Chapter 4

Data Types in C#

Data Types in C#

Objectives

After completing this unit you will be able to:

- **Explain what *strongly typed* means, and why strongly typed languages contribute to program reliability.**
- **Explain when implicit conversion is used in C#, and when casting must be used.**
- **Name the types available in C#, and state the size of each type.**
- **Use C# types in simple programs.**

Strong Typing

- **C# is a *strongly typed* language.**
  - Variables and constants of different types may not be mixed, except according to strict rules.
  - The C# compiler will refuse to compile type conversions that are not 'safe'.
- **By contrast, C++ is a weakly typed language, and Visual Basic is untyped.**
- **Strong typing prevents certain types of errors**
  - Unintended mixing of types
  - Loss of precision

Demo: Typing in C#, VB and C++

- **C# is a strongly typed language.**
  - See **Seven\Charp**
- **C++ is a weakly typed language.**
  - See **Seven\Cpp**
- **VB6 can be programmed in an untyped manner.**
  - See **Seven\Vb**

C# Types

- **In C#, the size of each type is specified.**
  - In C and C++, the sizes of the types are not completely specified; only relative sizes are given.
  - C# is similar to Java in size specification.
- **C# has five categories of types.**
  - Integer Types
  - Floating-point Types
  - Decimal Type
  - Character Type
  - Boolean Type

Integer Types

- **C# has a large variety of integer data types.**
  - Allows great flexibility to choose appropriate size and signed/unsigned for you application.

C# Keyword	Size	Signed/ Unsigned	Type in **System** Namespace
sbyte	8 bits	signed	**SByte**
byte	8 bits	unsigned	**Byte**
short	16 bits	signed	**Int16**
ushort	16 bits	unsigned	**UInt16**
int	32 bits	signed	**Int32**
uint	32 bits	unsigned	**UInt32**
long	64 bits	signed	**Int64**
ulong	64 bits	unsigned	**UInt64**

Integer Type Range

- **Use the *MinValue* and *MaxValue* members of the types in the *System* namespace to find the range of each integer type.**
  - Example:

```
Console.WriteLine("min int= " + UInt32.MinValue);
Console.WriteLine("max int= " + UInt32.MaxValue);
```

  - Output:

```
min int = -2147483648
max int = 2147483647
```

  - See the program **IntegerRange**

Integer Literals

- **A literal is a source code representation of a value.**

```
squaresChess = 64;
```

  - In this example, '64' is an integer literal.

- **Integer literals are always stored as either 32-bit or 64-bit values.**
  - In the above example, the value '64' is stored as a 32-bit number.

- **You may specify the type that is used with a suffix.**
  - L (or l) for long
  - U (or u) for unsigned
  - Suffixes may be combined: LU, UL, ul, lu (any combination of case and order)

- **Hexadecimal representation uses the prefix 0x (or 0X).**

Floating Point Types

- **Used to express very large and very small quantities.**
- **There are two floating point types in C#, corresponding to single and double precision.**

C# Keyword	Size	Type in **System** Namespace
float	32 bits	**Single**
double	64 bits	**Double**

- **There is actually a third floating point type, called *decimal*, which is covered later in this module.**
- **Use the *MinValue* and *MaxValue* members of the types in the *System* namespace to find the range of each floating point type.**
  - See Lab 4

Floating Point Literals

- **Floating point literals may use either decimal or exponential notation.**
  - Decimal notation example: 3.141529
  - Exponential notation example: 2e-9 (0.000000002)
  - Mixed notation: 3.01e3 (3,010)
- **The default type of a floating point literal is double.**
  - To specify float, use a suffix of **F** or **f**.
  - A suffix of **D** or **d** can be used to signify double (even though that is redundant, you might want to emphasize it for some reason).

IEEE Standard for Floating Point

- **C# uses the IEEE 754 floating point standard (see http://grouper.ieee.org/groups/754/).**
  - IEEE 754-1985 governs binary floating-point arithmetic. It specifies number formats, basic operations, conversions, and exceptional conditions. The related standard IEEE 854-1987 generalizes 754 to cover decimal arithmetic as well as binary.
- **IEEE 754 defines two special values, NaN, and Infinity**
  - Infinity results from an attempt to divide a non-zero number by a floating point zero.
  - NaN results from an attempt to divide a floating point zero by any number, including floating point zero.
- **Example program**
  - See **SpecialFloat**

Decimal Type

- **Floating point types (even double) are frequently not sufficiently precise for financial calculations.**
  - Even a small round-off error may be unacceptable.
- **To address this problem, a new type was introduced which can precisely represent decimal numbers with up to 28 digits.**

C# Keyword	Size	Type in **System** Namespace
decimal	96 bits	**Decimal**

- **As with integer and floating point types, the range may be found using the *MinValue* and *MaxValue* member functions of *Decimal*.**
- **This type is new for C#.**
  - C++ and Java require library support to get the equivalent of decimal.

Decimal Literals

- **The decimal literal suffix is M**
  - For "Money".
  - The M suffix may be combined with exponential and decimal notation.

```
decimal billGatesSalary = 4.0e6M;
```

- **Example program:**
  - See **DecimalLiterals**

Character Type

- **C# uses a 16-bit Unicode character set.**

C# Keyword	Size	Type in **System** Namespace
char	16 bits	**Char**

Character Literals

- **Character in single quotes**
  - '7' // Unicode character '7'
- **Hex encoding (\x)**
  - \x0055 // '7' in hex
- **Unicode prefix (\u)**
  - \u0055 // '7' using Unicode prefix
- **To represent characters with a special meaning, use a backslash (\) to 'escape'.**
  - '\"' // the single quote character
  - '\\' // the backslash character
- **Example program:**
  - See **CharacterLiterals**

Escape Characters

- **There are a number of non-printing or special characters used by C# which have been given special escape sequences.**

Escape Character	Name	Value
\’	Single quote	0x0027
\”	Double quote	0x0022
\\	Backslash	0x005C
\0	Null	0x0000
\a	Alert	0x0007
\b	Backspace	0x0008
\f	Form feed	0x000C
\n	New line	0x000A
\r	Carriage return	0x000D
\t	Horizontal tab	0x0009
\v	Vertical tab	0x000B

Boolean Type

- **The bool type represents the logical values 'true' and 'false'.**
  - **`true`** and **`false`** are the two Boolean literals.

C# Keyword	Size	Type in **System** Namespace
bool	8 bits	**Boolean**

Implicit Conversions

- **Since C# is *strongly typed*, if we need to use a value of one data type where another type is expected, we must use a *conversion*.**
- **An *implicit* conversion is done silently by the compiler where needed.**
  - Implicit conversions are only done for *safe* conversions, where the target type has a larger dynamic range (wider) than the type to be converted.
- **Examples:**
  - **float** to **double**
  - **byte** to **int**

Explicit Conversions

- **An *explicit* conversion is specified by the programmer, using a *cast*.**
  - An explicit conversion is generally required if the target type has a smaller dynamic range (narrower).
  - Example:

```
float pi = 3.141529;  // compiler error;
                      // default is double
float pi = (double) 3.141529; // ok
```

  - An explicit conversion is needed if the dynamic range of the target doesn't include all of the values possible for the type to be converted.
  - Example:

```
char seven = '7';
short number;
number = seven;  // compiler error
// character range 0 to 65535
// short range -32768 to 32767
```

- **The *bool* type may not be cast to or from any other type.**
  - However, there are conversion functions available in a special class (called **Convert**) for this purpose.

Conversions Example

- **Example program illustrates a number of issues in implicit and explicit conversion.**

```
// Conversions.cs

using System;

public class Conversions
{
    public static int Main(string[] args)
    {
        // float pi = 3.14;       // compiler error
        float pi = (float) 3.14;
        Console.WriteLine("pi = " + pi);
        // short seven = '7';     // compiler error
        short seven = (short) '7';         // cast
        ushort useven = '7';               // ok
        Console.WriteLine("seven = " + seven);
        Console.WriteLine("useven = " + useven);
        //int itrue = (int) true;    // cast fails
        int itrue = Convert.ToInt32(true);
        int ifalse = Convert.ToInt32(false);
        Console.WriteLine("itrue = " + itrue);
        Console.WriteLine("ifalse = " + ifalse);
        return 0;
    }
}
```

Summary

- **C# is strongly typed**
- **Types in C# have specified sizes**
- **Some conversions are safe, and can be implicitly performed by the compiler.**
- **Some conversions are potentially unsafe, but may be explicitly specified by the programmer, using casts**
- **Boolean conversions to or from other types are only permitted using member conversion functions**