Bölüm 05

Veri Tipleri ve Değişkenler

C# Dilinde Veri Tipi nedir?

C# dilinde her sınıf bir veri tipidir, her veri tipi bir sınıftır.

Klâsik dillerde *tamsayılar, kesirli sayılar, karakterler, boolean değerler* gibi ilkel veri tipleri dile gömülü öğelerdir; herbiri bir anahtar sözcükle belirtilir. Oysa, Nesne Yönelimli Programlamada her sınıf soyut bir veri yapısıdır. O nedenle, C# dilinde klâsik dillerdeki gibi ilkel veri tipleri yoktur. Her veri tipi bir sınıftır. Ancak, programcının işini kolaylaştırmak için, .NET bu sınıfları ve sınıf öğelerini hazır tanımlamıştır. .NET Framework ortamını anlatırken *C++, C#, VB, J#* dillerinin temel veri tiplerinin .NET'in *CTS* (Common Type System) denilen veri tiplerine dönüştüğünü, böylece bu diller için ortak bir platform yaratılmış olduğunu söylemiştik. C# dilinin temel veri tiplerinin .NET 'deki karşılıklarını *Veri Tipleri ve Değişkenler* bölümünde ayrıntılı olarak ele alacağız. Bu sınıflar sanki klâsik dillerdeki ilkel veri tipleriymiş gibi kullanılabilirler. Aşağıda ayrıntıları verilecek olan *temel veri tiplerinin (built-in-types)* birer sınıf olduğunu, dolayısıyla klâsik dillerdeki ilkel veri tiplerine göre çok daha işlevsel olduklarını daima anımsamalıyız.

Neden Veri Tipi?

Bir programda farklı veri tipleriyle işlem yapmamız gerekebilir. Örneğin, tamsayılar, kesirli sayılar, karakterler (harfler ve klavyedeki diğer simgeler), metinler (string), mantıksal (boolean) değerler (doğru=true, yanlış=false) ilk aklımıza gelen farklı veri tipleridir. Bu farklı veri tiplerinin büyüklükleri (bellekte kaplayacakları yer) ve onlarla yapılabilecek işlemler birbirlerinden farklıdır. Örneğin, sayılarla dört işlem yapabiliriz, ama metinlerle yapamayız. O nedenle, C# ve başka bazı diller verileri tiplere ayırır. Değişken tanımlarken onun hangi tip veriyi tutacağını belirtir. Böylece, ana bellekte o değişkene yetecek bir yer ayırır ve o veri tipine uygun işlemlerin yapılmasına izin verir.

Değişken Nedir?

Teknik açıdan, değişken, ana bellekte belli bir veri tipine ait değerlerin girilebileceği bir adrestir.

Değişkenler programın ham veri tiplerini taşıyan araçlardır. C# dilinde global değişken yoktur. Her değişken bir sınıf içinde ya da sınıftaki bir blok içinde tanımlıdır. Sınıf içinde tanımlı olanlar tanımlandığı sınıfın bir öğesi (class member) olur. Sınıf içindeki bir blok içinde tanımlananlar o blokun yerel değişkeni olur.

Her değişkene, ana bellekte, o değişkenin tutacağı veri tipine yetecek büyüklükte bir yer ayrılır. Bu yere ana bellekte *değişkenin adresi* denir. Her değişkene bir ad verilir. Bu ad kullanılarak, değişkene değer atanabilir, atanan değer okunabilir ve atanan değer değiştirilebilir (güncellenebilir). Bir değişkene erişim demek, o değişkene değer atama, atanan değeri okuyabilme ve atanan değeri istendiğinde değiştirebilme yeteneklerine sahip olmak demektir.

C# *tip-korumalı* bir dildir. Başka bir deyişle, C# dili programda kullanılacak değişkenlerin tutacağı veri tiplerini kesin sınırlarla birbirlerinden ayırır. Bu yönüyle C, C++ ve Java'ya benzer. Dolayısıyla, bir değişken bildirimi yapılırken, o değişkenin veri tipi kesinlikle belirtilir. Daha sonra o değişkene atanan veriler, belirtilen veri tipinin dışına çıkamaz. Bunun bir istisnası, bazı durumlarda bir değişkenin tuttuğu verinin başka bir tipe dönüştürülmesidir. İngilizce'de casting denilen bu yöntemi ileride açıklayacağız.

C# dilinde değişken bildirimi aşağıdaki gibi yapılır.

```
int  faizOranı;
```

Bu deyim, `int` tipinden `faizOranı` adlı bir değişkenin bildirimidir. Bazı kaynaklarda buna değişken tanımı denilir. Biz bildirim sözcüğünü yeğlemekle birlikte, yerine göre her ikisini eş anlamlı kullanacağız. Bazı durumlarda, değişkene hemen bildirim anında bir ilk-değer vermek gerekebilir. Bu durumda,

```
int  faizOranı = 6 ;
```

yazılır. Bu bildirim int tipinden faizOranı adlı bir değişken bildirmekte ve ona ilk-değer olarak 6 tamsayısını atamaktadır. Bu söylediklerimizi genelleştirirsek, değişken bildirimi için aşağıdaki sözdizimini kullanacağız:

```
veri_tipi   değişken_adı ;
veri_tipi   değişken_adı = ilk_değer ;
```

biçimindedir. Tabii, bildirim anında ilk değer vermek zorunlu değil, isteğe bağlıdır.

Kural

Değişkene değer atarken, değişken ile değeri arasına (=) simgesi konulur. Bu nedenle (=) simgesine atama operatörü diyoruz.

C# küçük-büyük harf ayrımı yapar. Değişken adları rakamla başlayamaz ve ad içinde boşluk olamaz.

Değişken adını istediğiniz gibi koyabilirsiniz. Ama değişken adlarını küçük harfle, sınıf adlarını büyük harfle başlatmak, bir gelenektir. Bu geleneğe uyunuz. Ayrıca, programda tanımladığınız değişken, sabit, sınıf, metot gibi varlıkların adlarını, işlevlerini ima edecek biçimde koymak program okumayı kolaylaştıran iyi bir alışkanlık

olacaktır. Bazı durumlarda, adlar birden çok sözcükten oluşabilir. O zaman sözcükler arasına (_) simgesi koymak izlenen yöntemlerden birisidir. Ama C# dilinde (_) simgesini kullanmak yerine, deve-notasyonu denen yöntem tercih edilir. Bu stilde, farklı sözcüklerin ilk harfleri büyük yazılarak bitişik olarak yazılır. Örneğin, aşağıdakiler geçerli birer değişken adıdır.

```
faizOranı, sicilNo, öğrencininAdıVeSoyadı, sigortalınınDoğumTarihi
```

Aynı değişkenleri

```
faiz_oranı, sicil_no, öğrencinin_adı_ve_soyadı, sigortalının_doğum_tarihi
```

diye de yazabilirsiniz. Ama C# ilk yöntemi tercih etmektedir.

Aynı veri tipinden olan birden çok değişken bildirimini tek bir deyimle yapabiliriz:

```
int  m, n, r ;
```

Geçerlik Bölgesi

Her değişken tanımlı olduğu bölgede geçerlidir. Bunun anlamı şudur. Değişken sınıfın bir öğesi ise, bütün sınıf içinde geçerlidir. Ona sınıfın her metodu, her bloku erişebilir. Sınıfa ait nesne bellekte kaldığı sürece değişken kendi adresini korur.

Sınıftaki bir blokun içinde tanımlı olan değişken yalnız o blok içinde geçerlidir; ona tanımlı olduğu blok içinden ve o blokun alt bloklarından erişilebilir. Blokun işi bitince bellekteki adresi silinir. Örneğin bir for döngüsünün sayacı döngü bloku içinde geçerlidir. Döngü başlarken ona bellekte bir yer ayrılır, döngü bitince bellekten silinir. Benzer olarak, bir metot içinde tanımlı olan bir değişkene metot çağrılınca ana bellekte bir yer ayrılır, metodun işi bitince bellekten silinir.

Bu nedenle, bir metot, bir döngü veya {} parantezleriyle belirlenen bir blok içinde tanımlı değişkene *yerel değişken* diyoruz. Hiçbir blok içinde olmayıp sınıf içinde tanımlı değişkenlere de *sınıf değişkenleri* ya da *sınıf öğeleri* (class member) diyoruz.

İç-içe bloklar olduğunda, dış bloktaki değişkene iç bloktan erişilebilir. Ama, dış bloktan iç bloktaki değişkene erişilemez. Bir sınıf içinde tanımlı metot, döngü veya {} ile belirli bloklara göre, sınıfın kendisi de bir bloktur ve öteki blokların hepsini içine alır; yani en dıştaki bloktur. Dolayısıyla, sınıf öğelerine iç blokların her birinden erişilebiliyor olması doğaldır.

Değer Tipleri- Referans Tipleri

Birinci bölümde ana bellekteki stack ve heap bölgelerinden söz etmiştik. C# değişkenleri stack ve heap'te oluşlarına göre ikiye ayırır.

1. Değer tipleri
2. Referans tipleri

Değer tipi değişken, ana bellekteki adresine bir değer yazılan değişkendir. Bu değişkenler *stack*'ta tutulur. Referans tipi değişken ise, ana bellekteki bir nesneyi işaret eden (reference, pointer) değişkendir. İşaret edilen nesneler *heap*'te tutulur. Sınıflara ait bütün nesneler heap'te tutulur.

C# Dilinde sabit nedir?

Sabitler de değişkenler gibi veri tutan sınıf öğeleridir. Dolayısıyla her sabite, ana bellekte tutacağı veri tipine yetecek kadar bir yer ayrılır. Bunun değişkenden tek farkı şudur: Değişkenlere atanan değerler program çalışırken değiştirilebilir (güncelleme). Ancak, sabitlere atanan değerler değiştirilemez. Ayrıca, sabit bildirimi yapıldığı anda değeri atanmalıdır. Örneğin,

```
const  int  yıllıkFaizOranı = 8 ;
const  float  amortismanPayı = 12.50;
```

Genel söz dizimi şöyledir:

```
const  veri_tipi  sabitin_adı = atanan_değer ;
```

Sabit kullanmadan, sabitin işleme girdiği her yerde onun değerini koyarak program yazmak mümkündür. Örneğin, yukarıda bildirimi yapılan yıllıkFaizOranı bir bankanın mudilerinin alacağı faizleri hesaplayan programda kullanılıyor olsun. Yıllık faizin hesaplandığı her işlemde yıllıkFaizOranı yerine 8 kullanılabilir ve program aynı sonucu verir. Ancak faiz oranını değiştirip 9 yapmak istediğinizde, kaynak programın bütününde 8 yerine 9 yazmak gerekecektir. Bu hem uzun zaman alıcı hem de bazıları unutulabileceği için, hataya açıktır. Onun yerine yıllıkFaizOranı adlı bir sabit kullanılırsa, kaynak programda onun değerini 9 ile değiştirmek, bütün programı güncellemeye denktir. Bunu yapmak büyük bir zaman kaybını önleyeceği gibi, programda hata oluşmasının da önüne geçmiş olacaktır.

Sayısal Değişken Tipleri

Bir çok işlemi tamsayılarla ve kesirli sayılarla yaparız. O nedenle, sayılar her programlama dilinde önemlidirler. Farklı dillerde yazılan derleyiciler sayıları alt sınıflarına ayırır. En basit derleyiciler bile, sayıları tamsayılar (integer) ve kesirli sayılar (floating numbers) diye ikiye ayırır. Bazan küçük sayılarla, Bazan da büyük sayılarla uğraşırız. Dolayısıyla, belleği etkin kullanabilmek için, bir çok dilde, sayılar, alt-gruplarına ayrılır. C# dili, bu yönde çok ayrıntılı bir gruplama yapmıştır.

Tamsayı Değişken Tipleri (Integer Variable Types)

C# dili tamsayıları bellekte kendilerine ayrılan büyüklüğe ve işaretli olup olmadıklarına göre gruplara ayırır. Aşağıdaki tabloda bu grupları, .NET karşılıklarını ve özeliklerini göreceksiniz.

C# Tipi	.NET Tipi	Uzunluk (Byte)	Değer Aralığı
byte	Byte	1 byte	0 - 255
sbyte	SByte	1 byte	-128 - 127
short	Int16	2 byte	-32 768 - 32 767
ushort	UInt16	2 byte	0 - 65 535
int	Int32	4 byte	-2 147 483 648 - 2 147 483 648
uint	UInt32	4 byte	0 - 4 294 967 295
long	Int64	8 byte	-10^{20} - 10^{20}
ulong	UInt64	8 byte	0 - 2×10^{20}

C# dilinde tamsayı veri tipleri

İpucu

C# dilinin veri tiplerinin .NET 'in veri tiplerine dönüştüğünü söylemiştik. Dolayısıyla, kaynak programlarımızda C# veri tipleri yerine .NET 'teki karşılığını kullanabiliriz. Örneğin,

```
short n ;
int m ;
long r ;
```

bildirimleri yerine, sırasıyla,

```
Int16 n;
Int32 m ;
Int64 r ;
```

bildirimlerini yapabiliriz. C# derleyicisi bunları denk deyimler olarak algılar.

Kesirli Sayı Değişken Tipleri (Floating Point Variables)

C# dilinde kesirli sayılar float ve double diye ikiye ayrılır. Aşağıdaki tabloda bu grupları ve özeliklerini göreceksiniz. Aksi istenmediğinde, C# kesirli sayıların double tipinden olduğunu varsayar (default tip).

C# Tipi	.NET Tipi	Uzunluk (Byte)	Değer Aralığı	Duyarlı Basamak Sayısı
float	Single	4 byte	$1.5 * 10^{-45}$ - $3.4 * 10^{38}$	6 - 7 basamak
double	Double	8 byte	$5.0 * 10^{-324}$ - $1.7 * 10^{308}$	15 - 16 basamak

C# dilinde kesirli sayı veri tipleri

Örnek

C# kesirli sayıları, aksi söylenmediği zaman `double` tipinden sayar. Bu öndeğer (default) durum istenmiyorsa, kesirli sayının sonuna `f` ya da `F` yazılır. Aşağıdaki örnek, `float` veri tipinin kullanılışını göstermektedir

VeriTipi01.cs

```
using System;
namespace Bölüm05
{
    class KarmaTipler01
    {
        public static void Main()
        {
            int x = 3;
            float y = 4.5f;
            short z = 5;
            Console.WriteLine("{0} * {1} / {2} = {3}", x,y,z,x * y/z);
        }
    }
}
```

Bu programın çıktısı:

```
3 * 4,5 / 5 = 2,7
```

Çıktıda kesir kısmını 3 haneli yazdırmak için `f3` biçemleyicisini kullanırız. (Sayısal çıktıları biçemlemeyi ileride ele alacağız.)

VeriTipi02.cs

```
using System;

namespace Bölüm05
{
    class KarmaTipler02
    {
        public static void Main()
        {
```

```
            int x = 4;
            float y = 2.5f;
            float z = 5.0f;
            Console.WriteLine("{0} * {1} / {2} = {3:f3}", x,y,z,x*y/z);
        }
    }
}
```

Çıktı

```
4 * 2,5 / 5 = 2,000
```

Decimal Veri Tipi

C# dili tamsayıları ve kesirli sayıları, yukarıdaki tablolarda görülen alt-gruplarına ayırmıştır. Bu gruplar belleğin optimal kullanımını sağlar. Ancak, her gruba özgü sınırlamalar vardır. Örneğin, özellikle tip dönüşümü (casting) istenmezse, iki tamsayının birbirine bölümünden çıkan bölümün kesir kısmı atılır, yalnızca tamsayı kısmı yazılır. Dolayısıyla, bölüm kesirli sayı olması gerekirken, tamsayı olur. Kesirli sayı tiplerinde ise, yuvarlamalar nedeniyle kayıplar olabilir. Bu iki sorunu çözmek için C# decimal veri tipini yaratmıştır.

C# Tipi	.NET Tipi	Uzunluk (Byte)	Değer Aralığı	Duyarlı Basamak Sayısı
decimal	Decimal	12 byte	$\pm 1.0 \times 10^{-28}$ - $\pm 7.9 \times 10^{28}$	28 - 29 basamak

C# dilinde decimal sayı veri tipi

Char Veri Tipi

Character sözcüğü, kaynak programda kullanılan bütün harf, rakam ve diğer simgelerin kümesidir. C# dili, karakterleri unicode diye adlandırılan ve uzunluğu 2 byte (16 bit) olan sistemle belirler. Bu nedenle 1 byte (8 bit) lık sistemlerde olduğu gibi yalnızca İngiliz alfabesini değil, Türkçe, Arapça, Çince, Japonca, Rusça, İbranice vb. bütün dillerin alfabelerindeki harfleri içine alır.

Burada şuna dikkat çekmeliyiz. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 rakamların her birisi sayısal değerlerinin dışında, ayrıca birer character olarak character veri tipinde yer alırlar. Zaten bütün sayıların ekrana ya da kağıda gönderilen görüntüleri 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 karakterleriyle temsil edilirler.

C# Tipi	.NET Tipi	Uzunluk (Byte)	Kapsam
char	Char	2 byte	Bütün unicode karakterleri

C# dilinde char veri tipi

C# dilinde character veri tipi, kısaca char diye yazılır. Bu tipli bir değişken bildirimi aşağıdakilere benzer olarak yapılır:

```
char  birHarf  ;
```

ya da bildirim anında ilk değeri verilmek istenirse

```
char  birHarf  = 'K' ;
```

İpucu

`char` değerler daima tek tırnak (‘ ‘) içinde yazılır. Örneğin,

```
'b' , '0' , '7' , '$' , '=' , '?' , '''
```

birer karakterdir, ama derleyici

```
b , 0 , 7 , $ , = , ? , '
```

simgelerini birer karakter olarak algılamaz.

Bütün dillerde olduğu gibi, C# dili de klavyede tek bir simgeyle temsil edilemeyen bazı karakterleri kullanır. Bunlar, özellikle, yazıcıya ya da ekrana çıktı alırken kullanılan karakterlerdir. Bazı harflerin önüne (\) konularak yazılırlar, ama derleyici onları birer karakter olarak algılar:

```
\n    New Line (yeni satıra geç)
\t    Tab (tab yaz)
\0    Null (boş)
\r    Carriage Return (satırbaşına git)
\\    Backslash (ters bölü çizgisi (\) yazar)
```

String Veri Tipi

`string` veri tipi, `String` sınıfının takma adıdır. Tek bir karakter yerine bir sözcük, bir tümce, bir paragraf ya da bir roman yazmak istediğimizde char veri tipi yeterli olmayacaktır. Modern dillerin çoğunda olduğu gibi, C# dili de metinleri içerecek bir veri tipi yaratmıştır. Bu tipe string denilir. string veri tipi (sınıfı), uzun ya da kısa her metni içerebilir. Tabii, burada metin derken, yalnızca alfabenin harf, rakam ve işaretleriyle yetinmiyor, char tiplerinden oluşabilecek her diziyi kastediyoruz.

String tipleri çift tırnak (" ") içine yazılırlar. char tipler tek tırnak (' ') içine yazıldığı için, C# derleyicisi

```
'b'
```

verisini char olarak algılar, ama

```
"b"
```

verisini string olarak algılar.

C# dilinde string tipli bir değişken bildirimi aşağıdakilere benzer olarak yapılır:

```
string birMetin  ;
```

ya da bildirim anında ilk değeri verilmek istenirse

```
string birMetin = "Merhaba, dünya!" ;
```

İpucu

string değerler daima çift tırnak (" ") içinde yazılır. Örneğin,

```
"b" , "0" , "Ankara" , "&%$*/?" , "3 + 2" , "123" , "-123"
```

birer string'dir. Ama, ama derleyici

```
b , 0 , Ankara , &%$*/? , 3 + 2 , 123 , -123
```

simgelerini birer string olarak algılamaz.

Uyarı

C# dilinde string değeri birden çok satıra yazılamaz. Metin satıra sığmadığı ya da satırlara bölmek gerektiğinde, yeni satıra geçmeyi sağlayan (\n) karakteri kullanılır. Örneğin,

```
string  birŞiir = "Neler yapmadık bu vatan için \n Kimimiz öldük, \n Kimimiz nutuk söyledik. \n O.V.Kanık";
```

Bunu ekrana yazdırmak için

```
System.Console.WriteLine (birŞiir);
```

metodunu kullanırsak, ekrana şu görüntü gelir:

Neler yapmadık bu vatan için

Kimimiz öldük,

Kimimiz nutuk söyledik.

O.V.Kanık

Boolean Veri Tipi

Matematiksel Mantığın kurucusu sayılan George Boole'un adına izafe edilen ve Boolean diye adlandırılan mantıksal veri tipi iki tanedir: `true (doğru), false (yanlış)`. Bütün programlama dillerinde boolean veri tipi çok önem taşır. Özellikle, program akışının yönlendirilmesinde ve döngü yapılarında olmazsa olmaz tiplerdir.

`bool` anahtar sözcüğü `System.Boolean` 'ın takma adıdır.

C# Tipi	.NET Tipi	Uzunluk (Byte)	Kapsam
bool	Boolean	1 byte	true , false

C# dilinde bool veri tipi

`bool` tipi değişken bildirimi, genel değişken bildirimi kuralına göre yapılır:

```
bool  değişken_adı;
```

ya da ilk değer atanmak istenirse

```
bool  değişken_adı = değişken_değeri;
```

yazılır. Bazı durumlarda true ya da false literal değerleri yerine, bir mantıksal (boolean) deyim de atanabilir. Örneğin,

```
bool  mezun;
bool  mezun   = true;
bool  emekli  = false ;
bool  bFormül = (x > 15 && x < 30);
```

VeriTipi03.cs

```
using System;

namespace Bölüm05
{
    using System;
    class test
    {
        public static void Main()
        {
            bool a = true;
            Console.WriteLine(a ? "evet" : "hayır");
        }
    }
}
```

Veri Tipi Dönüşümleri (Casting)

Veri tiplerini, bellekte kendilerine ayrılan yerin byte cinsinden uzunluğuna göre gruplamıştık. Bazan sayısal bir değişkenin değerini bir tipten başka bir tipe dönüştürmemiz gerekebilir. Bu eylem, veriyi bellekte bir adresten başka bir adrese taşımak demektir. Öyleyse, şöyle bir akıl yürütebiliriz. Daha küçük bir yere sığan bir veri daha büyük bir yere taşınabilir. Ama daha büyük bir yere sığan bir veri daha küçük bir yere taşınamaz. Taşınırsa, veri kaybı olabilir.

İstemsiz (implicit) Dönüşüm

Bunu bir örnekle açıklamak daha kolay olacaktır. Anımsayacağınız gibi tamsayı veri tiplerini sekiz alt gruba bölmüştük. Onlar arasında `int` tipi değişkenler ana bellekte 4 byte (32 bit), `long` tipi değişkenler ise ana bellekte 8 byte (64 bit) uzunluğunda bellek adreslerine sahiptirler.

```
int  aaa = 123;
long  bbb ;
```

bildirimlerini yapmış olalım. Sonra,

```
bbb = aaa ;
```

atamasını yaparsak, aaa 'nın bellekteki adresinde yer alan 123 sayısını bbb 'nin bellekteki adresine taşımış oluruz. Bu taşımayı 4 byte'lık bir yerden 8 byte'lık bir adrese yaptığımız için, veri kaybı söz konusu değildir. Dolayısıyla, yukarıdaki veri tipi dönüşümü her zaman geçerlidir. Derleyici hiç itiraz etmeden dönüşümü yapan kodları derler. Bu tür dönüşüme istemsiz (implicit) dönüşüm denir. Zorlayıcı bir istem (talep) olmadan, yazılan dönüşüm deyimine derleyici itirazsız uyar.

Ama yukarıdaki dönüşümün tersi yapılamaz; çünkü 8 byte'lık bir yere sığan sayıyı 4 byte'lık bir yere sığdıramazsınız. Bunu yapmak isterseniz, C# derleyicisi izin vermez ve o dönüşümü yapan kodu derlemez.

İstemli(explicit) Dönüşüm

Bazı durumlarda long tipinden tanımlanan bir değişkene atanacak değerlerin int tipinden asla büyük olmayacağı programcı tarafından biliniyorsa, derleyiciyi ikna ederek long tipi veriyi int tipine dönüştürebilir. Buna casting denilir. Bunun nasıl yapıldığını, yukarıdakine benzer bir örnekle açıklayalım:

```
int  ccc ;
long  ddd= 456 ;
```

bildirimlerini yapmış olalım. Sonra,

```
ccc = (int)ddd ;
```

atamasını yaparsak, ddd 'nin belekteki adresinde yer alan 456 sayısını ccc 'nin bellekteki adresine taşımış oluruz. Bu taşımayı 8 byte'lık bir yerde4 byte'lık bir adrese yaptığımız için, ccc = (int)ddd atama deyiminde ddd değişkeninin önüne (int) simgesini koyuyoruz. Bunu yapmakla, derleyiciyi büyük adresteki veriyi küçük adrese taşımaya ikna ediyoruz. Bu tür dönüşüme istemli (explicit) dönüşüm denir.

Uyarı

Bu tür dönüşümler ancak sayısal veriler için yapılabilir. string ya da bool tipleri arasında yapılamaz.

Örnekler

64 bitlik bir bellek alanı işgal eden bir `double` veri tipini 32 bitlik bir `int` veri tipine dönüştürelim.

```
int  n ;
double  x  = 12.73;
```

bildirimleri yapılmış olsun.

```
n = (int)x;
```

istemli dönüşümü, x değişkeninin değeri olan 12.73 sayısının kesir kısmını atar ve 12 tamsayı kısmını `n` değişkeninin bellekteki adresine aktarır. Dönüşümde veri kaybı vardır. O nedenle, bu tür dönüşümü derleyici kendiliğinden (istemsiz - implicit casting) yapmaz. Bu dönüşüm, *istemli bir dönüşümdür* (explicit casting). Programcının bu dönüşümü özellikle isteyerek ve veri kaybı olabileceğini bilerek yapması gerekir. Bu dönüşümden sonra `n` değişkeninin değeri

```
n = 12
```

olacaktır.

Yukarıdaki dönüşümde, `double` tipinden `int` tipine geçerken, kesir kısmının atılacağını tahmin etmek zor değildi. Ama, bilinçsizce yapılan dönüşüm, beklenmedik sonuçlar doğurabilir. Şu örneği inceleyelim.

```
short sss ;
int   nnn  = 32800;
```

bildiriminden sonra

```
short  sss    =  nnn;
```

istemsiz (implicit) ve

```
short  sss    = (short) nnn;
```

istemli (explicit) dönüşümlerini istediğimizde, derleyicinin yapacağı işlere bakalım. Önce istemsiz dönüşümü yazalım.

VeriTipi04.cs

```
using System;

namespace DenetimYapıları
{
    using System;
    class test
    {
        public static void Main()
        {
            short sss;
            int nnn = 32800;
            sss = nnn;
            Console.WriteLine(sss);
        }
    }
}
```

Derleyici bu programı derlemez ve şu hata iletisini verir.

```
Error 1 Cannot implicitly convert type 'int' to 'short'. An explicit conversion exists (are you missing a cast?) ...
```

Bu demektir ki, derleyici `int` tipinden `short` tipine *istemsiz* (implicit) dönüşüm yapamaz. Çünkü, `int` veri tipi `32` bitlik bir bellek alanı tutarken, `short` tipi 16 bitlik bir yer tutmaktadır.

Şimdi de derleyiciyi bu dönüşümü yapmaya zorlayalı. Bunun için, istemsiz dönüşümü, istemliye çevirmeliyiz:

VeriTipi05.cs

```
using System;

namespace DenetimYapıları
```

```
{
    using System;
    class test
    {
        public static void Main()
        {
            short sss;
            int nnn = 32800;
            sss = (short)nnn;
            Console.WriteLine(sss);
        }
    }
}
```

Çıktı

```
-32736
```

Tabii, 32800 sayısını bir başka değişkene ayen aktarmak istediğimizde, asıl sayı yerine -32736 sayının aktarılmış olması kabul edilemez bir hatadır. İşin kötüsü, programda sözdizimi hatası olmadığı için, program derlenmiş ve çalışmıştır. Böyle bir işi, diyelim ki, on binlerce satırlık büyük bir programda yapsaydık, hatanın farkına bile varamayabilirdik.

İpucu

Derleme veya koşma anında hata mesajı vermeyen bu tür yanlışlara mantıksal (semantik) yanlışlar diyoruz. Programlamada en tehlikeli yanlışlar bunlardır; çünkü farkına varılmayabilir. Derleme ve koşma anında oluşan hatalar, programcıyı yormaktan başka kimseye zarar veremez. Programcı onları düzeltmediği sürece program çalışmayacaktır. Ama mantıksal hata içeren programlardaki hatanın farkına varılmaksızın, koşmaya devam edebilirler.

Şimdi, böylesine ölümcül bir hatanın neden oluştuğunu araştıralım. Bu mantıksal hatayı yaratan nedeni kavrarsak, bir daha bu tür hatalara düşmeyiz.

Örneğimizde `nnn` ve `sss` değişkenlerinin her ikisi de tamsayıdır, ama `short` tipinden olan `sss` değişkeninin 16 bitlik adresine -32768 ile +32767 arasındaki tamsayılar sığar. Çünkü, 16 bitlik bellekte işaretli sayılar şöyle temsil edilir:

2-li gösterimdeki en yüksek bit (en soldaki bit) *işaret bit*'idir. İşaret biti **0** ise sayı pozitif, işaret biti **1** ise sayı negatiftir. Buna göre 16-bitlik bellek adresine sığacak *en büyük pozitif tamsayının* ikili sayıtlama dizgesindeki temsili şöyledir:

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Bunun 10-lu sistemdeki karşılığı

$$2^{14} + 2^{13} + 2^{12} + \cdots + 2^2 + 2^1 + 2^0 = \frac{1 - 2^{15}}{1 - 2} = 32767$$

olur.

Negatif sayılar bellekte *2 nin tümleyeni* denilen sistemle tutulur. Bu sistemde, işaret biti hariç, sayının öteki bitleri 0 ise 1, 1 ise 0 yapılır ve çıkan sayıya 1 eklenir. Negatif sayıların işaret biti daima 1 dir. Şimdi derleyiciyi,

```
short  sss    = (short) nnn;
```

istemli dönüşümüyle, 32800 sayısını `short` tipinden olan `sss` değişkeninin adresine yazmaya zorluyoruz. 32800 sayısının 32 bitlik bellek adresinde 2-li sayıtlama sistemine göre temsili şöyledir.

00000000 00000000 10000000 00100000

Bunu 16 bitlik adrese girmeye zorladığımızda, sağdaki 16 bit yerleşir ve dolayısıyla `sss` değişkeninin değeri

10000000 00100000

olur. İşaret biti 1 olduğu için, derleyici bunu negatif `short` veri tipi olarak algılayacaktır. Kural gereği böyle bir sayıyı *2-nin tümleyenine* dönüştürecektir. Bunu yaparken, en soldaki işaret bitine dokunmaz, öteki hanelerde 1 olanları 0 ve 0 olanları 1 yapar, sonra çıkan sayıya 1 ekler. 0 ları 1 ve 1 leri 0 yapınca, sayı

1**1**1111111 11011111

biçimini alacaktır. Bunu 1 ile toplarsak

1**1**1111111 11100000

olur. Bunun 10-lu sayıtlama sistemindeki karşılığı

$$-(2^{14} + 2^{13} + 2^{12} + \cdots + 2^6 + 2^5 + 0 + 0 + 0 + 0 + 0) = -32736$$

olur.

Buradan çıkaracağımız ders şudur. *İstemsiz (implicit)* dönüşümlerde sorun yoktur; yanlış dönüşüm yazarsak derleyici ona itiraz eder ve derlemez. Dolayısıyla program koşma anında farkında olmadığımız *mantıksal* (semantik) hatalar yapamaz. Ama *istemli (explicit)* dönüşüm derleyiciye zorla yaptırılan bir dönüşümdür. Programcı bilinçsizce istemli dönüşümler yaparsa, derleme hatası ve koşma hatası oluşmaz, ama program farkına varılamayan yanlış işlemler yapabilir. O nedenle, istemli dönüşüm yazarken, çıkacak sonucun ne olacağını biliyor ve o sonucu istiyor olmalıyız.

Alıştırma

Yukarıda söylenenleri gerçekten kavramış iseniz, aşağıdaki programı kolayca çözümler ve mantıksal hata yapıp yapmadığını bulabilirsiniz.

VeriTipi06.cs

```
using System;

class Uygulama
{
    static void Main(string[] args)
    {
        short i;
        ushort j;
        j = 60000;
        i = (short)j;
        Console.WriteLine(i);
    }
}
```