C Sharp Programlama Dili/Operatörler

Ders 6. Operatörler

Programlama dillerinde tek başlarına herhangi bir anlamı olmayan ancak programın işleyişine katkıda bulunan karakter ya da karakter topluluklarına operatör denir. Örneğin `a+b` ifadesinde + işareti bir operatördür. Operatörlerin etki ettikleri sabit ya da değişkenlere ise operand denir. Örneğin `a+b` ifadesinde a ve b birer operanddır. C#'ta operatörlerın bazıları tek, bazıları çift, bazıları da üç operand alır. Peki eğer bir ifadede birden fazla operatör varsa ne olacak? Örneğin 2+3*7 gibi bir ifadede önce hangi işlem yapılacak? Burada önce toplama işlemi yapılırsa sonuç 35 olur, ancak önce çarpma işlemi yapılırsa sonuç 23 olur. İşte C#'ta bu gibi durumları önlemek için operatör önceliği diye kavram bulunmaktadır. Operatör öncelikleri ile ilgili şu kuralları bilmenizde fayda var:

- Önce parantez içleri yapılır. Örneğin (3+5)*9 gibi bir ifadede önce toplama işlemi yapılır.
- İç içe parantezler söz konusuysa önce en içteki parantez yapılır, sıra ile dıştaki parantezlere gidilir. Örneğin (2+(9+2)*5)*2 gibi bir ifadede önce 9 ile 2 toplanıp 11 değeri bulur, sonra 11 ile 5 çarpılıp 55 değeri bulunur, sonra 55 ile 2 toplanıp 57 değeri bulunur, en son da 57 ile 2 çarpılıp 114 değeri bulunur.
- Dikkat ettiyseniz yukarıdaki örnekte bir parantez söz konusu olmamasına rağmen önce toplama yerine çarpmanın yapıldığı bir durumla karşılaştık. Çünkü C#'ta farklı operatörlerin farklı önceliği vardır. Yani ifadede bir parantez yoksa önce öncelik sırası daha önde olan operatörün işlemi yapılır. Aşağıdaki tabloda operatörler öncelik sırasına göre listelenmiştir. Tablo, öncelik sırası en önde olandan en geride olana göre sıralanmıştır. Yani aynı ifadede parantez söz konusu değilse bu tablodaki daha üstte olan operatör daha önce işletilecektir.

Birinci öncelikliler	x++, x--
Tek operand alan operatörler	+, -, !, ~, ++x, --x, (Tür)x
Çarpma ve bölme	*, /, %
Toplama ve çıkarma	+, -
Kaydırma operatörleri	<<, >>
İlişkisel ve tür testi operatörleri	<, >, <=, >=, is, as
Eşitlik operatörü	==, !=
Bitsel Ve (AND)	&
Bitsel Özel Veya (XOR)	^

Bitsel Veya (OR)	|
Mantıksal Ve	&&
Mantıksal Veya	||
Koşul operatörü	?:
Atama ve işlemli atama operatörleri	=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^=, |=

Dikkat ettiyseniz yukarıdaki tablodaki bazı operatörler aynı sırada, yani bunların öncelik sırası eşit. C#'ta eğer operatörler eşit öncelikliyse, soldan sağa işlem yapılır. Yani 2*6/5 işleminde önce 2 ile 6 çarpılır, sonra da sonuç 5'e bölünür. Çoğu durumda bunun sonuca bir değişiklik yapmadığı düşünülse de bazı durumlarda işleme soldan başlanmasının sağdan başlamaya göre değişik sonuçlar doğurduğu durumlara rastlanabilir. Örneğin:

```
using System;
class Operatorler
{
	static void Main()
	{
		int i=3*5/7;
		Console.Write(i);
	}
}
```

Bu programda ekrana 2 yazılır. Ancak;

```
using System;
class Operatorler
{
	static void Main()
	{
		int i=3*(5/7);
		Console.Write(i);
	}
}
```

Bu programda ekrana 0 (sıfır) yazılır. Çünkü C# sonunda herhangi bir harf olmayan tam sayı sabitleri int sayar ve intin kurallarına göre işlem yapar. int sabitlerle yapılan her işlemden sonra -oluşursa- ondalık kısım atılır. Ancak programı aşağıdaki gibi değiştirirsek sonuç değişmeyecektir.

```
using System;
class Operatorler
{
	static void Main()
	{
		int i=(int)(3f*(5f/7f));
		Console.Write(i);
	}
}
```

Burada sabitleri floatlaştırdık, dolayısıyla da işlemin sonunda float bir sabit oluştu. Sonra tür dönüştürme operatörüyle sonucu inte çevirip i değişkenine atadık ve bu değişkeni de ekrana yazdırdık. Şimdi isterseniz bütün operatörleri tek tek inceleyelim.

Matematiksel operatörler

+, -, / ve * operatörleri hakkında açıklama yapma gereksinmi görmüyorum. Burada henüz görmediğiniz bir operatörü tanıtmak istiyorum. % operatörü mod alma işlemi yapar, yani;

```
using System;
class Operatorler
{
	static void Main()
	{
		int i=5%2;
		Console.Write(i);
	}
}
```

Bu program ekrana 1 yazar. Yani 5 sayısının 2'ye bölümünden kalan i değişkenine atandı. Matematiksel operatörler hakkında şunu eklemek istiyorum: eğer işlem yapılan her iki operarand farklı türdeyse sonuç daha üst kapasiteli türde tutulur. Örnek:

```
using System;
class Operatorler
{
	static void Main()
	{
		float i=5/2f;
		Console.Write(i);
	}
}
```

Burada 5 sayısı da float gibi davranmış ve sonuç ondalıklı çıkmıştır. Ancak,

```
using System;
class Operatorler
{
	static void Main()
	{
		float i=2/5/2f;
		Console.Write(i);
	}
}
```

Burada sonuç 0 çıkar. Çünkü zaten işlem yapılırken sonuç daha floata (2f) gelmeden sıfırlanmıştı.

```
using System;
class Operatorler
{
	static void Main()
	{
		float i=2f/5/2;
		Console.Write(i);
	}
}
```

Burada ise zincirleme olarak her işlemin sonucu floatlaşmakta ve sonuç ondalıklı çıkmaktadır. Daha çarpıcı bir örnek:

```
using System;
class Operatorler
{
	static void Main()
	{
		float i=2f/5/2/2/4/8/4/5/3;
		Console.Write(i);
	}
}
```

Burada da sonuç ondalıklıdır.

Bir artırma ve bir eksiltme operatörleri

C#'ın en çok kullanılan operatörlerindendir. Önüne veya sonuna geldiği değişkeni bir artırır veya bir eksiltirler, sabitlere uygulanamazlar. Hem uygulandığı değişkenin değerini 1 artırır veya eksiltirler hem de tutucu vazifesi görürler. Ön ek ve son ek olmak üzere iki şekilde kullanılabilirler. Örnek;

```
using System;
class Operatorler
{
	static void Main()
	{
		int a=5;
		int i=++a;
		Console.Write(i+" "+a);
	}
}
```

Programda da gördüğünüz gibi ++ operatörü hem a'nın değerini bir artırdı hem de bu değeri tuttu ve değer i'ye atandı. İşte ön ek ve son ek şeklinde kullanımın farkı burada ortaya çıkıyor. Örneği inceleyiniz.

```
using System;
class Operatorler
{
	static void Main()
	{
		int a=5;
		int i=a++;
		Console.Write(i+" "+a);
	}
}
```

Bu programda ++ operatörü ön ek şeklinde değil de son ek şeklinde kullanılmış. Bu durumda 7. satırda önce a'nın değeri i'ye atanır, sonra a'nın değeri 1 artırılır. Halbuki ilk örneğimizde önce a'nın değeri 1 artırılmış sonra bu değer i'ye atanmıştı. İsterseniz şimdi bu ++ ve -- operatörlerini bir örnekte hep beraber görelim;

```
using System;
class Operatorler
{
	static void Main()
	{
		int a=5;
		int i=a++;
		int b=i--;
		int c=10;
		int d=--c;
		Console.Write("{0}\n{1}\n{2}\n{3}",a,i,b,d);
	}
}
```

Bu program ekrana alt alta 6, 4, 5 ve 9 yazacaktır. Dikkat ettiyseniz `Write` metodunu farklı şekilde kullandık. Siz de böyle çok fazla değişkeni daha kolay yazdırabilirsiniz. Ayrıca bu yöntemi `WriteLine` metodunda da kullanabilirsiniz.
NOT: Bir artırma ve bir azaltma operatörleri byte, sbyte, short ve ushort türlerindeki değişkenlerin türünü değiştirmez.

Karşılaştırma operatörleri

Daha önceki derslerimizden birinde 2<5 gibi bir ifadenin aslında bool türünden bir değer olduğunu söylemiştik. İşte bu operatörler bu bool türünden değerleri operandları vasıtasıyla üretmek için kullanılır. İki sayının birbirine göre büyüklüğünü ve küçüklüğünü kontrol ederler. Tabii ki yalnızca sayısal türdeki (char da dâhil) değişken ya da sabitlerle kullanılabilirler. Charla sayısal türler karşılaştırıldığında charın Unicode karşılığı hesaba katılır. Bu operatörler şunlardır: < (küçüktür), > (büyüktür), <= (küçük eşittir), >= (büyük eşittir), == (eşittir), != (eşit değildir). Örnekler:

```
using System;
class Operatorler
{
	static void Main()
	{
		bool a=4<6;
		bool b=6>5;
		bool c=7<=7;
		bool d=9>=12;
		bool e=10==12;
		bool f=1!=8;
		Console.Write("{0}\n{1}\n{2}\n{3}\n{4}\n{5}",a,b,c,d,e,f);
	}
}
```

Bu program ekrana alt alta True, True, True, False, False ve True yazacaktır.

as operatörü

Tüm değişken türlerinden objecte ve objectten referans değişkenlerine dönüşüm yapar. Dönüşüm gerçekleşmiyorsa null değer döndürür. Örnek:

```
using System;
class Operatorler
{
	static void Main()
	{
		object i="50";
		string s=i as string;
		Console.Write(s);
	}
}
```

veya

```
using System;
class Operatorler
{
	static void Main()
	{
		int i=50;
		object s=i as object;
		Console.Write(s);
	}
}
```

is operatörü

Verilen değişken, sabit ya da ifadenin türünü kontrol eder. Eğer söz konusu değişken, sabit ya da ifade verilen türle uyumluysa true değilse false değeri üretir. Eğer söz konusu değişken, sabit ya da ifadenin türü her zaman true ya da false üretiliyorsa derleyici uyarı verir, ancak bu uyarı derlemeye engel değildir. Kullanımına bir örnek:

```
using System;
class Operatorler
{
	static void Main()
	{
		int i=50;
		bool b1=i is int;
		bool b2=i is double;
		bool b3=12 is byte;
		bool b4='c' is string;
		bool b5=12f+7 is int;
		Console.Write("{0}\n{1}\n{2}\n{3}\n{4}",b1,b2,b3,b4,b5);
	}
}
```

Bu program alt alta True, False, False, False ve False yazacaktır. Dikkat ettiyseniz 12 sabitini byte saymadı ve 12f+7 sabitini de int saymadı. Çünkü C#, harfsiz tam sayı sabitleri int sayar. 12f+7 sabiti de floatlaşmıştır.

Mantıksal operatörler

Bu operatörler true veya false sabit ya da değişkenleri mantıksal ve, veya, değil işlemine sokarlar. Bunlar && (ve), || (veya) ve ! (değil) operatörleridir. Örnekler:

```
using System;
class Operatorler
{
	static void Main()
	{
		bool b1=35>10&&10==50; //false
		bool b2=35>10&&10!=50; //true
		bool b3=5 is int||12*3==200; //true
		bool b4=5<4||45/5==9; //true
		bool b5=!(5<4); //true
		Console.Write(b1+" "+b2+" "+b3+" "+b4+" "+b5);
	}
}
```

Eğer ! operatöründen sonra bir değişken ya da sabit yerine bir ifade geliyorsa ifadeyi parantez içine almak zorundayız.

Bitsel operatörler

Bitsel operatörler sayıların kendisi yerine sayıların bitleriyle ilgilenirler. Diğer bir deyişle sayıları ikilik sisteme dönüştürüp öyle işlem yaparlar. Bu operatörler yalnızca tam sayı sabit, değişken ya da ifadelerle kullanılabilirler. Eğer bitsel operatörler bool türünden değişken, sabit ya da ifadelerle kullanılırsa mantıksal operatörlerin gördüğü işin aynısını görürler. Örneğin aşağıdaki iki kullanım da birbiyle aynı sonucu doğuracaktır:

```
bool b1=false&&false;
bool b1=false&false;
```

~ (bitsel değil), & (bitsel ve), | (bitsel veya) ve ^ (bitsel özel veya) operatörleri bitsel operatörlerdir. Örnekler:

```
using System;
class Operatorler
{
       static void Main()
       {
              byte a=5&3; // 00000101&00000011 -->sonuç 00000001 çıkar. Yani 1
çıkar.
              byte b=5|3; // 00000101|00000011 -->sonuç 00000111 çıkar. Yani 7
çıkar.
              byte c=5^3; // 00000101^00000011 -->sonuç 00000110 çıkar. Yani 6
çıkar.
              byte d=5;
              byte e=(byte)~d; // 00000101'in değili -->sonuç 11111010 çıkar.
Yani 250 çıkar.
              byte f=(byte)~(a+b); // 00001000'ın değili -->sonuç 11110111
çıkar. Yani 247 çıkar.
              byte g=(byte)~(a+7); // 00001000'ın değili --->sonuç 11110111
çıkar. Yani 247 çıkar.
              Console.Write(a+" "+b+" "+c+" "+e+" "+f+" "+g);
       }
}
```

Bitsel operatörler sayıları ikilik sisteme dönüştürüp karşılıklı basamakları "ve", "veya" veya "özel veya" işlemine sokarlar ve sonucu operandlarıyla birlikte tutarlar. & (ve) işlemi karşılıklı basamaklardan her ikisi de 1'se ilgili basamağında 1 tutar, diğer durumlarda 0 tutar. | (veya) işlemi karşılıklı basamaklarından herhangi birisi 1'se ilgili basamağında 1 tutar, diğer durumlarda 0 tutar. ^ (özel veya) işlemi karşılıklı basamakları farklıysa ilgili basamağında 1 tutar, diğer durumlarda 0 tutar. ~ (değil) operatörü operandının her basamağını tersleştirip tutar. Bu operatörlerin operandları byte, sbyte, short, ushort olması durumunda sonuç intleşir.
NOT: Başında 0x veya 0X olan sayılar 16'lık düzende yazılmış demektir. Örnek:

```
using System;
class Operatorler
{
	static void Main()
	{
		int a=0xff;
		int b=0Xff;
		Console.Write(a+" "+b);
	}
}
```

Bu program ff sayısının onluk düzendeki karşılığı olan 255 sayısını ekrana iki kere yazacaktır.

Bitsel kaydırma operatörleri

Bu operatörler (<< ve >>) sayıların bitlerini istenildiği kadar kaydırmak için kullanılır. << sayıları sola kaydırırken, >> sağa kaydırır. Örnekler:

```
using System;
class Operatorler
{
	static void Main()
	{
		byte b=100; // 01100100
		byte c=(byte)(b<<1); //b bir kez sola kaydırıldı ve 11001000 oldu.
(200 oldu.)
		byte d=50; // 00110010
		byte e=(byte)(d>>2); // d iki kez sağa kaydırıldı ve  00001100
oldu. (12 oldu.)
		Console.Write(c+" "+e);
	}
}
```

<< ve >> operatörlerinin operandları değişken olabileceği gibi sabit ya da ifade de olabilir. Operand(lar) byte, sbyte, short, ushortsa sonuç intleşir.

Atama ve işlemli atama operatörleri

Şimdiye kadar çok kullandığımız = operatörüyle ilgili bilmeniz gereken iki şey var.

> Sol tarafta kesinlikle yalnızca bir tane değişken olmalı. Örneğin aşağıdaki gibi bir kullanım hatalıdır.

```
int a=0, b=0;
a+b=20;
```

> = operatöründe işlemler sağdan sola gerçekleşir ve = operatörü operandlarıyla birlikte atadığı değeri tutar. Örnek:

```
using System;
class Operatorler
{
	static void Main()
	{
		byte b=7, a=1, c;
		c=a=b;
		Console.Write(a+" "+b+" "+c);
	}
}
```

Bu program ekrana yan yana üç tane 7 yazacaktır. c=a=b; satırında önce b a'ya atanır ve a=b ifadesi 7 değerini tutar. Çünkü a=b atamasında 7 sayısı atanmıştır. Sonra a=b ifadesinin tuttuğu 7 sayısı c'ye atanır. Sonunda üç değişkenin değeri de 7 olur. Ancak şu örnek hatalıdır:

```
using System;
class Operatorler
{
	static void Main()
	{
		byte b=7, a=1, c;
		(c=a)=b;
		Console.Write(a+" "+b+" "+c);
	}
}
```

Burada önce a c'ye atanıyor ve c=a ifadesi 1 değeri tutuyor ve ifademiz 1=b ifadesine dönüşüyor. Atama operatörünün prensibine göre böyle bir ifade hatalı olduğu için program derlenmeyecektir. Bir de işlemli atama operatörleri vardır. Aslında bunlar = operatörünün yaptığı işi daha kısa kodla yaparlar:

- `a=a+b;` yerine `a+=b;`
- `a=a-b;` yerine `a-=b;`
- `a=a*b;` yerine `a*=b;`
- `a=a/b;` yerine `a/=b;`
- `a=a%b;` yerine `a%=b;`
- `a=a<<b;` yerine `a<<=b;`
- `a=a>>b;` yerine `a>>=b;`
- `a=a&b;` yerine `a&=b;`
- `a=a^b;` yerine `a^=b;`
- `a=a|b;` yerine `a|=b;`

kullanılabilir.

?: operatörü

?: operatörü C#'ta üç operand alan tek operatördür. Verilen koşula göre verilen değerlerden (object türünden) birini tutar. Kullanımı şöyledir:

```
koşul?doğruysa_değer:yanlışsa_değer
```

Örnek:

```
using System;
class Operatorler
{
	static void Main()
	{
		Console.Write("Vikikitap'ı seviyor musunuz? (e, h): ");
		string durum=Console.ReadLine();
		Console.Write(durum=="e"?"Teşekkürler!!":"Sağlık olsun...");
	}
}
```

Tabii ki verilen değerlerin illa ki sabit olma zorunluluğu yoktur, değişken ya da ifade de olabilir.

İşaret değiştirme operatörü (-)

- operatörü bir değişken, sabit ya da ifadenin işaretini değiştirmek için kullanılır. Örnekler:

```
using System;
class Operatorler
{
	static void Main()
	{
		int a=50;
		int b=-a;
		int c=-23;
		int d=-(a+b+c);
		int e=-12;
		Console.Write("{0}\n{1}\n{2}\n{3}\n{4}",a,b,c,d,e);
	}
}
```

Stringleri yan yana ekleme operatörü (+)

+ operatörü stringleri veya charları yan yana ekleyip operandlarıyla birlikte string olarak tutar. Ancak ifadedeki bütün bileşenlerin string olarak birleşebilmesi için ilk stringten önce charların yan yana gelmemesi gerekir. Yalnızca charlardan oluşan bir ifade stringleşmez.
NOT: + ve - operatörleri aynı zamanda sayısal (ve char) sabit, değişken ve ifadeleri toplamak ve çıkarmak için de kullanılır. Bunu daha önce görmüştük.

typeof operatörü

Herhangi bir değişken türünün CTS karşılığını Type türünden tutar. Örnek:

```
using System;
class Operatorler
{
	static void Main()
	{
		Type a=typeof(int);
		Console.Write(a);
	}
}
```

Bu program ekrana `System.Int32` yazar.