Chapter 2

First C# Programs

First C# Programs

Objectives

After completing this unit you will be able to:

- **Write a basic "Hello, World" program in C#.**
- **Compile and run C# programs in your local development environment.**
- **Describe the basic structure of C# programs.**
- **Describe how related C# classes can be grouped into namespaces.**
- **Use variables and simple expressions in C# programs.**
- **Write C# programs that can perform simple calculations.**
- **Perform simple input and output in C#.**
- **Describe objects and classes in C#.**
- **Use an input wrapper class to perform input in C#.**

Hello, World

- **Whenever learning a new programming language, a good first step is to write and run a simple program that will display a single line of text.**
  - Such a program demonstrates the basic structure of the language, including output.
  - You must learn the pragmatics of compiling and running the program.
- **Here is "Hello, World" in C#.**
  - See **Demos\Hello\Hello.cs**, backed up in **Hello**.

```
// Hello.cs

class Hello
{
    public static int Main(string[] args)
    {
        System.Console.WriteLine("Hello, World");
        return 0;
    }
}
```

Compiling, Running (Command Line)

- **See Appendix B for information on the Microsoft Visual Studio.NET IDE (integrated development environment).**
- **If you are using the .NET SDK, you may do the following:**
  - Compile the program via the command line:

```
csc Hello.cs
```

  - An executable file **Hello.exe** will be generated. To execute your program, type at the command line:

```
Hello
```

  - The program will now execute, and you should see displayed the greeting. That's all there is to it!

```
Hello, World
```

Program Structure

```
// Hello.cs

class Hello
{
...
}
```

- **Every C# program has at least one *class*.**
  - A class is the foundation of C#'s support for object-oriented programming.
  - A class encapsulates data (represented by **variables**) and behavior (represented by **methods**).
  - All of the code defining the class (its variables and methods) will be contained between the curly braces.
  - We will discuss classes in detail later.
- **Note the *comment* at the beginning of the program.**
  - A line beginning with a double slash is present only for documentation purposes and is ignored by the compiler.
- **C# files have the extension *.cs***

Program Structure (Cont'd)

```
// Hello.cs

class Hello
{
    public static int Main(string[] args)
    {
        ...
        return 0;
    }
}
```

- **Every C# program has a distinguished class that has a method whose name must be *Main*.**
  - Note the capitalization!
  - The method should be **public** and **static**.
  - An **int** exit code can be returned to the operating system. Use **void** if you do not return an exit code.

```
public static void Main(string[] args)
```

  - Command line arguments are passed as an array of strings.
  - The runtime will call this **Main** method – it is the entry point for the program.
  - All the code of the **Main** method will be between the curly braces.

Program Structure (Cont'd)

```
// Hello.cs

class Hello
{
    public static int Main(string[] args)
    {
        System.Console.WriteLine("Hello, World");
        return 0;
    }
}
```

- **Every method in C# has one or more *statements*.**
- **A statement is terminated by a semicolon.**
  - A statement may be spread out over several lines.
- **The *Console* class provides support for standard output and standard input.**
  - The method **WriteLine** displays a string, followed by a new line.

Namespaces

- **Much standard functionality in C# is provided through many classes in the .NET Framework.**
- **Related classes are grouped into *namespaces*.**
- **The fully qualified name of a class is specified by the namespace followed by a dot followed by class name.**

```
System.Console
```

- **A *using* statement allows a class to be referred to by its class name alone.**

```
// Hello2.cs

using System;

class Hello
{
    public static int Main(string[] args)
    {
        Console.WriteLine("Hello, World");
        return 0;
    }
}
```

- **Note that in C# it is not necessary for the file name to be the same as the name of the class containing the *Main* method.**

Exercise

- **Take a few minutes to add two more lines of code to your program.**
  - Print out the phrase "My name is XXXX" (use your own name).
  - Then print out "Goodbye".
- **Save your file, compile the program, and run it.**
  - Your output should be something like this:

```
Hello, World
My name is Bob
Goodbye
```

Answer

```
// Hello3.cs

using System;

class Hello
{
    public static int Main(string[] args)
    {
        Console.WriteLine("Hello, World");
        Console.WriteLine("My name is Bob");
        Console.WriteLine("Goodbye");
        return 0;
    }
}
```

Variables

- **In C# you can define *variables* to hold data.**
- **Variables represent storage locations in memory.**
- **In C# variables are of a specific data *type*.**
  - Some common types are **int** for integers and **double** for floating point numbers.
  - You must declare variables before you can use them.
- **A variable declaration reserves memory space for the variable and may optionally specify an initial value.**

```
int kilo = 1024;      // reserves space and assigns
                      // an intial value
int mega;             // reserves space but does
                      // not initialize
```

  - If an initial value is not specified, C# initializes the variable to a default value, such as 0.

Expressions

- **You can combine variables and constants (or "literals") via *operators* to form *expressions*.**
- **Examples of operators include the standard arithmetic operators:**

```
+       addition
-       subtraction
*       multiplication
/       division
```

- **Here are some examples of expressions:**

```
kilo * 1024
(fahrenheit – 32) * 5 / 9
3.1416 * radius * radius
```

Assignment

- **You can assign a value to a variable by using the = symbol.**
  - On the left hand side is a variable.
  - On the right hand side is an expression.
  - The expression is evaluated and its value is assigned to the variable on the left.
  - Assignment is a statement and must be terminated by a semicolon.

```
mega = kilo * 1024;
celsius = (fahrenheit – 32) * 5 / 9;
area = 3.1416 * radius * radius;
```

- **Note that the same variable can be used on both sides of an assignment statement.**

```
int item = 5;
int total = 30;
total = total + item;
```

  - The expression **total + item** evaluates to 35, using the old value of **total**, and this value is assigned to **total**, creating a new value.

Calculations Using C#

- **You can easily use C# to perform calculations by adding code to the *Main* method of a C# class.**
  - Declare whatever variables you need.
  - Create expressions and assign values to your variables.
  - Print out the answer using **Console.WriteLine()**.
- **You can easily do labeled output relying on two features of C#:**
  - The operator + performs concatenation for **string** data.
  - There is an automatic, implicit conversion available that converts numeric data to string data when required.
  - Hence this code ...

```
int total = 35;
System.Console.WriteLine("The total is " + total);
```

  - ... will produce this output:

```
The total is 35
```

Sample Program

- **This program will convert from Fahrenheit to Celsius.**
  - See **Convert\Step1**.

```
// Convert.cs - Step 1
//
// Program converts a hardcoded temperature in
// Fahrenheit to Celsius

using System;

class Convert
{
    public static void Main(string[] args)
    {
        int fahr = 86;
        int celsius = (fahr - 32) * 5 / 9;
        Console.WriteLine("fahrenheit = " + fahr);
        Console.WriteLine("celsius = " + celsius);
    }
}
```

Input in C#

- **Our first Convert program is not too useful because the Fahrenheit temperature is hardcoded.**
  - To convert a different temperature you would have to edit the source file and recompile
- **What we really want to do is allow the user of the program at runtime to enter a value for the Fahrenheit temperature.**
- **Although simple console input in C# is fairly easy, we can make it even easier using object oriented programming.**
  - We can encapsulate or "wrap" the details of input in a class.
  - It will be easy to use the wrapper class.

More About Classes

- **Although we will discuss classes in more detail later, there is a little more you need to know now.**
- **A class can be thought of as a template for creating objects.**
  - An **object** is an instance of a **class**.
- **A class specifies data and behavior.**
  - The data is different for each object instance.
- **In C# you instantiate a class by using the *new* keyword.**

```
InputWrapper iw = new InputWrapper();
```

  - This code creates the object instance **iw** of the **InputWrapper** class.

InputWrapper Class

- **The *InputWrapper* class "wraps" interactive input for several basic data types.**
  - The supported data types are **int**, **double**, **decimal** and **string**.
  - Methods **getInt**, **getDouble**, **getDecimal** and **getString** are provided.
  - A prompt string is passed as an input parameter.
  - See the files **InputWrapper.cs** in directory **InputWrapper** which implements the class and **TestInputWrapper.cs** which tests the class.
- **Although the code is quite short, it is a little complex, involving a number of different methods of different .NET Framework classes**
- **But you do not need to be familiar with the implementation of *InputWrapper* in order to use it.**
  - That is the beauty of "encapsulation" – complex functionality can be hidden by an easy to use interface.

Echo Program

- **We illustrate interactive input by a simple "echo" program.**
  - Program prompts user for name, and then prints out a personalized greeting.
  - See **Echo**.
- **This directory has two files, each defining a class.**
  - **InputWrapper.cs** defines the wrapper class. There is no **Main** method in this class.
  - **EchoName.cs** has a class **Echo** with a **Main** method.

```
// EchoName.cs
//
// Prompts user to enter name and then
// prints out greeting using name

using System;

class Echo
{
 public static void Main(string[] args)
 {
   InputWrapper iw = new InputWrapper();
   string name = iw.getString("Enter your name: ");
   Console.WriteLine("Hello, " + name);
 }
}
```

Using InputWrapper

- **The shaded statements illustrate how to use the *InputWrapper* class.**
  - Instantiate an **InputWrapper** object **iw** by using **new**.
  - Prompt for an obtain input data by calling the appropriate **getXXX** method.

Compiling Multiple Files

- **It is easy to compile multiple files at the command line.**

```
csc *.cs
```

  - This will compile all the files in the current directory.
  - The file containing a class with the **Main** method will be used as the name of the generated EXE file:

```
Directory of C:\OI\CSharp\Chap2\Echo

01/05/2001  12:20p    <DIR>          .
01/05/2001  12:20p    <DIR>          ..
01/05/2001  12:02p                334 EchoName.cs
01/05/2001  12:03p              3,584 EchoName.exe
01/05/2001  11:36a                855 InputWrapper.cs
```

  - If multiple classes contain a **Main** method, you can use the **/main** command line option to specify which class contains the **Main** method that you want to use as the entry point into the program.

```
csc *.cs /main:Echo
```

The .NET Framework

- **The .NET Framework has a very large class library (over 2500 classes).**
- **To make all this functionality more manageable, the classes are partitioned into *namespaces*.**
- **The root namespace is *System*, which directly contains many useful classes, among them:**
  - **Console** provides access to standard input, output and error streams for doing I/O.
  - **Convert** provides conversions among base data types.
  - **Math** provides mathematical constants and functions.

The .NET Framework (Cont'd)

- **Underneath *System* there are other namespaces, among them:**
  - **System.Data** contains classes constituting the ADO.NET architecture for accessing databases.
  - **System.Xml** provides standards based support for processing XML.
  - **System.Drawing** contains classes providing GDI+ graphics functionality.
  - **System.WinForms** provides support for creating applications with rich Windows based interfaces.
  - **System.Web** provides support for browser/server communication.
  - **System.IO** provides support for reading and writing with streams and files. Both synchronous and asynchronous IO are supported.
  - **System.Net** provides support for several standard network protocols.

Summary

- **Every C# application has a class with a method *Main*, which is the entry point into the application.**
- **The *System* class includes methods for doing output, such as *WriteLine*.**
- **Expressions in C# are formed from constants, variables and operators.**
- **With the assignment statement you can assign a value computed by an expression to a variable.**
- **Input in C# is a little more complicated than output, but you can use a wrapper class that encapsulates the required C# classes and presents a simple programming interface.**
- **The .NET Framework has a large class library that is partitioned into namespaces.**