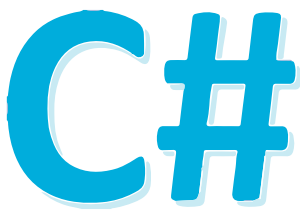
Contents

1 Getting started

Welcome to the exciting world of C# programming. This chapter introduces the Visual Studio Integrated Development Environment and shows you how to create a real Windows application.

Introducing C#



The introduction of the Microsoft .NET framework at the Professional Developers Conference in July 2000 also saw Microsoft introduce a new programming language called C# (pronounced "see-sharp"). The name was inspired by musical notation where a # sharp symbol indicates that a written note should be a semitone higher in pitch. This notion is similar to the naming of the C++ programming language where the ++ symbol indicates that a written value should be incremented by 1.

- C# is designed to be a simple, modern, general-purpose, object-oriented programming language, borrowing key concepts from several other languages – most notably the Java programming language. Consequently, everything in C# is a class "object" with "properties" and "methods" that can be employed by a program.
- C# is an elegant and "type-safe" programming language that enables developers to build a variety of secure and robust applications. You can use C# to create Windows client applications, XML web services, distributed components, client-server applications, database applications, and much, much more.
- C# is specifically designed to utilize the proven functionality built into the .NET framework "class libraries". Windows applications written in C# therefore require the Microsoft .NET framework to be installed on the computer running the application – typically, an integral component of the system.

The Microsoft .NET Framework

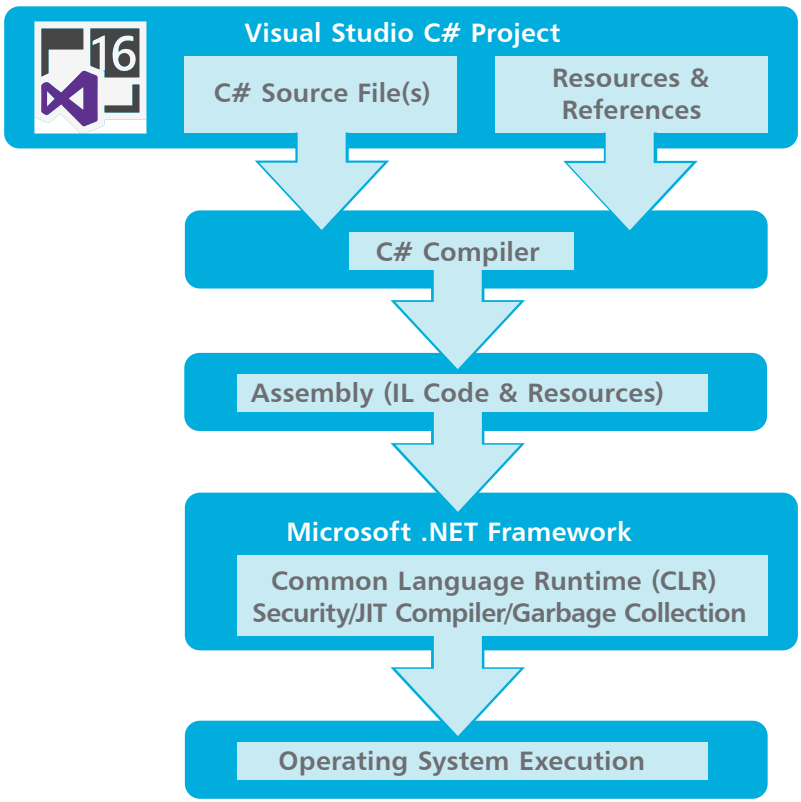
Each version of the Microsoft .NET framework includes a unified set of class libraries and a virtual execution system called the Common Language Runtime (CLR). The CLR allows the C# language and the class libraries to work together seamlessly.

To create an executable program, source code written in the C# language is compiled by the C# Compiler into Intermediate Language (IL) code. This is stored on disk, together with other program resources such as images, in an "assembly". Typically, the assembly will have a file extension of .exe or .dll. Each assembly contains a "manifest" that provides information about that program's security requirements.

The source code of all examples in this book is available for free download at www.ineasysteps.com/resource-center/downloads

If you don't achieve the result illustrated in any example, simply compare your code to that in the original example files you have downloaded to discover where you went wrong.

When a C# program is executed, the assembly is loaded into the Common Language Runtime (CLR), and the security requirements specified in its assembly manifest are examined. When the security requirements are satisfied, the CLR performs Just-In-Time (JIT) compilation of the IL code into native machine instructions. The CLR then performs "garbage collection", exception handling, and resource management tasks before calling upon the operating system to execute the program:



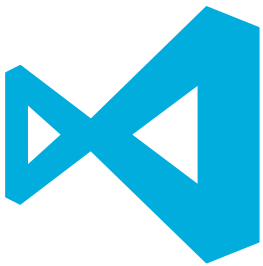
Just-In-Time compilation is also known as "Dynamic Translation".

Just-In-Time compilation occurs during program execution, rather than prior to its execution.

As language interoperability is a key feature of the Microsoft .NET framework, the IL code generated by the C# Compiler can interact with code generated by the .NET versions of other languages such as Visual Basic and Visual C++. The examples throughout this book demonstrate Visual C# program code.

Installing Visual Studio

In order to create Windows applications with the C# programming language, you will first need to install a Visual Studio Integrated Development Environment (IDE).



Microsoft Visual Studio is the professional development tool that provides a fully Integrated Development Environment for Visual Basic, Visual C++, Visual J#, and Visual C#. Within its IDE, code can be written in Visual Basic, C++, J#, or the C# programming language to create Windows applications.

Visual Studio Community edition is a streamlined version of Visual Studio, specially created for those people learning programming. It has a simplified user interface and omits advanced features of the professional edition to avoid confusion. C# code can be written within the **Code Editor** of either version of the Visual Studio IDE to create Windows applications.

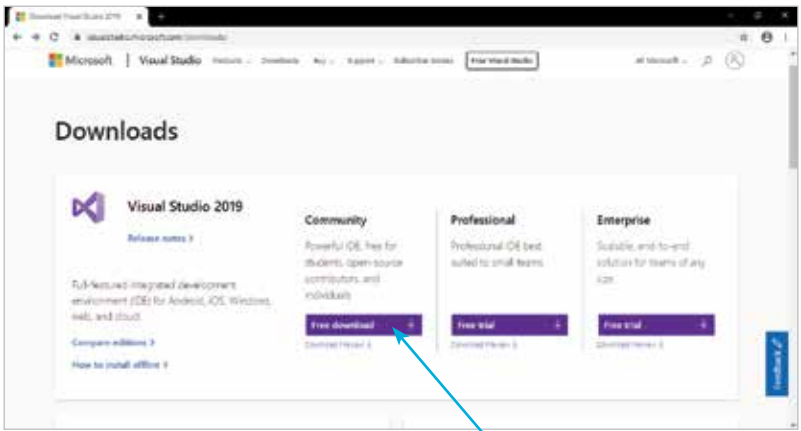
Both Visual Studio and Visual Studio Community provide an IDE for C# programming but, unlike the fully-featured Visual Studio product, the Visual Studio Community edition is completely free and can be installed on any system meeting the following minimum requirements:

The **New** icon pictured above indicates a new or enhanced feature introduced with the latest version of C# and Visual Studio.

Component	Requirement
Operating system	Windows 10 (version 1703 or higher) Windows Server 2016 or 2019 Windows 8.1 (with update 2919355) Windows 7 Service Pack 1 Windows Server 2012 R2
CPU (processor)	1.8 GHz or faster
RAM (memory)	2 GB (8 GB recommended)
HDD (hard drive)	Up to 210 GB available space
Video Card	Minimum resolution of 1280 x 720 Optimum resolution of 1366 x 768

The Visual Studio Community edition is used throughout this book to demonstrate programming with the C# language, but the examples can also be recreated in Visual Studio. Follow the steps opposite to install the Visual Studio Community edition.

1 Open your web browser and navigate to the Visual Studio Community download page – at the time of writing this can be found at **visualstudio.microsoft.com/downloads**



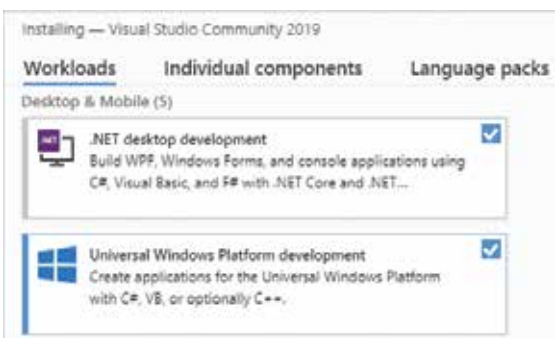
Choosing a different destination folder may require other paths to be adjusted later – it's simpler to just accept the suggested default.

2 Click the button in the Community edition section to download a **vs_community** installer file

3 Click on the  **vs_community** file to run the installer

4 Accept the suggested installation location, then click **Next**

5 Choose the two C# options below for installation



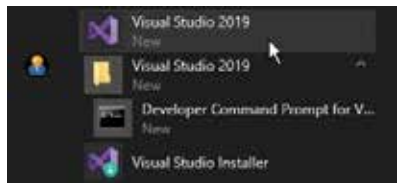
You can re-run the installer at a later date to add or remove features.

6 Click the **Install** button at the bottom-right of the installer to begin the download and installation process

Exploring the IDE

The first time Visual Studio starts it takes a few minutes as it performs configuration routines.

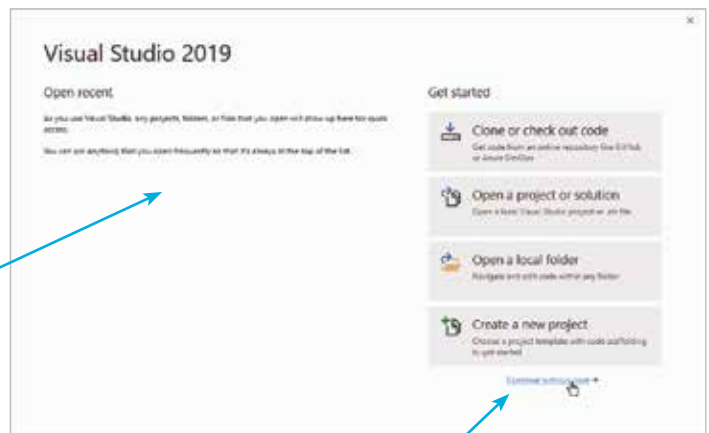
1 Go to your apps menu, then select the Visual Studio 2019 menu item added there by the installer:



2 Sign in with your Microsoft account, or register an account then sign in, to continue

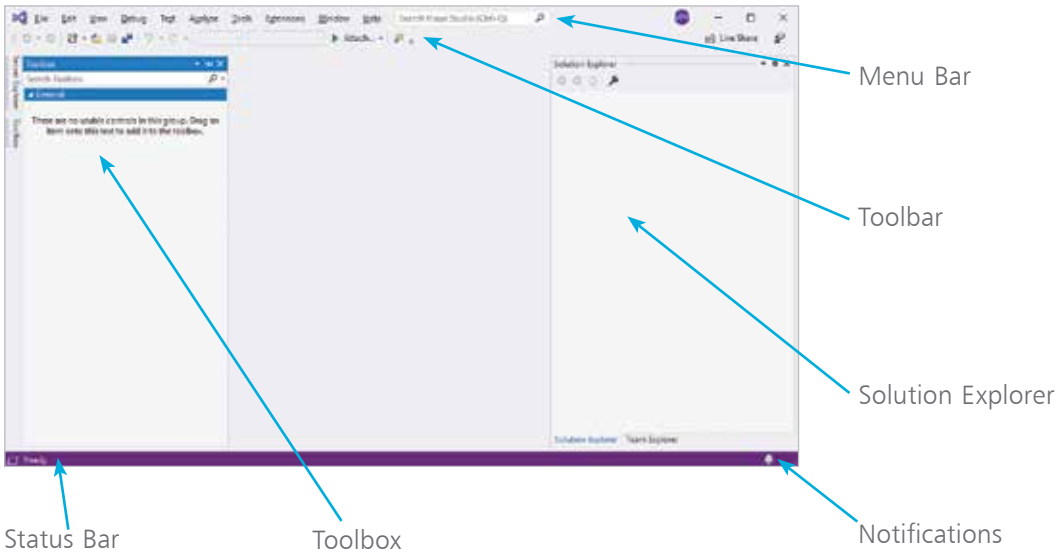
3 See a default **Start Page** appear where recent projects will be listed alongside several "Get started" options

In the future your recent projects will be listed here so you can easily reopen them.



4 For now, just click the link to **Continue without code** to launch the Visual Studio application

The Visual Studio Integrated Development Environment (IDE) appears, from which you have instant access to everything needed to produce complete Windows applications – from here you can create exciting visual interfaces, enter code, compile and execute applications, debug errors, and much more.



Visual Studio IDE components

The Visual Studio IDE initially provides these standard features:

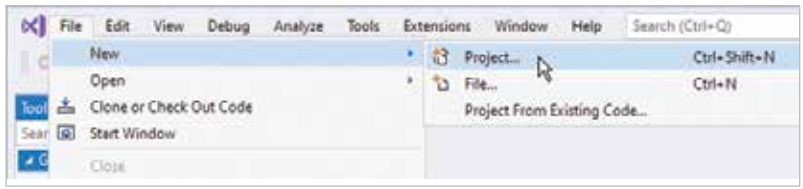
- **Menu Bar** – Where you can select actions to perform on all your project files and to access Help. When a project is open, extra menus of Project and Build are shown in addition to the default menu selection of File, Edit, View, Debug, Test, Analyze, Tools, Extensions, Window, and Help.
- **Toolbar** – Where you can perform the most popular menu actions with just a single click on their associated shortcut icons.
- **Toolbox** – Where you can select visual elements to add to a project. Click the Toolbox side bar button to see its contents. When a project is open, "controls" such as Button, Label, CheckBox, RadioButton, and TextBox are shown here.
- **Solution Explorer** – Where you can see at a glance all the files and resource components contained within an open project.
- **Status Bar** – Where you can read the state of the current activity being undertaken. When building an application, a "Build started" message is displayed here, changing to a "Build succeeded" or "Build failed" message upon completion.

To change the color, choose the **Tools**, **Options** menu then select **Environment**, **General, Color Theme**.

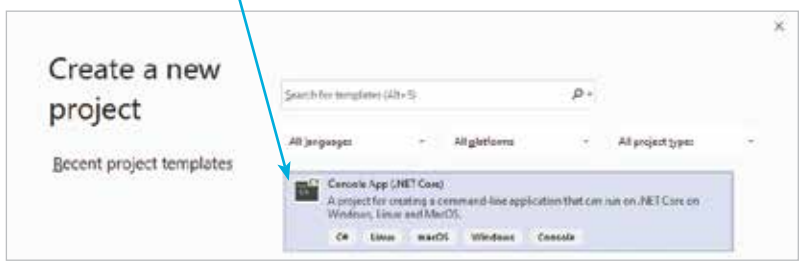
Starting a Console project

GettingStarted

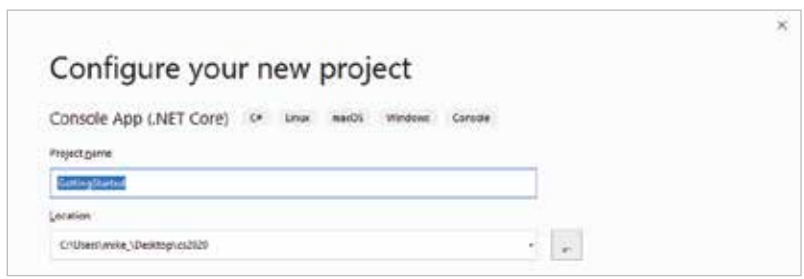
1 On the Menu Bar, click **File**, **New**, **Project...**, or press the **Ctrl** + **Shift** + **N** keys, to open the "New Project" dialog box



2 In the "Create a new project" dialog box, select the **C# Console App (.NET Core)** item and click **Next**



3 On the next dialog, enter a project name and location then click the **Create** button



Visual Studio now creates your new project and loads it into the IDE. A **Code Editor** window appears, containing default skeleton project code generated by Visual Studio.

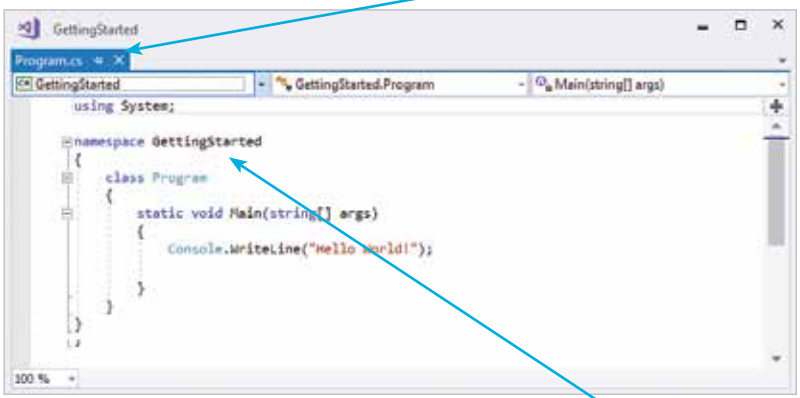
4 Drag the **Code Editor** window tab to undock the **Code Editor** window from the Visual Studio IDE frame

The source code of all examples in this book is available for free download at www.ineasysteps.com/resource-center/downloads

The default location for Visual Studio projects is a **C:\Users*username*\source\repos** directory.

If the Code Editor window does not open automatically, click the **Program.cs** file icon in Solution Explorer to open the Code Editor.

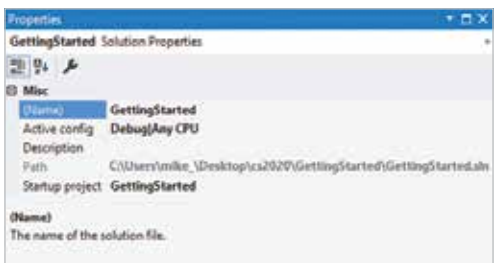
The undocked window title displays the project name, and the tab displays the file name of the code as "Program.cs".



The code **namespace** is declared using your chosen project name – in this case it's "GettingStarted".

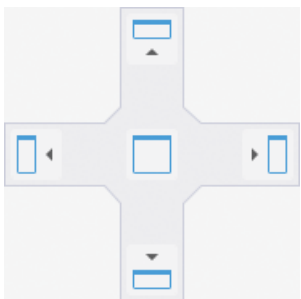
5 Select the **View**, **Solution Explorer** menu to open a **Solution Explorer** window, to discover all the items in your project

6 Select the **View**, **Properties** menu to open a **Properties** window, then select any item in the **Solution Explorer** window to see its properties appear in the **Properties** window



The **Code Editor** window is where you write C# code to create an application. The Visual Studio IDE has now gathered all the resources needed to build a default Console application. You can click the **Start** button on the toolbar to see Visual Studio build this application, but it will do nothing until you add some code.

You can drag the title bar of any window to undock that window from the Visual Studio IDE frame. When dragging, you can drop a window on the "guide diamond" (shown below) to dock the window in your preferred position.



Alternatively, you can run applications using the **Debug**, **Start Debugging** menu options, or press the **F5** shortcut key.

Writing your first program

In order to create a working Console application you need to add C# code to the default skeleton project code generated by the Visual Studio IDE:

Hello

1. On the Menu Bar, click **File**, **New**, **Project**, or press the **Ctrl** + **Shift** + **N** keys, to open the "New Project" dialog box

2. In the "New Project" dialog box, select the **Installed**, **Visual C#**, **Console App (.NET Core)** item

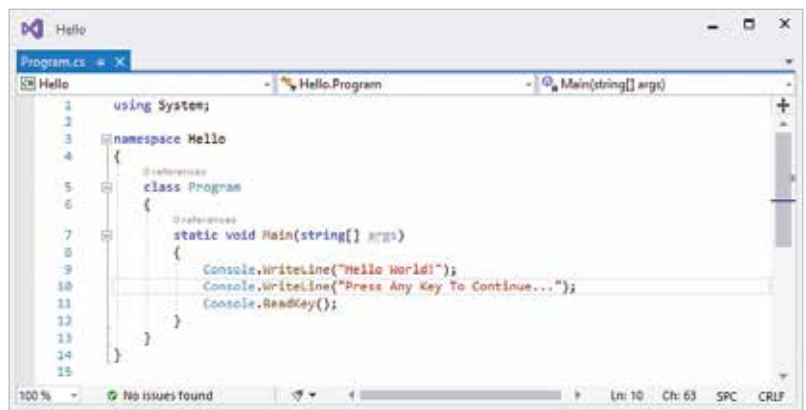
3. Enter a project name of your choice in the **Name** field – in this case the project name will be "Hello"

4. Click on the **OK** button to create the new project and see the **Code Editor** display the default skeleton project code

5. Position the cursor at the end of the line that reads
   `Console.WriteLine( "Hello World!" ) ;`

6. Hit Enter to add a new line, then precisely type this code
   `Console.WriteLine( "Press Any Key To Continue..." ) ;`

7. Hit **Enter** to add another new line, then type this code
   `Console.ReadKey( ) ;`

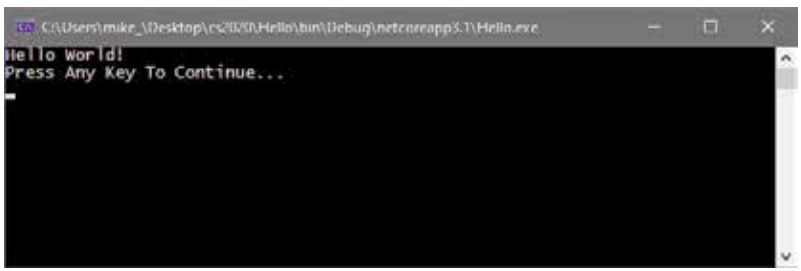


As you type the code, a suggestion box will appear. This is the "IntelliSense" feature. You can select an item then insert it into your code by pressing the Tab key or the Spacebar.

The **Main()** method is automatically called whenever a C# program is run – to execute the instructions contained within its **{ }** braces.

8 Now, select **File**, **Save Hello**, or press the **Ctrl** + **S** keys, to save the completed C# Console application

9 Then, select the Start **Start** button on the Toolbar, or press the **F5** key, to build and run the application



A Console window like the one shown above should now appear, displaying a traditional programming greeting.

10 Hit **Enter**, or click the **Stop** button, to close the application and see the Console window disappear

Code analysis

Examination of the code helps to understand what is happening:

- **using System ;** This is a directive allowing the **System.Console** class object to be written without the **System.** prefix.
- **namespace Hello { }** This is a declaration that creates a unique namespace wrapper in which to enclose your program.
- **class Program { }** This declaration creates a "Program" class in which to create your own program properties and methods.
- **static void Main(string[ ] args) { }** This declaration creates a standard **Main( )** method in which to write your C# code.
- **Console.WriteLine("Hello World!") ;** This is a statement that calls upon the **WriteLine( )** method of the **Console** class to output text enclosed in quote marks within its parentheses.
- **Console.ReadKey( ) ;** This statement calls upon the **ReadKey( )** method of the **Console** class to wait for any key to be pressed.

To edit the default Console window colors and font, right-click its window Titlebar and choose **Properties**. For clarity, all other Console window screenshots in this book feature Lucida Console 14-pixel **Font** in black **Screen Text** on a white **Screen Background**.

Code listed throughout this book is colored to match the default syntax highlight colors of the Visual Studio Code Editor for easy recognition.

Calling the **ReadKey( )** method is a little trick to keep the Console window open until you press any key. Without this statement, the application would output its message then immediately exit.

Following the rules

As with all programming languages, C# has a number of syntax rules that must be precisely followed to ensure the code is correctly formatted for the C# compiler to clearly understand:

- **Case-sensitivity** – C# is a case-sensitive language, which means that uppercase "A" and lowercase "a" are regarded as totally different items.
- **Termination** – All statements in C# language must be terminated by a **;** semicolon character, just as all sentences in English language must be terminated by a . period character. For example: `Console.WriteLine( "Hello World!" ) ;`
- **Single-line comments** – Brief comments on a single line must begin with // two forward slash characters. For example: `// Output the traditional greeting.`
- **Block comments** – Extended comments on multiple lines must begin with /* forward slash and asterisk characters, and must end with the reverse */ asterisk and forward slash. For example:

```
/*
  C# Programming in easy steps.
  Getting started with the traditional greeting.
*/
```

- **White space** – Spaces, tabs, newline characters, and comments are ignored by the C# compiler, so can be used extensively to organize code without performance penalty.
- **Escape sequences** – The C# compiler recognizes **\n** as a newline character and **\t** as a tab character, so these can be used to format output. For example: `Console.WriteLine("Line One \n Line Two") ;`
- **Naming conventions** – A programmer-defined identifier name in C# code may begin with an _ underscore character or a letter in uppercase or lowercase. The name may also contain an underscore, letters, and numerals. For example: `class MyNo1_Class`
- **Keywords** – The C# language has a number of keywords (listed opposite) that have special syntactic meaning and may not be used to name programmer-defined items in code.

It is recommended that you comment your code to make it readily understood by others or when revisiting your own code later.

The **WriteLine( )** method automatically adds a newline after its output.

C# Reserved Keywords			
abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	override	params	private
protected	public	readonly	ref
return	sbyte	sealed	short
sizeof	stackalloc	static	string
struct	switch	this	throw
true	try	typeof	uint
ulong	unchecked	unsafe	ushort
using	virtual	void	volatile
while			

If you absolutely must use a keyword to name a programmer-defined element, it may be prefixed by an @ character to distinguish it from the keyword – but this is best avoided.

C# Contextual Keywords			
add	alias	ascending	async
await	descending	dynamic	from
get	global	group	into
join	let	orderby	partial
remove	select	set	value
var	where	yield	

Contextual keywords have special significance in certain code. For example, **get** and **set** in method declarations.

Summary

- **C#** is an object-oriented programming language that utilizes the proven functionality of the Microsoft **.NET** class libraries.
- The C# compiler generates **Intermediate Language (IL)** code that is stored on disk alongside resources in an assembly.
- The **Common Language Runtime (CLR)** examines an assembly's security requirements before JIT compilation.
- **Just-In-Time** compilation translates IL code into native machine code for execution by the operating system.
- Microsoft Visual Studio provides a fully **Integrated Development Environment (IDE)** for C# programming.
- A new Visual C# **Console** application generates default skeleton project code in the Visual Studio Code Editor.
- The Visual Studio **Solution Explorer** shows all files in a project, and the **Properties** window shows their properties.
- C# code needs to be added to the default skeleton code in the **Code Editor** to create a C# program.
- The `using System` directive allows the `System.Console` class to be written in the code without its `System.` prefix.
- The `Console` class has a `WriteLine( )` method that can be used to output a specified text string, and a `ReadKey( )` method that can recognize when the user presses any key.
- A C# program can be run in the Visual Studio IDE by selecting the **Debug**, **Start Debugging** menu, or by clicking the **Start** button, or by pressing the **F5** key.
- C# is a case-sensitive programming language in which all statements must be terminated by a ; semicolon character.
- Single-line `//` comments and `/* */` block comments can be incorporated to explain C# program code.
- C# has keywords that have special syntactic meaning, so cannot be used to name programmer-defined code items.