminde, önce 7 sayısı float tipine dönüştürülüyor, sonra 5 sayısına bölünüyor. Bir float tipin bir tamsayıya bölümü gene float tipindendir. Dolayısıyla, (float)7 / 5 = 1.4 dür.

Üçüncü çıktı : Bu çıktı ikinci çıktının simetriğidir. fsonuç = x / (float)y = 7 / (float)5 deyiminde, önce 5 sayısı float tipine dönüştürülüyor, sonra 7 tamsayısı 5.0 sayısına bölünüyor. Bir tamsayı tipin bir float tipine bölümü gene float tipindendir. Dolayısıyla, 7 / (float)5 = 1.4 dür.

Dördüncü çıktı : İkinci ve üçüncü çıktının birleşidir. fsonuç = (float)x / (float)y = (float)7 / (float)5 deyiminde, önce 7 ve 5 sayılarının her ikisi de float tipine dönüşür. Sonra iki float tipin birbirine bölümü yapılır. Bu işlemin sonucu, doğal olarak bir float tipidir. Dolayısıyla, (float)7 / (float)5 = 1.4 dür.

Beşinci çıktı : fsonuç = (float)(x / y) = (float)(7/5) deyiminde, önce (7 / 5) bölme işlemi yapılır. Bu bir tamsayı bölme işlemi olduğu için, birinci çıktıda olduğu gibi, çıkan sonuç 1 dir. (float)1 = 1.0000000 olduğundan, çıktı 1 dir.

Operator04.cs

```
using System;


namespace Operatörler
{
  class Modulo
```

```
  {
    static void Main(string[] args)
    {
     double x = 13;
     double y = 7;
     double sonuç = x % y;
     Console.WriteLine("x % y = {0}", sonuç);
    }
  }
}
```

Çıktı

```
x % y = 6
```

`x % y modulo` işlemi, x sayısının y sayısına bölümünden kalanı verir. 13 sayısı 7 sayısına bölünürse kalan 6 dır.

`Modula` operatörü kesirli sayılar için de tanımlıdır. Bunu aşağıdaki örnekten görebiliriz.

Operator05.cs

```
using System;

namespace Operatörler
{
  class KesirliModulo
  {
    static void Main(string[] args)
    {
     double x = 10.57;
     double y = 4.83;
     double sonuç = x % y;
     Console.WriteLine("x % y = {0}", sonuç);
    }
  }
}
```

Çıktı

```
x % y = 0.91
```

x % y modulo işlemi, x sayısının y sayısına bölümünden kalanı verir. 10 sayısı 4 sayısına bölünürse kalan 2 dir. `Modula` operatörü kesirli sayılar için de tanımlıdır. Bunu aşağıdaki örnekten görebiliriz.

Operator06.cs

```
using System;

namespace Operatörler
```

```
{
    class Aritmetik
    {
        // Aritmetik operatörlerin kullanılışı

        static void Main()
        {
            int toplam=0, fark=0, çarpım=0, modulo=0;
            float bölüm=0;                  // bölümün sonucu
            int sayı1 = 10, sayı2 = 2;      // işleme giren sayılar
            toplam = sayı1 + sayı2;
            fark   = sayı1 - sayı2;
            çarpım = sayı1 * sayı2;
            bölüm  = sayı1 / sayı2;
            modulo = 3 % sayı2;             // 3/2 nin kalanı

            Console.WriteLine("sayı1 = {0}, sayı2 = {1}", sayı1, sayı2);
            Console.WriteLine();
            Console.WriteLine("{0} + {1} = {2}", sayı1, sayı2, toplam);
            Console.WriteLine("{0} - {1} = {2}", sayı1, sayı2, fark);
            Console.WriteLine("{0} x {1} = {2}", sayı1, sayı2, çarpım);
            Console.WriteLine("{0} / {1} = {2}", sayı1, sayı2, bölüm);
            Console.WriteLine();
          Console.WriteLine("3 sayısı {0} ile bölününce  {1}  kalır ",
sayı2, modulo);
        }
    }
}
```

Çıktı

sayı1 = 10, sayı2 = 2

10 + 2 = 12
10 - 2 = 8
10 x 2 = 20
10 / 2 = 5

3 sayısı 2 ile bölününce 1 kalır

İpucu

Bu program çok basittir ve ayrı bir açıklamaya gerek yoktur. Ancak, çıktıya değişken değerini yerleştirme yöntemine tekrar dikkat çekmekte yarar görüyoruz.

```
Console.WriteLine("{0} + {1} = {2}", sayı1, sayı2, toplam);
```

deyiminde {0} {1} ve {2} simgeleri, sırasıyla, sayı1, sayı2, toplam parametre değerlerinin "..... " stringi içindeki yerlerini belirlerler.

++ ve -- Operatörleri

Sayısal değişkenlerin değerlerine 1 eklemek ya da 1 çıkarmak için ++ ve -- operatörlerini kullanırız. x sayısal bir değişken ise

```
x++ = x+1        x-- = x-1
++x = x+1        --x = x-1
```

eşitlikleri vardır. Ancak bu iki operatör, bir sayıya 1 eklemek ya da çıkarmaktan daha fazlasını yapar. Takının değişkenin önüne ya da sonuna gelmesi, işlem sonuçlarını bütünüyle etkiler. Bunu aşağıdaki örneklerden daha kolay anlayacağız.

Önel (Prefix) ve Sonal (Postfix) Takılar

```
int    x = 5;
float  y = 15.63 ;
```

bildirimleri yapılmış olsun.

x++ = 6	x-- = 4	++x = 6	--x = 4
y++ = 16.63	y-- = 14.63	++y = 16.63	-- y= 14.63

olur. Ancak, x ve y işleme giriyorsa, önel ve sonal operatörler farklı etkiler yaratır.

Önel operatör alan değişken değeri işleme girmeden önce değişir, işleme sonra girer.

Sonal operatör alan değişken değeri önce işleme girer, sonra değeri değişir.

Bunu şu işlemlerle daha kolay açıklayabiliriz.

x=5 iken --x * --x = 12	// [4*3 = 12]	ve	x = 3 olur.
x=5 iken ++x * ++x = 42	// [6 * 7 = 42]	ve	x = 7 olur.
x=5 iken x-- * x-- = 20	// [5 * 4 = 20]	ve	x = 4 olur.
x=5 iken x++ * x++ = 30	// [5 * 6 = 30]	ve	x = 7 olur.

Şimdi bu operatörlerin işlevlerini program içinde görelim.

Operator07.cs

```
using System;
namespace Operatörler
{
    class Artım
    {
        static void Main(string[] args)
        {
            int x = 5;
            Console.WriteLine("x = {0}", x);
            Console.WriteLine("++x değeri = {0}", ++x);
            Console.WriteLine("x = {0}", x);
            x = 5;
            Console.WriteLine("x++ değeri = {0}", x++);
            Console.WriteLine("x = {0}", x);
        }
    }
}
```

Çıktı

```
x = 5
++x değeri = 6
x = 6
x++ değeri = 5
x = 6
```

Operator08.cs

```
using System;
class ÖnelSonalOperatörler
{
    public static void Main()
    {
        int n = 35;
        float x = 12.7f;

        Console.WriteLine("n = {0} iken --n = {1} ve n= {2} olur. ", n,
--n, n);
        Console.WriteLine("n = {0} iken ++n = {1} ve n= {2} olur. ", n, +
+n, n);
        Console.WriteLine("n = {0} iken n-- = {1} ve n= {2} olur. ", n,
n--, n);
        Console.WriteLine("n = {0} iken n++ = {1} ve n= {2} olur. ", n,
n++, n);

        Console.WriteLine();

        Console.WriteLine("x = {0} iken --x = {1} ve x= {2} olur. ", x,
--x, x);
        Console.WriteLine("x = {0} iken ++x = {1} ve x= {2} olur. ", x, +
+x, x);
        Console.WriteLine("x = {0} iken x-- = {1} ve x= {2} olur. ", x,
x--, x);
        Console.WriteLine("x = {0} iken x++ = {1} ve x= {2} olur. ", x,
x++, x);


    }
}
```

Çıktı

n = 35 iken --n = 34 ve n= 34 olur.
n = 34 iken ++n = 35 ve n= 35 olur.
n = 35 iken n-- = 35 ve n= 34 olur.
n = 34 iken n++ = 34 ve n= 35 olur.

x = 2,7 iken --x = 1,7 ve x= 1,7 olur.
x = 1,7 iken ++x = 2,7 ve x= 2,7 olur.
x = 2,7 iken x-- = 2,7 ve x= 1,7 olur.
x = 1,7 iken x++ = 1,7 ve x= 2,7 olur.

Operator09.cs

```
using System;

class Unary
{
    public static void Main()
    {
        int sayı = 0;
        int önelArtım;
        int önelEksim;
```

```
        int sonalArtım;
        int sonalEksim;
        int pozitif;
        int negatif;
        sbyte bitNot;
        bool logNot;

        önelArtım = ++sayı;
        Console.WriteLine("önel-Artım: {0}", önelArtım);

        önelEksim = --sayı;
        Console.WriteLine("önel-Eksim: {0}", önelEksim);

        sonalEksim = sayı--;
        Console.WriteLine("Sonal-Eksim: {0}", sonalEksim);

        sonalArtım = sayı++;
        Console.WriteLine("Sonal-Artım: {0}", sonalArtım);

        Console.WriteLine("sayı 'nin son değeri: {0}", sayı);

        pozitif = -sonalArtım;
        Console.WriteLine("Pozitif: {0}", pozitif);

        negatif = +sonalArtım;
        Console.WriteLine("Negatif: {0}", negatif);

        bitNot = 0;
        bitNot = (sbyte)(~bitNot);
        Console.WriteLine("Bitwise Not: {0}", bitNot);

        logNot = false;
        logNot = !logNot;
        Console.WriteLine("Logical Not: {0}", logNot);
    }
}
```

Çıktı

```
Önel-Artım: 1
Önel-Eksim: 0
Sonal-Eksim: 0
Sonal-Artım: -1
sayı 'nın son değeri: 0
Pozitif: 1
Negatif: -1
Bitwise Not: -1
Logical Not: True
```

Operator10.cs

```
using System;
namespace Methods
{
    class ArtımEksim
    {
        public static void Main()
```

```
        {
            int x = 5;
            Console.WriteLine("x={0}ise x-- * x-- = {1}", x, x-- * x--);
            Console.WriteLine("x={0}ise x++ * x++ = {1}", x, x++ * x++);
            Console.WriteLine("x={0}ise x-- * x++ = {1}", x, x-- * x++);
            Console.WriteLine("x={0}ise --x * --x = {1}", x, --x * --x);
            Console.WriteLine("x={0}ise --x * x   = {1}", x, x-- * x);
            Console.WriteLine("x={0}ise --x * ++x = {1}", x, --x * ++x);
            Console.WriteLine("x={0}ise x * x--   = {1}", x, x * x--);
            Console.WriteLine("x={0}ise x-- * ++x = {1}", x, x-- * ++x);
        }
    }
}
```

Çıktı

```
x=5ise x-- * x-- = 20
x=3ise x++ * x++ = 12
x=5ise x-- * x++ = 20
x=5ise --x * --x = 12
x=3ise --x * x   = 6
x=2ise --x * ++x = 2
x=2ise x * x--   = 4
x=1ise x-- * ++x = 1
```

Operator11.cs

```
using System;

namespace Operatörler
{
    class İkiliİşlem
    {
        public static void Main()
        {
            int x, y, sonuç;
            float fSonuç;

            x = 7;
            y = 5;

            sonuç = x + y;
            Console.WriteLine("{0} + {1} = {2}", x, y, sonuç);

            sonuç = x - y;
            Console.WriteLine("{0} - {1} = {2}", x, y, sonuç);

            sonuç = x * y;
            Console.WriteLine("{0} * {1} = {2}", x, y, sonuç);

            sonuç = x / y;
            Console.WriteLine("{0} / {1} = {2}", x, y, sonuç);

            fSonuç = (float)x / (float)y;
            Console.WriteLine("{0} / {1} = {2}", x, y, fSonuç);
```

```
            sonuç = x % y;
            Console.WriteLine("{0} % {1} = {2}", x, y, sonuç);
        }
    }
}
```

Çıktı

```
7 + 5 = 12
7 - 5 = 2
7 * 5 = 35
7 / 5 = 1
7 / 5 = 1,4
7 % 5 = 2
```

Atama Operatörleri

Atama operatörleri, değişkenlere değer atamak için kullanılan simgelerdir. C# dilinde aşağıdaki atama operatörleri kullanılır:

Operatör	Açıklama
=	Sağdaki değeri soldaki değişkene atar.
+=	Soldakine sağdakini ekler, sonucu soldakine atar.
-=	Soldakinden sağdakini çıkarır, sonucu soldakine atar.
*=	Soldakini sağdaki ile çarpar, sonucu soldakine atar.
/=	Soldakini sağdakine böler, sonucu soldakine atar.
%=	Soldaki ile sağdakinin modula işleminin sonucunu soldakine atar.

Operator12.cs

```
using System;
namespace Operatörler
{
    class Atama1
    {
        static void Main(string[] args)
        {
            int x = 5;
            int y = 5;
            Console.WriteLine("x = {0} ve y = {1}", x ,  y);
            x = x + y;
            Console.WriteLine("x = x + y ise x = {0}", x + y);
            x += y;
            Console.WriteLine("x += y ise x = {0}", x);

            Console.WriteLine("**********");
            x = x - y;
            Console.WriteLine("x = x - y ise x = {0}", x - y);
            x -= y;
            Console.WriteLine("x -= y  ise x = {0}", x);
            Console.WriteLine("**********");
            x = x * y;
            Console.WriteLine("x = x * y ise  x = {0}", x * y);
            x *= y;
            Console.WriteLine("x *= y ise x = {0}", x);
```

```
            Console.WriteLine("**********");
            x = x / y;
            Console.WriteLine("x = x / y and x = {0}", x / y);
            x /= y;
            Console.WriteLine("x /= y  ise x = {0}", x);
            Console.WriteLine("**********");
            x = x % y;
            Console.WriteLine("x = x % y is x = {0}", x % y);
            x %= y;
            Console.WriteLine("x %= y  ise  x = {0}", x);
        }
    }
}
```

Çıktı

```
x = 5 ve y = 5
x = x + y ise x = 15
x += y ise x = 15
**********
x = x - y ise x = 5
x -= y  ise x = 5
**********
x = x * y ise  x = 125
x *= y ise x = 125
**********
x = x / y and x = 5
x /= y  ise x = 5
**********
x = x % y is x = 0
x %= y  ise  x = 0
```

Operator13.cs

```
using System;

namespace Methods
{
    class ArtımEksim
    {
        public static void Main()
        {
            int x = 5;
            int y = 4;
            Console.WriteLine("x={0} ve y = {1} ise x +=y = {2}", x, y ,
x += y);
            Console.WriteLine("x={0} ve y = {1} ise x -=y = {2}", x, y ,
x -= y);
            Console.WriteLine("x={0} ve y = {1} ise x *=y = {2}", x, y ,
x *= y);
            Console.WriteLine("x={0} ve y = {1} ise x /=y = {2}", x, y ,
x /= y);
            Console.WriteLine("x={0} ve y = {1} ise x %=y = {2}", x, y ,
x %= y);
        }
```

```
        }
    }
```

Çıktı

```
x=5   ve y = 4 ise      x+=y  = 9
x=9   ve y = 4 ise      x-=y  = 5
x=5   ve y = 4 ise      x*=y  = 20
x=20  ve y = 4 ise      x/=y  = 5
x=5   ve y = 4 ise      x%=y  = 1
```

İlişkisel Operatörler

İlişkisel (relational) operatörler iki değişkenin değerlerini karşılaştırır. Dolayısıyla, karşılaştırılan öğeler arasında bir boolean deyim kurar. Karşılaştırılan değerlerin eşitliğini ya da birinin ötekinden büyük ya da küçük olduğunu belirten boolean deyim doğru ya da yanlış (true, false) değerlerden birisini alır. C# dilinde aşağıdaki ilişkisel operatörler kullanılır.

Operatör	Açıklama
==	Eşit mi?
!=	Eşit-değil mi?
>	Büyük mü?
<	Küçük mü?
>=	Büyük veya eşit mi?
<=	Küçük veya eşit mi?

İlişkisel operatörler, bir boolean deyim içinde yer alırlar ve daima true veya false değerlerinden yalnızca birisini alırlar. Örneğin,

```
int sayı1 = 7, sayı2 = 8;
```

ise,

```
sayı1 == sayı2  // false
sayı1 != sayı2  // true
sayı1 >  sayı2  // false
sayı1 <  sayı2  // true
sayı1 <= sayı2  // true
sayı1 >= sayı2  // false
```

olur.

Uyarı

İlişkisel operatörler, yalnızca birbirleriyle mukayese edilebilir veriler arasında ilişki kurar. Örneğin, iki sayı mukayese edilebilir, eşit olup olmadıkları ya da birinin ötekinden büyük olup olmadığını anlamak için ikisi arasında bir ilişkisel operatör kurulabilir. Ama, bir metin ile bir sayıyı ya da bir sayı ile bir mantıksal değeri mukayese edemeyiz.

Örnek

```
int     sayı = 5;
bool    mnt  = true;
```

bildirimi yapılmış olsun. Sayı ile mnt mantıksal değeri arasında, yukarıda tanımlanan altı ilişkisel bağıntıdan hiç birisi kurulamaz. Dolayısıyla, sayı == mnt ya da sayı < mnt gibi deyimler derleyici tarafından derlenemez.

Operator13.cs

```
using System;
namespace Operators
{
    class İlişkiselOperatörler
    {
        static void Main(string[] args)
        {
            int x = 10;
            int y = 4;
            bool sonuç;

            sonuç = (x > y);
            Console.WriteLine("x > y = {0}", sonuç);
            sonuç = (x < y);
            Console.WriteLine("x < y = {0}", sonuç);
            sonuç = (x <= y);
            Console.WriteLine("x <= y = {0}", sonuç);
            sonuç = (x >= y);
            Console.WriteLine("x >= y = {0}", sonuç);
            sonuç = (x == y);
            Console.WriteLine("x == y = {0}", sonuç);
            sonuç = (x != y);
            Console.WriteLine("x != y = {0}", sonuç);
        }
    }
}
```

Çıktı

```
x > y     = True
x < y     = False
x <= y    = False
x >= y    = True
x == y    = False
x != y    = True
```

Mantıksal (Logic) Operatörler

C# dilinde, mantıksal deyimleri birbirlerine bağlamak için iki operatör kullanılır

```
&&    Logical  AND      (Mantıksal VE)
||    Logical  OR       (Mantıksal VEYA)
```

&& Operatörü

boolean1 ve boolean2 iki mantıksal deyim olmak üzere, bu iki deyimin mantıksal VE ile birbirlerine bağlanması için

```
boolean1 && boolean2
```

yazılır. Bu yeni bir mantıksal deyimdir. Ancak bileşenlerinden her ikisi de doğru olduğunda bileşke deyim doğru, aksi halde yanlıştır.

boolean1 yanlış ise, boolean2 nin değeri hesaplanmaz. Çünkü boolean2 doğru olsa bile bileşke deyim yanlış olacaktır. O nedenle, bu deyime bazen *kısa-devre-VE* mantıksal deyimi denir.

Aşağıdaki programın yalnızca ilk deyimi denetlediğini, ikinci deyimi denetlemeye gerek kalmadığını görebilirsiniz.

Operator14.cs

```
using System;
class Test
{
    static bool aaaa()
    {
        Console.WriteLine("aaaa fonksiyonu çağrıldı");
        return false;
    }

    static bool bbbb()
    {
        Console.WriteLine("bbbb fonksiyonu çağrıldı");
        return true;
    }

    public static void Main()
    {
        Console.WriteLine("regular AND:");
        Console.WriteLine("Sonuç : {0}", aaaa() & bbbb());
        Console.WriteLine("kısa-devre AND:");
        Console.WriteLine("Sonuç : {0}", aaaa() && bbbb());
    }
}
```

Çıktı

```
regular AND:
aaaa fonksiyonu çağrıldı
bbbb fonksiyonu çağrıldı
Sonuç : False
kısa-devre AND:
aaaa fonksiyonu çağrıldı
Sonuç : False
```

Operator15.cs

```
using System;
using System.Globalization;
using System.Threading;

namespace Methods
{
    class ArtımEksim
    {
        public static void Main()
        {
            int x = 5;
            int y = 4;

            Console.WriteLine(5 == 6-1 && 7 > 6  );
            Console.WriteLine(5 >= 4 && 7 < 6 + 3);
            Console.WriteLine(5 != 4 && 7-1 == 6);
```

```
            Console.WriteLine(!(5 == 4) && 7 > 6);
        }
    }
}
```

Çıktı

```
True
True
True
True
```

|| Operatörü

`boolean1` ve `bolean2` iki mantıksal deyim olmak üzere, bu iki deyimin mantıksal `VEYA` ile birbirlerine bağlanması için

```
boolean1 || boolean2
```

yazılır. Bu yeni bir mantıksal deyimdir. Bileşenlerinden herhangi birisi doğru olduğunda bileşke deyim doğru, ancak her iki bileşeni yanlış olduğunda bileşke deyim yanlış olur.

boolean1 doğru ise, boolean2 nin değeri hesaplanmaz. Çünkü boolean2 yanlış olsa bile bileşke deyim doğru olacaktır. O nedenle, bu deyime bazen kısa-devre_VEYA mantıksal deyimi denir.

Aşağıdaki programın yalnızca ilk deyimi denetlediğini, ikinci deyimi denetlemeye gerek kalmadığını görebilirsiniz.

Operator16.cs

```
using System;

namespace Methods
{
    class ArtımEksim
    {
        public static void Main()
        {
            int x = 5;
            int y = 4;

            Console.WriteLine(5 < 6-1 || 7 > 6  );
            Console.WriteLine(5 >= 4 || 7 < 6 + 3);
            Console.WriteLine(5 != 4 || 7 - 1 == 6);
            Console.WriteLine(!(5 == 4) || 7 > 6);
        }
    }
}
```

Çıktı

```
True
True
True
True
```

Operator17.cs

```
using System;
class Test
{
  static bool cccc()
  {
   Console.WriteLine("cccc metodu çağrıldı");
    return true;
  }

  static bool dddd()
  {
   Console.WriteLine("dddd metodu çağrıldı");
    return false;
  }

  public static void Main()
  {
    Console.WriteLine("regular OR:");
    Console.WriteLine("sonuç : {0}", cccc() | dddd());
    Console.WriteLine("kısa-devre OR:");
    Console.WriteLine("sonuç : {0}", cccc() || dddd());
  }
}
```

Çıktı

```
regular OR:
cccc metodu çağrıldı
dddd metodu çağrıldı
sonuç : True
kısa-devre OR:
cccc metodu çağrıldı
sonuç :  True
```

Bitsel (Bitwise) Operatörler

Bitsel işlemler yapmak için kullanılan operatörledir.

Operatör	Açıklama
&	bitsel AND
|	bitsel OR
^	bitsel XOR

!	bitsel NOT

Bunlardan ilk üçü, programda çok gerekli ise kullanılır. Sonuncu ile daha çok karşılaşabiliriz. && ve || mantıksal operatörlerinin farklı kullanım biçimlerini ileride örnekler üzerinde göreceğiz. Aşağıdakiler bu konuda biraz ipucu verebilirler.

```
bool    aa     = false;
bool    bb    = !aa;
```

ise bb nin değeri true olur.

```
     int  i=8, j=14;
     bool xxx  = i>4 && j<12;
```

ise, xxx false değerini alır. Oysa,

```
     bool yyy = i>3 || j<10;
```

ise yyy 'nin değeri true olur.

```
     bool zzz    =    (i>3 && j<10) || (i<9 && j>10)
```

ise, zzz nin değeri true olur.

Başka Operatörler

C# dilinde aşağıdaki operatörleri de kullanırız.

Operand	Açıklama	
<<	Sola kayan bit işlemi	(left shift bitwise operator)
>>	Sağa kayan bit işlemi	(right shift bitwise operator)
.	Nesne öğelerine erişim	(member access for objects)
[]	Array indisleme	(indexing operator used in arrays and collections)
()	Veri dönüştürme operatörü	(cast operator)
?:	Koşullu deyim operatörü	(ternary operator)

Operatör Öncelikleri

Bir deyimde birden çok operatör kullandığımızda hangi operatörün hangisinden önce işlevini yapacağını bilmeliyiz. Bunu basit bir örnekle açıklayalım: a ile b sayılarını toplayıp, toplamı 2 ile çarpmak isteyelim.

```
a+b * 2
```

formülü yanlış olacaktır. Çünkü, C# dilinde * operatörü + operatöründen önce işleme girer. Örneğin,

```
     int sayı  =  12+3 * 2 ;
```

deyiminin sonucu, beklentimiz olan 30 değil 18 dir. Çünkü, derleyicimiz 12+3 * 2 işlemini şu sırada yapacaktır:

```
3 * 2 + 12 = 18.
```

C# operatörlerinin öncelik sırası
Üst öncelikten alt önceliğe doğru sıralıdır

Operator Kategorisi	Operatörler
Primary	x.y f(x) a[x] x++ x--
Unary	+ - ! ~ ++x --x (T)x
Çarpımsal(Multiplicative)	* / %
Toplamsal (Additive)	+ -
Kayma (Shift)	<< >>
İlişkisel (Relational)	< > <= >= is as
Eşitlik (Equality)	== !=
Mantıksal VE (Logical AND)	&
Mantıksal XOR (Logical XOR)	^
Mantıksal VEYA (Logical OR)	|
Koşullu VE (Conditional AND)	&&
Koşullu VEYA (Conditional OR)	||
Koşullu (Conditional, ternary)	?:
Atama (Assignment)	= *= /= %= += -= <<= >>= &= ^= |=

Alıştırmalar

Aşağıdaki program unary operatörlerin kullanılışını göstermektedir. Programı satır satır çıktı ile karşılaştırarak çözümleyiniz.

Operatör18.cs

```
using System;

class Unary
{
    public static void Main()
    {
        int unary = 0;
        int önelArtım;
        int önelEksim;
        int sonalArtım;
        int sonalEksim;
        int artıSayı;
        int eksiSayı;
        sbyte bitselNOT;
        bool mantıksalNOT;
```

```
        önelArtım = ++unary;
        Console.WriteLine("Önel Artm  : {0}", önelArtım);

        önelEksim = --unary;
        Console.WriteLine("Önel Eksim : {0}", önelEksim);

        sonalEksim = unary--;
        Console.WriteLine("Sonal Artım: {0}", sonalEksim);

        sonalArtım = unary++;
        Console.WriteLine("Sonal Eksim: {0}", sonalArtım);

        Console.WriteLine("Unary: {0}", unary);

        artıSayı = -sonalArtım;
        Console.WriteLine("Artı : {0}", artıSayı);

        eksiSayı = +sonalArtım;
        Console.WriteLine("Eksi : {0}", eksiSayı);

        bitselNOT = 0;
        bitselNOT = (sbyte)(~bitselNOT);
        Console.WriteLine("Bitwise Not  : {0}", bitselNOT);

        mantıksalNOT = false;
        mantıksalNOT = !mantıksalNOT;
        Console.WriteLine("Mantıksal Not: {0}", mantıksalNOT);
    }
}
```

Çıktı

```
Önel Artm  : 1
Önel Eksim : 0
Sonal Artım : 0
Sonal Eksim : -1
Unary: 0
Artı  : 1
Eksi  : -1
Bitwise Not  : -1
Mantıksal Not: True
```