
# ASP.NET CORE

## Documentation

### Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you're ready to add your content, just click here and start typing.]

Dhruvil Dobariya
dhruvildobariya21@gmail.com

# INDEX


**1 VISUAL STUDIO 2019 IDE OVERVIEW ..... 1**

1.1 WHAT IS A VISUAL STUDIO? ..... 1
1.2 DIFFERENT TYPES OF WINDOWS ..... 1
1.3 SOLUTION AND PROJECT ..... 2
1.4 CODE EDITOR FEATURES ..... 4
1.5 POPULAR KEYBOARD SHORTCUTS FOR VISUAL STUDIO: ..... 9

**2 PROJECT TYPES ..... 17**

2.1 WINDOWS DEVELOPMENT ..... 17
2.2 CLASS LIBRARY ..... 18
2.3 MOBILE DEVELOPMENT ..... 19
2.4 WEB DEVELOPMENT ..... 20

**3 INTRODUCTION TO C# ..... 22**

3.1 WHAT IS A C#? ..... 22
3.2 "HELLO WORLD" PROGRAM ..... 22
3.3 NAMESPACE: ..... 23
3.4 THE "USING" KEYWORD ..... 24
3.5 CLASS ..... 25
3.6 METHODS ..... 26
3.7 VARIABLES ..... 27

**4 UNDERSTANDING C# PROGRAM ..... 28**

4.1 PROGRAM FLOW ..... 28
4.2 UNDERSTANDING SYNTAX ..... 28
4.3 COMMON LANGUAGE RUNTIME (CLR) ..... 29

**5 WORKING WITH CODE FILES, PROJECTS & SOLUTIONS ..... 30**

5.1 STRUCTURE OF SOLUTION ..... 30
5.2 UNDERSTANDING STRUCTURE OF PROJECT ..... 30

**6 UNDERSTANDING DATATYPES & VARIABLES WITH CONVERSION ..... 37**

6.1 DATA TYPE: ..... 37
6.2 DATATYPE CONVERSION ..... 41

**7 UNDERSTANDING DECISION MAKING & STATEMENTS ..... 46**

7.1 IF ELSE ..... 46
7.2 SWITCH ..... 53

**8 OPERATORS AND EXPRESSIONS ..... 57**

8.1 OPERATORS ..... 57

**9 LOOP ITERATION ..... 62**

9.1 WHILE ..... 62

9.2 Do While ..... 63
9.3 For ..... 65
9.4 For Each ..... 66
9.5 Break Statement ..... 68
9.6 Continue Statement ..... 69
9.7 Go To Statement ..... 71

**10 UNDERSTANDING ARRAYS ..... 72**

10.1 Introduction ..... 72
10.2 Declaring Array ..... 72
10.3 Single dimensional array ..... 73
10.4 Multidimensional array ..... 74
10.5 Jugged array ..... 77
10.6 Array Class ..... 79

**11 DEFINING AND CALLING METHODS ..... 83**

11.1 Introduction ..... 83
11.2 Defining the method ..... 83
11.3 Calling Method ..... 84
11.4 Passing Parameter to the method ..... 85

**12 WORKING WITH STRINGS ..... 86**

12.1 Introduction ..... 86
12.2 String ..... 86
12.3 StringBuilder ..... 87

**13 WORKING WITH DATETIMES ..... 90**

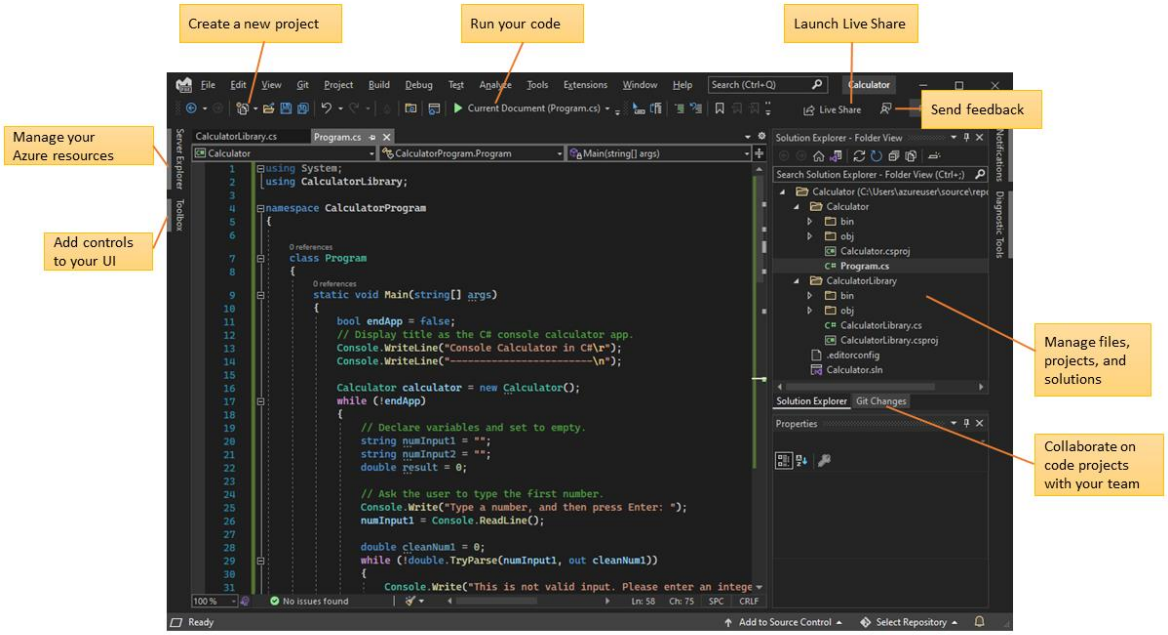
13.1 Introduction ..... 90
13.2 DateTime ..... 90

# VISUAL STUDIO 2019 IDE OVERVIEW

## 1.1 WHAT IS A VISUAL STUDIO?

- Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps.
- Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists.

## 1.2 DIFFERENT TYPES OF WINDOWS




- In Solution Explorer, at upper right, you can view, navigate, and manage your code files. Solution Explorer can help organize your code by grouping the files into solutions and projects.
- The central editor window, where you'll probably spend most of your time, displays file contents. In the editor window, you can edit code or design a user interface such as a window with buttons and text boxes.

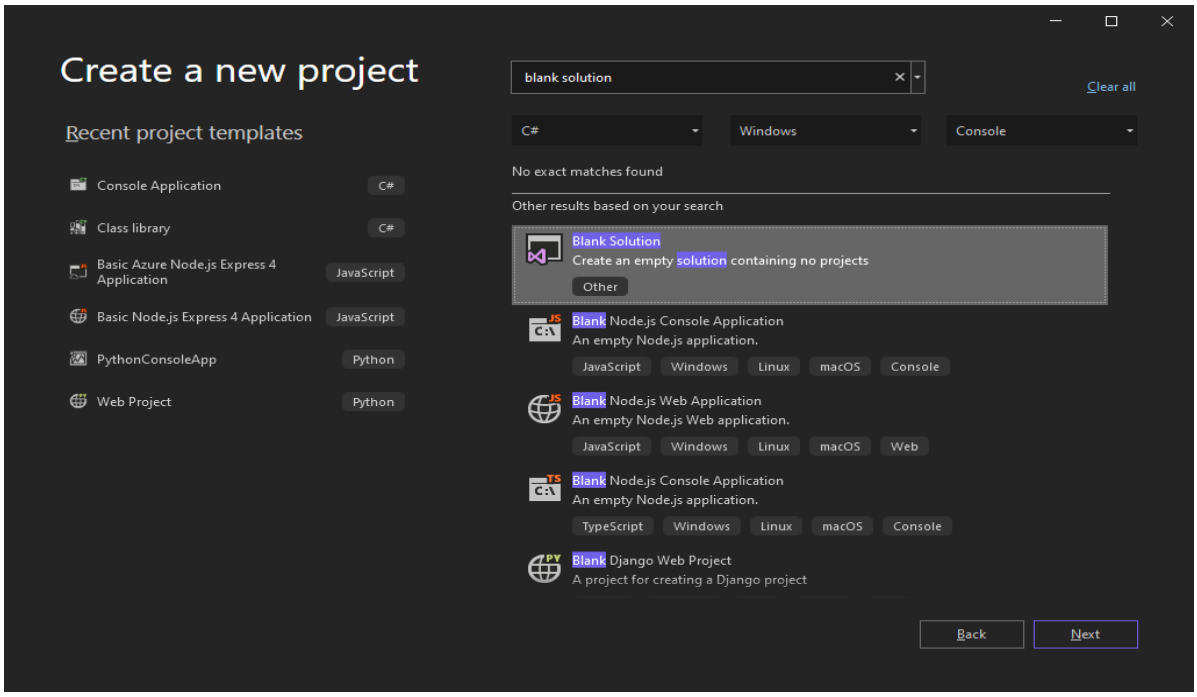
- The central editor window, where you'll probably spend most of your time, displays file contents. In the editor window, you can edit code or design a user interface such as a window with buttons and text boxes.

## 1.3 Solution and Project

- We are used Solution Explorer to manage project. Using Solution Explorer we should manage folder structure and navigate different files.
- Solution Explorer have a file which is contain solution details and projects details.
- This file have *'.sln'* extension.
- Solution contain one or more projects.
- Project have one file which containe details about project.
- Which have *'.csproj'* extension(if project base on C#).

### 1.3.1 Create a solution:

- Open Visual Studio, and on the start window, select Create a new project.




- On the Configure your new project page, give the name of Solution, and then select Create.

- Let say our solution name is **'QuickSolution'**.

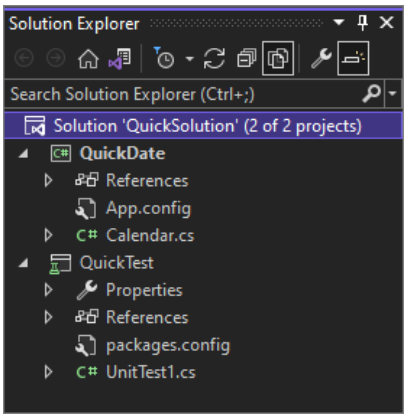
### 1.3.2 ADD A PROJECT:

- Right-click Solution **'QuickSolution'** in Solution Explorer, and select Add > New Project from the context menu.
- On the Add a new project page, type empty into the search box at the top, and select C# under All languages.
- Select the C# Empty Project (.NET Framework) template, and then select Next.
- On the Configure your new project page, give the name of Project, and then select Create.
- Let say our solution name is **'QuickDate'**.

### 1.3.3 ADD AN ITEM TO THE PROJECT:

- From the right-click or context menu of the **'QuickDate'** project in Solution Explorer, select Add > New Item.
- Expand Visual C# Items, and then select Code. In the middle pane, select the Class item template. Under Name, give the name of file, and then select Add.
- Let say our file name is **'Calander.cs'**.

### 1.3.4 ADD A SECOND PROJECT:

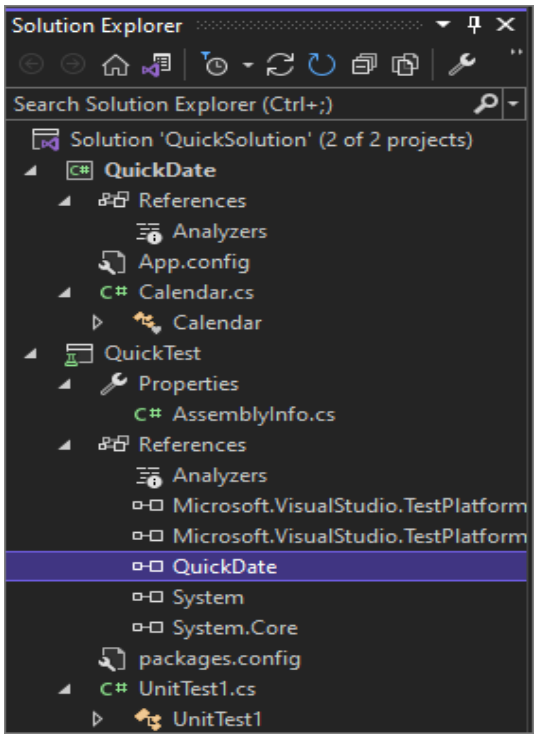



- From the right-click or context menu of Solution **'QuickSolution'** in Solution Explorer, select Add > New Project.
- In the Add a new project dialog box, type unit test into the search box at the top, and then select C# under All languages.
- Select the C# Unit Test Project (.NET Framework) project template, and then select Next.
- On the Configure your new project page, give the name of project, and then select Create.
- Let say our second project name is **'QuickTest'.**
- Add one file inside the second project.
- Let say our file name is **'UnitTest1.cs'**.

### 1.3.5 ADD A PROJECT REFERENCE:

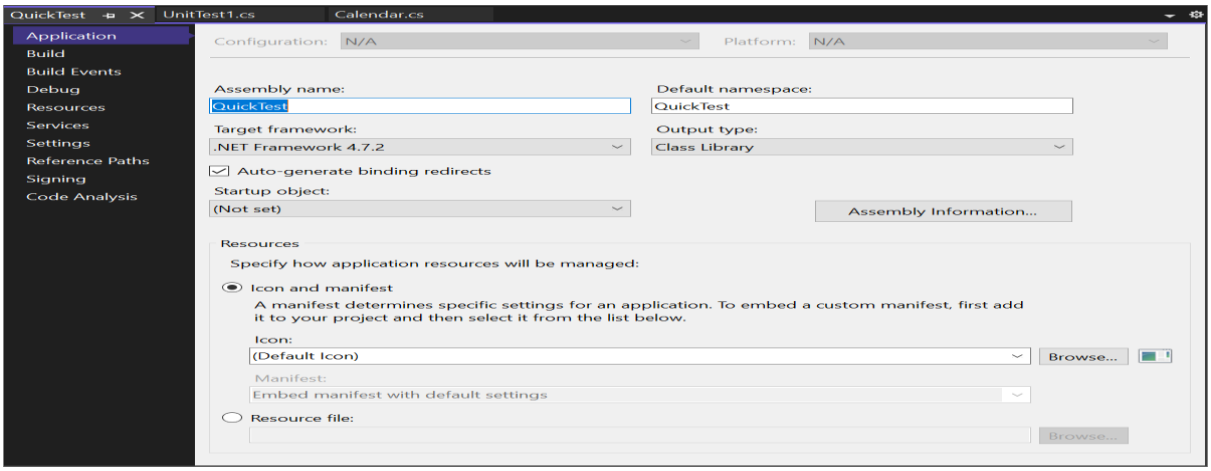
- In Solution Explorer, right-click the References node of the **'QuickTest'** project, and select Add Reference from the context menu.
- In the Reference Manager dialog box, under Projects, select the checkbox next to **'QuickDate',** and then select OK.

- Now a reference to the **'QuickDate'** project appears under the **'QuickTest'** project in Solution Explorer.




### 1.3.6 PROJECT PROPERTIES:

- In Solution Explorer, right-click the **'QuickTest'** project and select Properties, or select the project and press Alt+Enter.




## 1.4 CODE EDITOR FEATURES

- Some popular features in Visual Studio that improve your productivity when developing software include

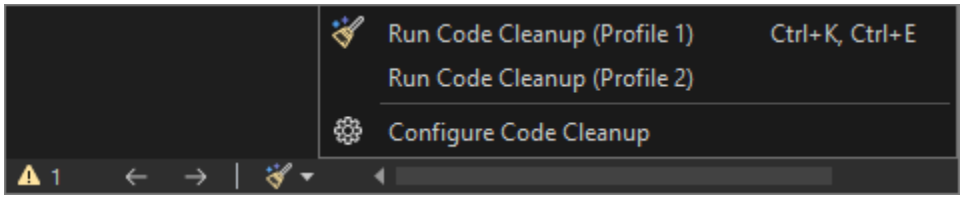
### 1.4.1 Squiggles and Quick Actions:

- Squiggles are wavy underlines that alert you to errors or potential problems in your code as you type.
- These visual clues help you fix problems immediately, without waiting to discover errors during build or runtime.


- If you hover over a squiggle, you see more information about the error.
- A lightbulb might also appear in the left margin showing Quick Actions you can take to fix the error.

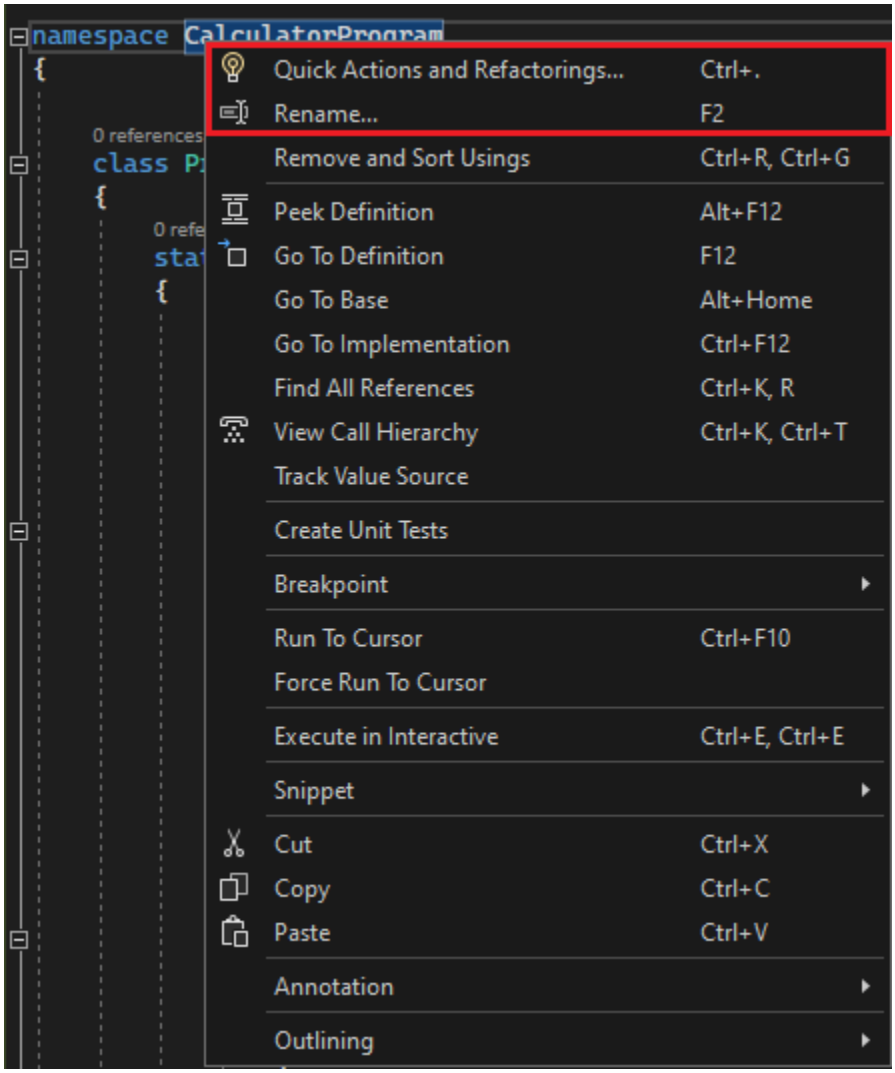
### 1.4.2 Code Cleanup:

- With the click of a button, you can format your code and apply any code fixes suggested by your code style settings, .editorconfig conventions, and Roslyn analyzers.
- Code Cleanup, currently available for C# code only, helps you resolve issues in your code before it goes to code review.



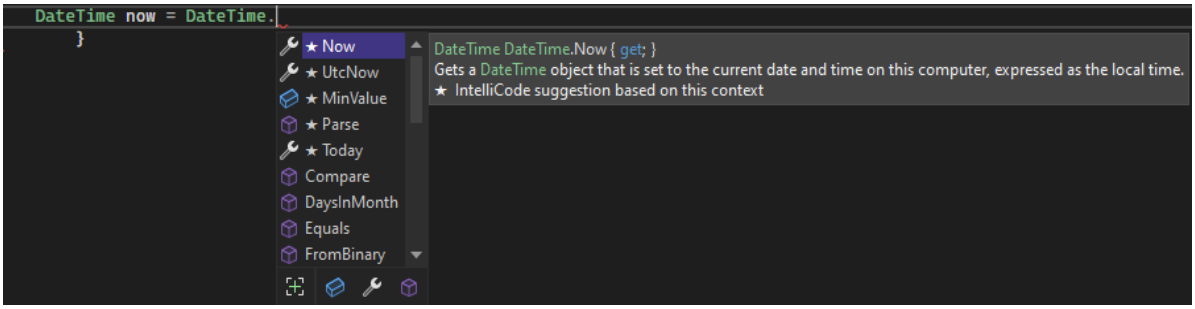

### 1.4.3 Refactoring:

- Refactoring includes operations such as intelligent renaming of variables, extracting one or more lines of code into a new method, and changing the order of method parameters.




### 1.4.4 IntelliSense:

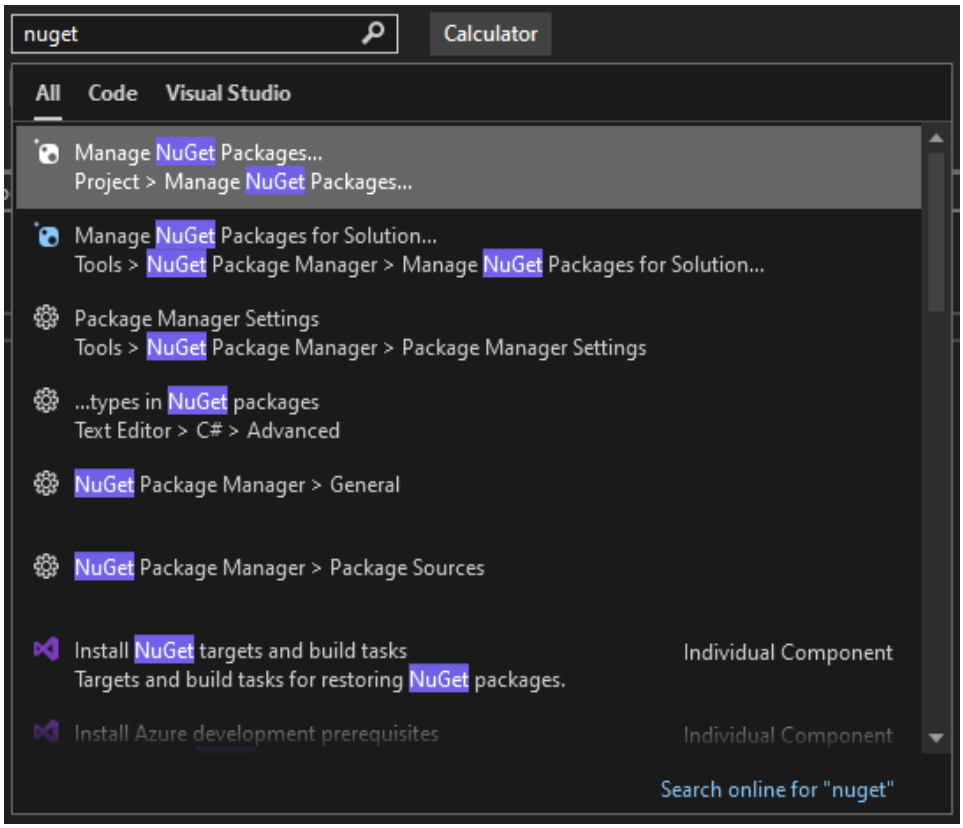
- IntelliSense is a set of features that display information about your code directly in the edor and, in some cases, write small bits of code for you.
- It's like having basic documentation inline in the editor, so you don't have to look up type




information elsewhere.

### 1.4.5 Visual Studio search:

- Visual Studio menus, options, and properties can seem overwhelming at times.
- Visual Studio search, or *'Ctrl+Q'*, is a great way to rapidly find IDE features and code in one place.

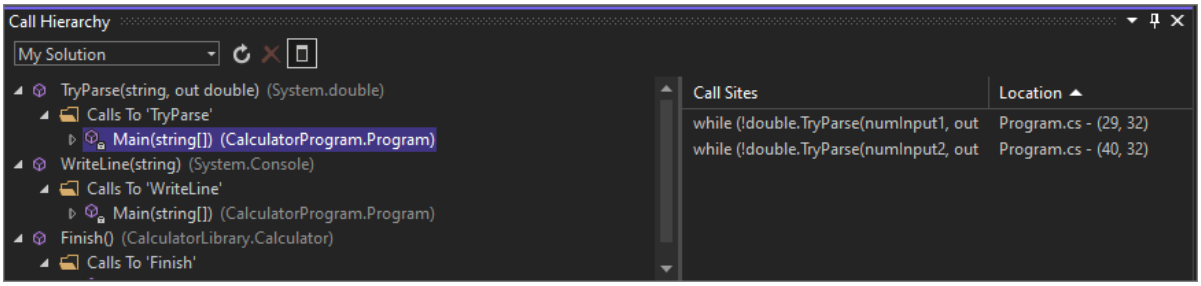



### 1.4.6 Live Share:

- Collaboratively edit and debug with others in real time, regardless of your app type or programming language.
- You can instantly and securely share your project. You can also share debugging sessions, terminal instances, localhost web apps, voice calls, and more.

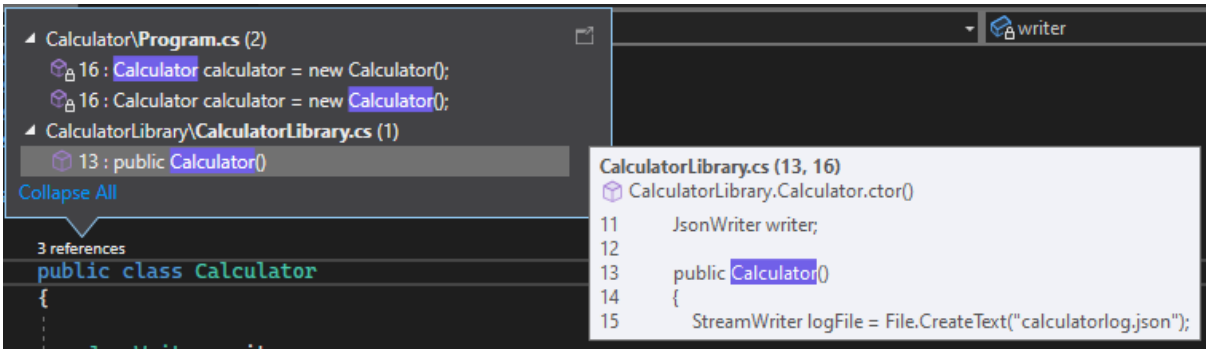
### 1.4.7 Call Hierarchy:

- The Call Hierarchy window shows the methods that call a selected method.
- This information can be useful when you're thinking about changing or removing the method, or when you're trying to track down a bug.



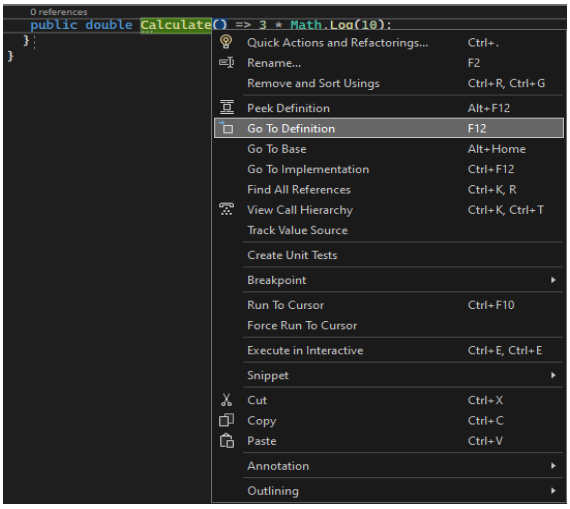

### 1.4.8 CodeLens:

- CodeLens helps you find code references, code changes, linked bugs, work items, code reviews, and unit tests, without leaving the editor.



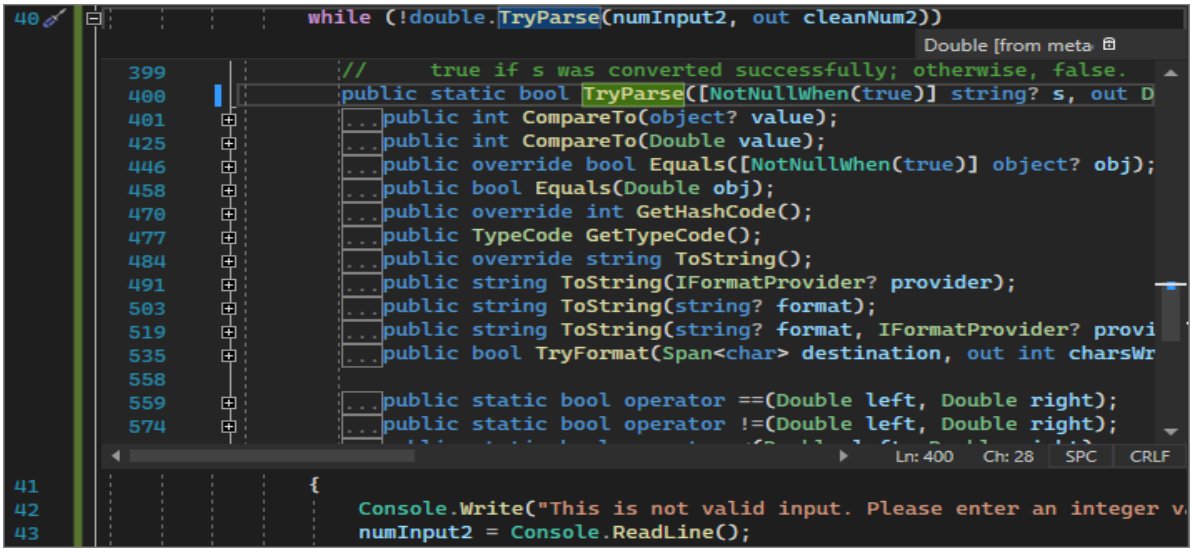

### 1.4.9 Go To Definition:

- The Go To Definition feature takes you directly to the location of a function or type definition.



### 1.4.10 Peek Definition:

- The Peek Definition window shows a method or type definition without opening a separate file.




## 1.5 Popular keyboard shortcuts for Visual Studio:

### 1.5.1 Build:

| Commands | Keyboard shortcuts | Command ID |
|---|---|---|
| Build solution | Ctrl+Shift+B | Build.BuildSolution |
| Cancel | Ctrl+Break | Build.Cancel |
| Compile | Ctrl+F7 | Build.Compile |
| Run code analysis on solution | Alt+F11 | Build.RunCodeAnalysisonSolution |

### 1.5.2 Debug:

| Commands | Keyboard shortcuts [Special contexts] | Command ID |
|---|---|---|
| Break at function | Ctrl+B | Debug.BreakatFunction |
| Break all | Ctrl+Alt+Break | Debug.BreakAll |

| | | |
|---|---|---|
| Delete all breakpoints | Ctrl+Shift+F9 | Debug.DeleteAllBreakpoints |
| Exceptions | Ctrl+Alt+E | Debug.Exceptions |
| Quick watch | Ctrl+Alt+Q<br><br>or Shift+F9 | Debug.QuickWatch |
| Restart | Ctrl+Shift+F5 | Debug.Restart |
| Run to cursor | Ctrl+F10 | Debug.RunToCursor |
| Set next statement | Ctrl+Shift+F10 | Debug.SetNextStatement |
| Start | F5 | Debug.Start |
| Start without debugging | Ctrl+F5 | Debug.StartWithoutDebugging |
| Step into | F11 | Debug.StepInto |
| Step out | Shift+F11 | Debug.StepOut |
| Step over | F10 | Debug.StepOver |
| Stop debugging | Shift+F5 | Debug.StopDebugging |
| Toggle breakpoint | F9 | Debug.ToggleBreakpoint |

### 1.5.3 EDIT:

| **Commands** | **Keyboard shortcuts [Special contexts]** | **Command ID** |
|---|---|---|
| Break line | Enter [Text Editor, Report Designer, Windows Forms Designer]<br><br>or Shift+Enter [Text Editor] | Edit.BreakLine |
| Collapse to definitions | Ctrl+M, Ctrl+O [Text Editor] | Edit.CollapseToDefinitions |

| | | |
|---|---|---|
| Comment selection | Ctrl+K, Ctrl+C [Text Editor] | Edit.CommentSelection |
| Complete word | Alt+Right Arrow [Text Editor, Workflow Designer]<br><br>or Ctrl+Spacebar [Text Editor, Workflow Designer]<br><br>or Ctrl+K, W [Workflow Designer]<br><br>or Ctrl+K, Ctrl+W [Workflow Designer] | Edit.CompleteWord |
| Copy | Ctrl+C<br><br>or Ctrl+Insert | Edit.Copy |
| Cut | Ctrl+X<br><br>or Shift+Delete | Edit.Cut |
| Delete | Delete [Team Explorer]<br><br>or Shift+Delete [Sequence Diagram, UML Activity Diagram, Layer Diagram]<br><br>or Ctrl+Delete [Class Diagram] | Edit.Delete |
| Find | Ctrl+F | Edit.Find |
| Find all references | Shift+F12 | Edit.FindAllReferences |
| Find in files | Ctrl+Shift+F | Edit.FindinFiles |
| Find next | F3 | Edit.FindNext |
| Find next selected | Ctrl+F3 | Edit.FindNextSelected |
| Format document | Ctrl+K, Ctrl+D [Text Editor] | Edit.FormatDocument |

| | | |
|---|---|---|
| Format selection | Ctrl+K, Ctrl+F [Text Editor] | Edit.FormatSelection |
| Go to | Ctrl+G | Edit.GoTo |
| Go to declaration | Ctrl+F12 | Edit.GoToDeclaration |
| Go to definition | F12 | Edit.GoToDefinition |
| Go to find combo | Ctrl+D | Edit.GoToFindCombo |
| Go to next location | F8 | Edit.GoToNextLocation |
| Insert snippet | Ctrl+K, Ctrl+X | Edit.InsertSnippet |
| Insert tab | Tab [Report Designer, Windows Forms Designer, Text Editor] | Edit.InsertTab |
| Line cut | Ctrl+L [Text Editor] | Edit.LineCut |
| Line down extend column | Shift+Alt+Down Arrow [Text Editor] | Edit.LineDownExtendColumn |
| Line open above | Ctrl+Enter [Text Editor] | Edit.LineOpenAbove |
| List members | Ctrl+J [Text Editor, Workflow Designer]<br><br>or Ctrl+K, Ctrl+L [Workflow Designer]<br><br>or Ctrl+K, L [Workflow Designer] | Edit.ListMembers |
| Navigate to | Ctrl+, | Edit.NavigateTo |
| Open file | Ctrl+Shift+G | Edit.OpenFile |
| Overtype mode | Insert [Text Editor] | Edit.OvertypeMode |
| Parameter info | Ctrl+Shift+Spacebar [Text Editor, Workflow Designer]<br>or Ctrl+K, Ctrl+P [Workflow Designer]<br>or Ctrl+K, P [Workflow Designer] | Edit.ParameterInfo |

| | | |
|---|---|---|
| Paste | Ctrl+V<br>or Shift+Insert | Edit.Paste |
| Peek definition | Alt+F12 [Text Editor] | Edit.PeekDefinition |
| Redo | Ctrl+Y<br>or Shift+Alt+Backspace<br>or Ctrl+Shift+Z | Edit.Redo |
| Replace | Ctrl+H | Edit.Replace |
| Select all | Ctrl+A | Edit.SelectAll |
| Select current word | Ctrl+W [Text Editor] | Edit.SelectCurrentWord |
| Selection cancel | Esc [Text Editor, Report Designer, Settings Designer, Windows Forms Designer, Managed Resources Editor] | Edit.SelectionCancel |
| Surround with | Ctrl+K, Ctrl+S<br>(available only in Visual Studio 2019 and earlier) | Edit.SurroundWith |
| Tab left | Shift+Tab [Text Editor, Report Designer, Windows Forms Editor] | Edit.TabLeft |
| Toggle all outlining | Ctrl+M, Ctrl+L [Text Editor] | Edit.ToggleAllOutlining |
| Toggle bookmark | Ctrl+K, Ctrl+K [Text Editor] | Edit.ToggleBookmark |
| Toggle completion mode | Ctrl+Alt+Space [Text Editor] | Edit.ToggleCompletionMode |
| Toggle outlining expansion | Ctrl+M, Ctrl+M [Text Editor] | Edit.ToggleOutliningExpansion |
| Uncomment selection | Ctrl+K, Ctrl+U [Text Editor] | Edit.UncommentSelection |
| Undo | Ctrl+Z<br>or Alt+Backspace | Edit.Undo |
| Word delete to end | Ctrl+Delete [Text Editor] | Edit.WordDeleteToEnd |

| | | |
|---|---|---|
| Word delete to start | Ctrl+Backspace [Text Editor] | Edit.WordDeleteToStart |

### 1.5.4 File: popular shortcuts:

| Commands | Keyboard shortcuts [Special contexts] | Command ID |
|---|---|---|
| Exit | Alt+F4 | File.Exit |
| New file | Ctrl+N | File.NewFile |
| New project | Ctrl+Shift+N | File.NewProject |
| New web site | Shift+Alt+N | File.NewWebSite |
| Open file | Ctrl+O | File.OpenFile |
| Open project | Ctrl+Shift+O | File.OpenProject |
| Open web site | Shift+Alt+O | File.OpenWebSite |
| Rename | F2 [Team Explorer] | File.Rename |
| Save all | Ctrl+Shift+S | File.SaveAll |
| Save selected items | Ctrl+S | File.SaveSelectedItems |
| View in browser | Ctrl+Shift+W | File.ViewinBrowser |

### 1.5.5 Project:

| Commands | Keyboard shortcuts [Special contexts] | Command ID |
|---|---|---|
| Add existing item | Shift+Alt+A | Project.AddExistingItem |
| Add new item | Ctrl+Shift+A | Project.AddNewItem |

### 1.5.6 Refactor:

| Command | Keyboard shortcut [Special contexts] | Command ID |
|---|---|---|
| Extract method | Ctrl+R, Ctrl+M | Refactor.ExtractMethod |

### 1.5.7 TOOLS:

| Command | Keyboard shortcut [Special contexts] | Command ID |
|---|---|---|
| Attach to process | Ctrl+Alt+P | Tools.AttachtoProcess |

### 1.5.8 VIEW:

| Commands | Keyboard shortcuts [Special contexts] | Command ID |
|---|---|---|
| Class view | Ctrl+Shift+C | View.ClassView |
| Edit label | F2 | View.EditLabel |
| Error list | Ctrl+\, Ctrl+E<br>or Ctrl+\, E | View.ErrorList |
| Navigate backward | Ctrl+- | View.NavigateBackward |
| Navigate forward | Ctrl+Shift+- | View.NavigateForward |
| Object browser | Ctrl+Alt+J | View.ObjectBrowser |
| Output | Ctrl+Alt+O | View.Output |
| Properties window | F4 | View.PropertiesWindow |
| Refresh | F5 [Team Explorer] | View.Refresh |
| Server explorer | Ctrl+Alt+S | View.ServerExplorer |
| Show smart tag | Ctrl+.<br><br>or Shift+Alt+F10 [HTML Editor Design View] | View.ShowSmartTag |
| Solution explorer | Ctrl+Alt+L | View.SolutionExplorer |
| TFS Team Explorer | Ctrl+\, Ctrl+M | View.TfsTeamExplorer |
| Toolbox | Ctrl+Alt+X | View.Toolbox |
| View code | Enter [Class Diagram]<br><br>or F7 [Settings Designer] | View.ViewCode |

| | | |
|---|---|---|
| View designer | Shift+F7 [HTML Editor Source View] | View.ViewDesigner |

### 1.5.9 WINDOW:

| **Commands** | **Keyboard shortcuts [Special contexts]** | **Command ID** |
|---|---|---|
| Activate document window | Esc | Window.ActivateDocumentWindow |
| Close document window | Ctrl+F4 | Window.CloseDocumentWindow |
| Next document window | Ctrl+F6 | Window.NextDocumentWindow |
| Next document window nav | Ctrl+Tab | Window.NextDocumentWindowNav |
| Next split pane | F6 | Window.NextSplitPane |

# PROJECT TYPES

## 2.1 WINDOWS DEVELOPMENT

- There are three main application framework available in .Net Framework for developing windows application.
  - Windows Forms
  - Windows Presentation Foundation (WPF)
  - Universal Windows Platform (UWP)

### 2.1.1 WINDOWS FORMS:

- The first version of Windows Forms was released in 2002 at the same time as .NET framework 1.0.
- The code was written in an event-driven manner.
- Here application contain multiple windows, called as a forms.
- Business logic of application spread across the many event handlers in multiple forms.
- So it's difficult to manage large application.
- So for the avoid this issue we are used MVP (model-view-presenter) design pattern architecture.
- But, all of this makes Windows Forms not very suitable for creating new applications.

### 2.1.2 WINDOWS PRESENTATION FOUNDATION (WPF):

- Windows Presentation Foundation (WPF) was released as a part of .NET framework 3.5 in 2007.
- It is saved as an XML file using a special syntax named XAML (Extensible Application Markup Language).
- Unlike the Windows forms, this XML file is much easier to understand and edit manually.
- Also, the synchronization between the designer and the XML file is bidirectional.
- Any changes made directly to the XML file are immediately visible in the designer.
- This allows for greater flexibility when editing the layout.
- But, the code is still event driven.
- But, here the data property and event handler can be bound to control using XAML markup.
- So taking advantage of this and introduce the MVVM (model-view-viewmodel) design pattern.

- Even today, WPF is the most versatile and flexible framework for creating Windows desktop applications and as such the recommended choice for most new Windows desktop applications.

### 2.1.3 Universal Windows Platform (UWP):

- The origin of Universal Windows Platform (UWP) can be traced back to the release of Windows 8 in 2012.
- This is used to accompanying framework for development of touch-first applications, called Metro applications.
- With the release of Windows 10 in 2015, the framework got its final name and eventually supported development of applications like…
    - Windows desktop
    - Windows Mobile
    - Windows IoT Core
    - Windows Mixed Reality
    - Xbox Ones
- It is very similar to WPF.
- User interfaces are save as a XAML files.
- It used MVVM design pattern.
- UWP applications can call some Win32 APIs when their code is written in C++/CX.
- Windows API and WinAPI, Win32 is the main set of Microsoft Windows APIs used for developing 32-bit applications.
- These APIs are responsible for functions in the following categories:
    - Administration
    - Management – Install
    - Configure
    - Service applications or systems
- UWP applications are your only choice if you want to target any non-desktop Windows devices.
- You might also prefer them over WPF for Windows desktop applications if you want to target other Windows devices with the same application or want to publish your application in Microsoft Store as long as you don't need any Win32 APIs not available to you in UWP applications.

## 2.2 Class Library

- Class library is a type of project in .Net which is help use to manage application in different types of modules.

- It will also used to share code across the multiple projects.
- There are three types of class libraries that you can use:
  - Platform-specific class libraries
  - Portable class library
  - .Net Standard class library

### 2.2.1 Platform-specific class libraries:

- It have access to all the APIs in a given platform (for example, .NET Framework on Windows, Xamarin iOS), but can only be used by apps and libraries that target that platform.
- Platform-specific libraries are bound to a single .NET platform and can therefore take significant dependencies on a known execution environment.
- Platform-specific libraries have been the primary class library type for the .NET Framework.

### 2.2.2 Portable class libraries:

- It have access to a subset of APIs, and can be used by apps and libraries that target multiple platforms.

### 2.2.3 .NET Standard class libraries:

- It is a combination of the platform-specific and portable library concept into a single model that provides the best of both.
- .NET Standard exposes a set of library contracts.
- .NET implementations must support each contract fully or not at all.
- Each implementation supports a set of .NET Standard contracts.
- .NET Standard class library is supported on the platforms that support its contract dependencies.
- These libraries do expose many more APIs than Portable Class Libraries.
- The following implementations support .NET Standard libraries:
  - .NET Core
  - .NET Framework
  - Mono
  - Universal Windows Platform (UWP)

## 2.3 Mobile Development

- In .Net we have two mobile development frameworks:
  - Xamarin

- .Net MAUI

### 2.3.1 Xamarin:

- Xamarin extends the .NET developer platform with tools and libraries specifically for building apps for Android, iOS, tvOS, watchOS, macOS, and Windows.
- Xamarin apps are native apps! Whether you're designing a uniform UI across platforms.

### 2.3.2 .Net MAUI:

- .NET Multi-platform App UI (.NET MAUI) is a framework for building modern, multi-platform, natively compiled iOS, Android, macOS, and Windows apps using C# and XAML in a single codebase.
- This is support in .Net 6 and above.
- We can also use .Net Blazor Native, Which is help use to develop hybrid applications with C# instead of JavaScript.
- Also you can share youre Blazor web components directly in .NET MAUI apps while having access to native device capabilities and packaging.

## 2.4 Web Development

- ASP.NET offers many frameworks for creating web applications:
  - Web Forms
  - ASP.NET MVC
  - ASP.NET Web Pages
- Many other new framework launched .Net like Blazor, .Net MAUI, Microservices.

### 2.4.1 Web Forms:

- With ASP.NET Web Forms, we can build dynamic websites using a familiar drag-and-drop, event-driven model.
- This is only available in .Net Framework.

### 2.4.2 ASP.Net MVC:

- This is used to create powerful web application based on MVC design pattern with full control and using agile methodology.
- It is available in both .Net Framework and .Net Core.

### 2.4.3 ASP.Net Web Pages:

- ASP.NET Web Pages and the Razor syntax provide a fast, approachable, and lightweight way to combine server code with HTML to create dynamic web content.

- Connect to databases, add video, link to social networking sites, and include many more features that help you create beautiful sites that conform to the latest web standards.
- It is also available in both .Net Framework and .Net Core.

### 2.4.4 ASP.NET WEB API:

- It is used to create HTTP services that reach a broad range of clients, including browsers and mobile devices.
- It is available in both .Net Framework and .Net Core.

### 2.4.5 BLAZOR WEBASSEMBLY:

- It is used to develop single page client side web application with the help of Html, Css and C#.
- Because it's real .NET running on WebAssembly, you can re-use code and libraries from server-side parts of your application.
- It is only available in .Net Core.

### 2.4.6 BLAZOR SERVER:

- It is used to develop real time web application with the SignalR concept.
- It is only available in .Net Core.

# INTRODUCTION TO C#

## 3.1 WHAT IS A C#?

- C# is a general purpose object oriented programing language.
- It developed by Microsoft.
- C# was developed by Anders Hejlsberg and his team during the development of .Net
- Framework.
- C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

### 3.1.1 FEATURES OF C#:

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Technology.

## 3.2 "HELLO WORLD" PROGRAM

- C# Contain following part in a program:
  - Namespace declaration
  - A class
  - Class methods
  - Class attributes or properties
  - A Main Method
  - Statements and Expressions
  - Comments
- C# is case sensitive.
- All statements and expression must end with a semicolon (;).
- The program execution starts at the Main method.
- Unlike Java, program file name could be different from the class name.

```
using System;

namespace HelloWorldAPP
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

```
Output : Hello World
```

## 3.3 NAMESPACE:

- Namespace is a collection of classes.
- It helps to archive level of separation of classes.
- Namespace have a following type as a member
    - Namespace (Nested)
    - Classes
    - Interfaces
    - Structures
    - Delegates
    - Enum
- It is not **mandatory to declare in program**, but they do play an important role to manage large project.
- Namespace also solve a naming conflict problem, we can put two same class name inside different namespace.
- We have following types of namespace declaration formats
    - Block level namespace
    - File Scoped namespace
    - Nested namespce

### 3.3.1 BLOCK LEVEL NAMESPACE:

- In this namespace have scope inside two parentheses.

```
namespace Namespace1
{
```

```
    class Program
    {
        //Code
    }
}
```

### 3.3.2 File scoped level namespace:

- It have file level scope.
- It support C# 10 or above.

```
namespace Namespace;
class Program
{
    //Code
}
```

### 3.3.3 Nested namespace:

- Namespace inside namespace is called nested namespace.

```
namespace OuterNamespace
{
    class Program
    {
        //Code
    }
    namespace InnerNamespace
    {
        class InnerClass
        {
            //Code
        }
    }

}
```

## 3.4 The "using" Keyword

- If we want to import some namespace inside the program, then we use **"using"** keyword.
- For Ex : We want to print something, so We are use "Console.Writeline();"
- Here is the "Console" is the class which is contain "System" namespace.

- So we need to import “System” namespace inside the Program.
- So using “using” keyword we can import “System” namespace inside the Program.

```
using System;

namespace HelloWorldAPP
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

## 3.5 CLASS

- Class is a user define type which is describe how certain object look.
- In C# we are use “class” key word to define any class.
- Class have a property and behavior.
- Class contain following members
  - methods
  - variables
- Structure of class:

```
[Access Specifier] class [ClassName]
{
    //Code
}
```

### 3.5.1 NAMING CONVENTIONS OF CLASS:

- It should begin with an alphabet.
- It should being Pascal casing.
- There may be more than one alphabet, but without any spaces between them.
- Digits may be used but only after alphabet.
- No special symbol can be used except the underscore (_) and currency ($) symbol. When multiple words are needed, an underscore should separate them.
- No keywords or command can be used as a variable name.

### 3.5.2 Access Specifiers:

- In a C# we have a following assess specifiers

| Caller's location | public | protected internal | protected | internal | private protected | private |
|---|---|---|---|---|---|---|
| **Within the class** | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| **Derived class (same assembly)** | ✔ | ✔ | ✔ | ✔ | ✔ | ✘ |
| **Non-derived class (same assembly)** | ✔ | ✔ | ✘ | ✔ | ✘ | ✘ |
| **Derived class (different assembly)** | ✔ | ✔ | ✔ | ✘ | ✘ | ✘ |
| **Non-derived class (different assembly)** | ✔ | ✘ | ✘ | ✘ | ✘ | ✘ |

### 3.5.3 Nested Classes:

- Class inside the class called nested class.

```
namespace HelloWorldAPP
{
    class Program
    {
        class InnerClass
        {
            //Code
        }

        //Code
    }
}
```

## 3.6 Methods

- Method is a block code which is contain a series of statement.
- Method declare inside class, struct or interface.

```
[Access Specifier] [return datatype] [Method Name]([datatype of args] args)
{
    //Code
    return [return member]

}
```

## 3.7 VARIABLES

- It is used to represent storage locations.

| Type | Example |
|---|---|
| **Integral types** | sbyte, byte, short, ushort, int, uint, long, ulong, and char |
| **Floating point types** | float and double |
| **Decimal types** | decimal |
| **Boolean types** | true or false values, as assigned |
| **Nullable types** | Nullable data types |

```
[datatype] [name of variable]
```

- For Ex:

```
class A
{
    public static int x;
    int y;

    void F(int[] v, int a, ref int b, out int c)
    {
        int i = 1;
        c = a + b++;
    }
}
```

# UNDERSTANDING C# PROGRAM

## 4.1 PROGRAM FLOW

- Let us understand the flow of hello world program.

```
using System;

namespace HelloWorldAPP
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

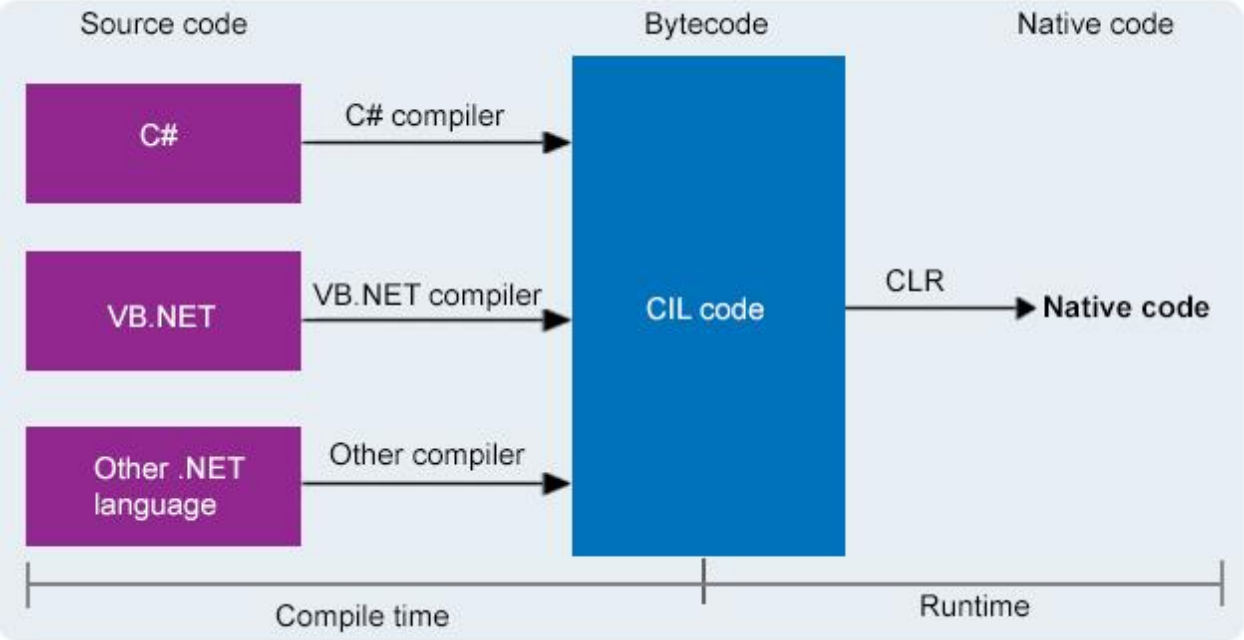
- .Net compiler platform also known as Roslyn.
- It is open source compiler and code analysis API for C# and VB.
- Compiler enter in program from **Main** method.
- Compiler follow code execution process.
- We can use different way to compile program.
    - Using Visual Studio
    - Using Command Line
        - For Ex : **csc Program.cs** and enter (if unsafe code in program then use **csc /unsafe Program.cs**)
        - Then **Program** enter
- Program compile using CLR.

## 4.2 UNDERSTANDING SYNTAX

- First of all we import some namespace using “using” keyword.
- Then we define namespace of program
- Then we should define class of program which contain **Main** method.
- Then inside class we should define **Main** method.
- **Main** method must static.
- Under the **Main** method we should write program.
- **Main** method return nothing and take array of string as an argument.

## 4.3 Common Language Runtime (CLR)

- It is a runtime environment.
- It manages life cycle and execution of .Net.
- It also provide easy service for deployment.
- Developer should write program in any language like C#, VB or F#, It convert into intermediate language.
- Intermediate language generated code which is in byte form is converted into native code using CLR



# WORKING WITH CODE FILES, PROJECTS & SOLUTIONS

## 5.1 STRUCTURE OF SOLUTION

- A project is contained within a solution.
- It's simply a container for one or more related projects, along with build information, Visual Studio window settings, and any miscellaneous files that aren't associated with a particular project.
- Visual Studio uses two file types (.sln and .suo) to store settings for solutions:

| Extension | Name | Description |
|---|---|---|
| **.sln** | Visual Studio Solution | Organizes projects, project items, and solution items in the solution. |
| **.suo** | Solution User Option | Stores user-level settings and customizations, such as breakpoints. |

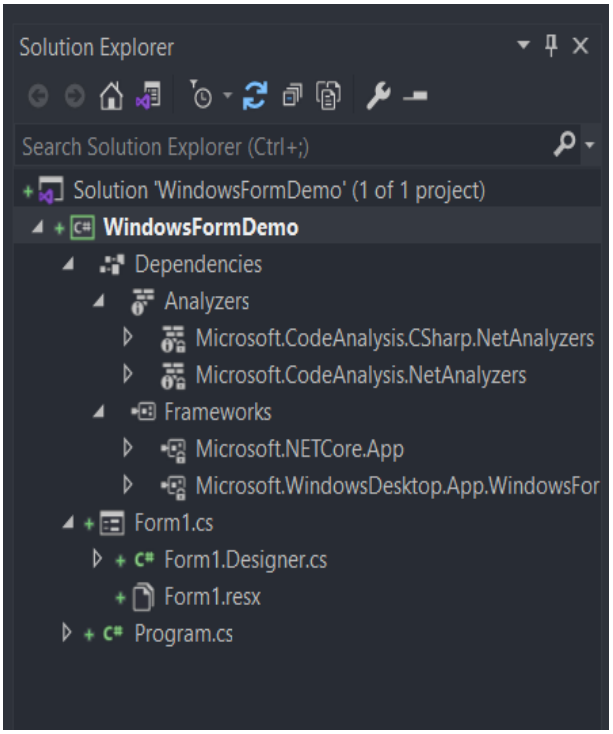
- A "solution folder" is a virtual folder that's only in Solution Explorer, where you can use it to group projects in a solution.
- If you want to locate a solution file on a computer, go to Tools > Options > Projects and Solutions > Locations.

## 5.2 UNDERSTANDING STRUCTURE OF PROJECT

- We have two commen folder which is exist in all projects of .Net
- bin :
  - The bin folder holds binary files, which are the actual executable code for your application or library.
- obj :
  - The obj folder holds object, or intermediate, files, which are compiled binary files that haven't been linked yet.
  - They're essentially fragments that will be combined to produce the final executable.
  - The compiler generates one object file for each source file, and those files are placed into the obj folder.

- Each of these folders are further subdivided into Debug and Release folders, which simply correspond to the project's build configurations.
- Dependency :
  - It contain dependency of project like framework and NuGet packages and other things which is helps to create and run .Net project.
- .csproj file :
  - It tells dotnet how to build the application.
  - It's one of the most important files in an project.
- .csproj.user :
  - csproj.user files Is made for storing your settings in your VS Project, can be understood as personal settings.

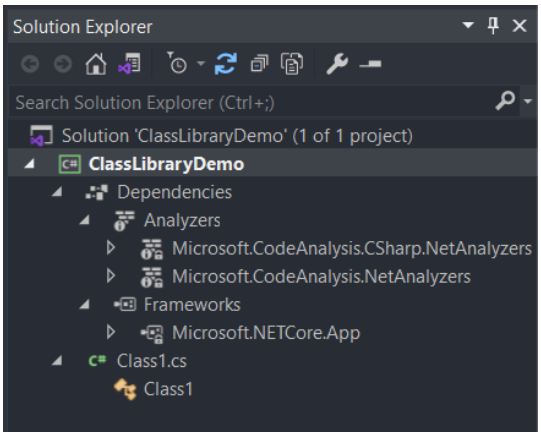
### 5.2.1 Windows Forms App:



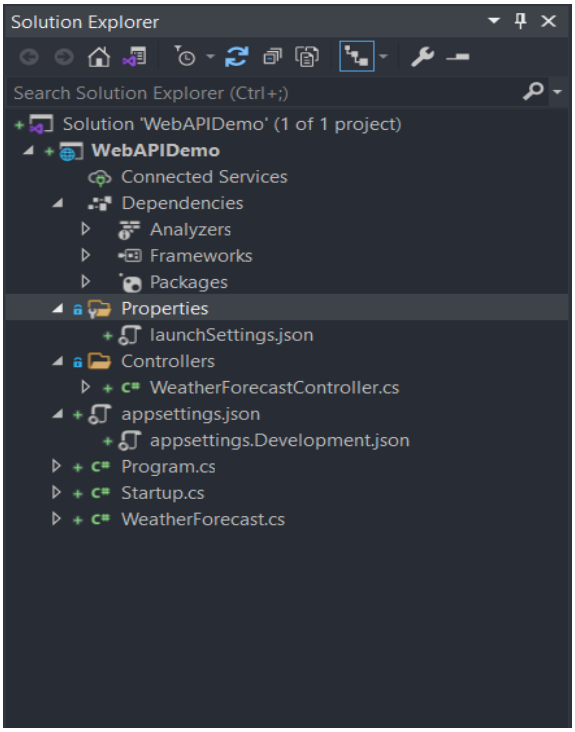

- Program.cs is a entry point of application.
- This will execute the first when application run.
- It also contain dependency injections and filters which is used in projects.
- Forms: It have two file one is Form1.cs which is used to develop UI and bind business logic with UI.
- Second is Form1.Designer.cs which is contain any control existing on the ,Form1.cs markup page is represented here. Most important are the name and type of the control.
- Form1.resx which is consists of XML entries, which specify objects and strings inside XML tags.
- It contains a standard set of header information, which describes the format of the resource entries and specifies the versioning information for the XML used to parse the data.

### 5.2.2 Class Library:

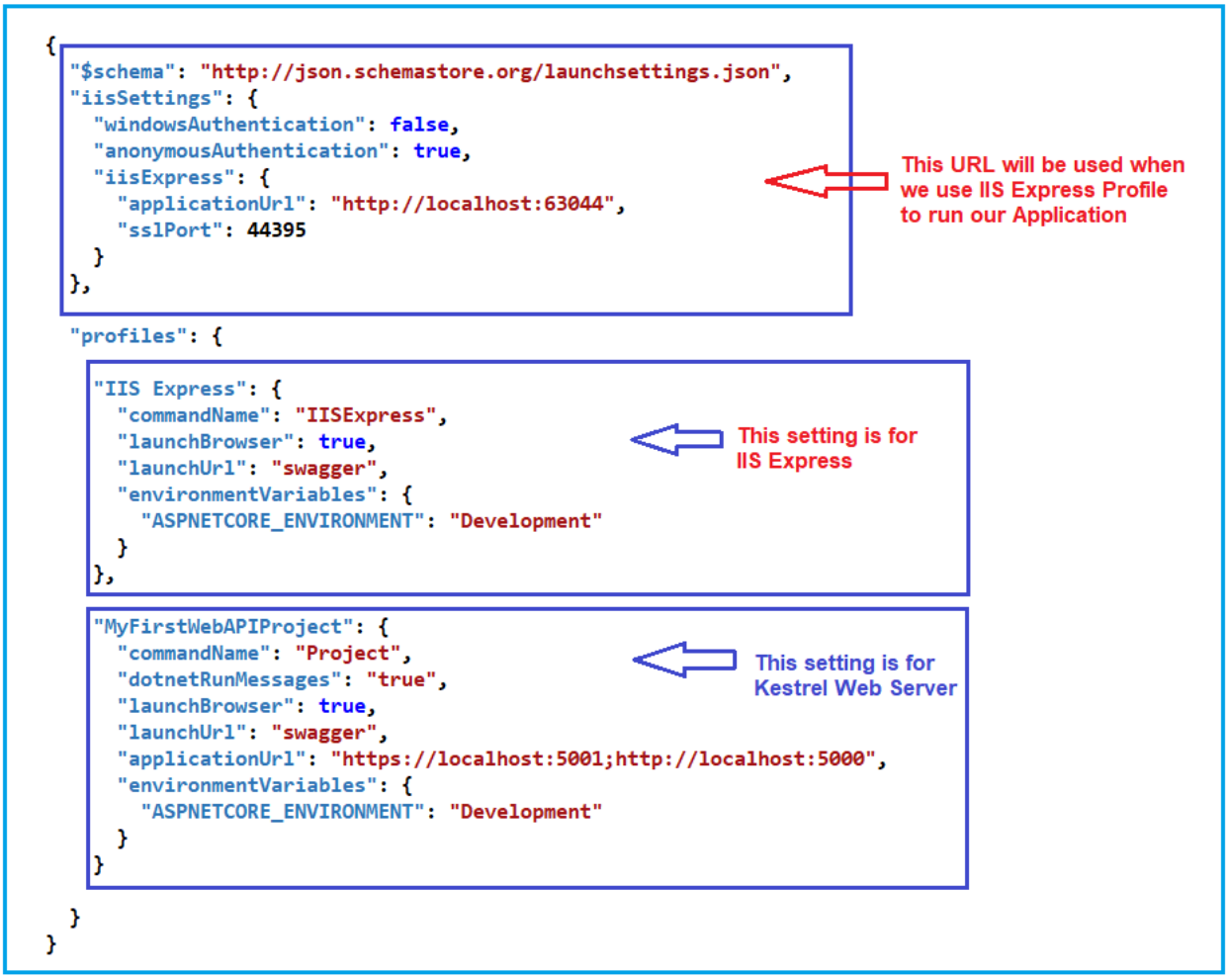
- It is used to create class library, so we should improve reusability of our project.
- It contain C# file which is only class file which contain different type of logic.
- We can attach or detach this class library in different projects.




### 5.2.3 Web API:




- **Properties:** The Properties Folder in ASP.NET Core Web Application by default contains one JSON file called as launchSettings.json file as shown in the below image.
- **launchSetting.json:** The launchsettings.json file contains some settings that are going to be used by .NET Core when we run the application either from Visual Studio or by using .NET Core CLI.
- The launchSettings.json file is only used within the local development machine. So, this file is not required when we publishing our ASP.NET Core Web API application into the production server.
- Now, open the launchSettings.json file, by default you will see the following settings.
- If you are running your application from the visual studio then IIS Express Profile will be used (for HTTP the port number will be 63044 and for HTTPS the port number will be 44395).
- if you are running your application using .NET Core CLI, then WebAPIDemo profile will be used which is nothing but using Kestrel Web Server and for HTTP protocol it uses the port number 5000 and for HTTPS protocol it uses the port number 5001.s




**Controllers:**

- The ASP.NET Core Web API is a controller-based approach.
- All the controllers of your ASP.NET Core Web API Application should and must reside inside the Controllers folder.
- It contain business logic of Web API, It handles all requests and give corresponding response.
- This File inherited from **"Controller"** class or **"ControllerBase"** class.
- **"Controller"** class is derived from the **"ControllerBase"** class.
- **"Controller"** should use when we want to return View from **"ActionMethod"**, Because when we use **"ControllerBase"** class then we can't render View.

**appsettings.json file:**

- This is the same as web.config or app.config of our traditional .NET Application.

- The appsettings.json file is the application configuration file in ASP.NET Core Web Application used to store the configuration settings such as database connections strings, any application scope global variables, etc.

**appsettings.Development.json:**

- If you want to configure some settings based on the environments then you can do such settings in appsettings.{Environment}.json file.
- You can create n number of environments like development, staging, production, etc.
- If you set some settings in the appsettings.Development.json file, then such settings can only be used in the development environment, can not be used in other environments.

**Program.cs:**

- It has a public static void Main() method.
- The Main method is the entry point of our Application.
- Each ASP.NET Core Web API Application initially starts as a Console Application and the Main() method is the entry point to the application.
- So, when we execute the ASP.NET Core Web API application, it first looks for the Main() method and this is the method from where the execution starts for the application.
- The Main() method then configures ASP.NET Core and starts it. At this point, the application becomes an ASP.NET Core Web API application.

**Startup.cs:**

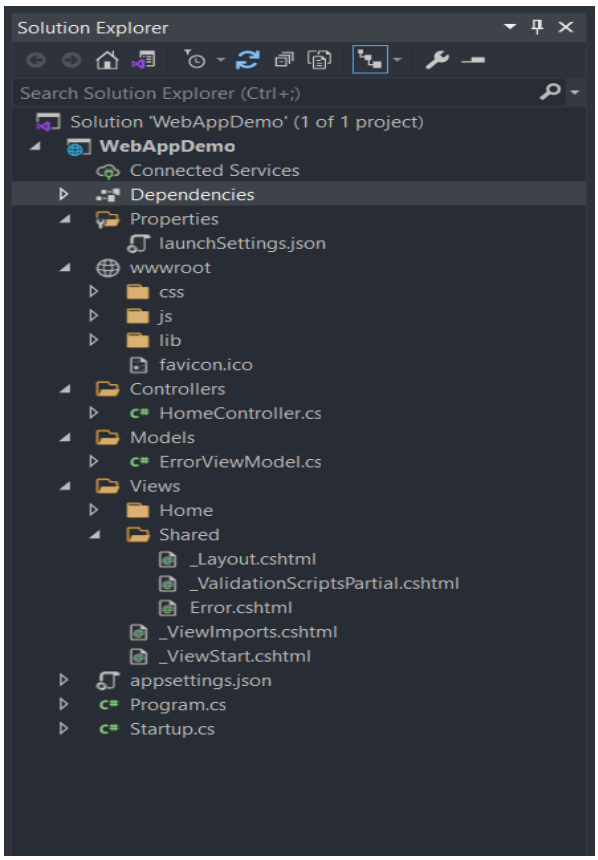
- The Startup class is like the Global.asax file of our traditional .NET application.
- As the name suggests, it is executed when the application starts.
- Startup class includes two public methods:
  - ConfigureServices
  - Configure
- ConfigureServices():
  - The ConfigureServices method of the Startup class is the place where we can register our dependent classes with the built-in IoC container.
  - Once we register the dependent classes, they can be used anywhere within the application.
  - The ConfigureServices method includes the IServiceCollection parameter to register services to the IoC container.
- Configure:

- The Configure method of the Startup class is the place where we configure the application request pipeline using the IApplicationBuilder instance that is provided by the built-in IoC container.
- ASP.NET Core introduced the middleware components to define a request pipeline, which will be executed on every request.

- If you look at the Configure method, it registered UseDeveloperExceptionPage, UseSwagger, UseSwaggerUI, UseHttpsRedirection, UseRouting, UseAuthorization, and UseEndpoints middleware components to the request processing pipeline.
- After coming .Net 6 **"Startup.cs"** file was removed and combine the functionality with the **"Program.cs"** file.
- So now after .Net 6 we have only **"Program.cs"**.

### 5.2.4 Web Application:

- Web application contain all folder and files which have a web api and it have additional file and folders.
- **wwwroot:** The **"wwwroot"** folder in the ASP.NET Core project is treated as a web root folder.
- Static files can be stored in any folder under the web root and accessed with a relative path to that root.
- **Libmen.json:** This file contains the list of libraries for static file to download.
- Each library has a name, a version, a list of files to download, and the location where the file will be copied.
- **Models:** The Models folder of an ASP.NET Core MVC application contains the class files which are used to store the domain data (you can also say business data) as well as business logic to manage the data.
- **View:** The Views Folder of an ASP.NET Core MVC application contains all the ".cshtml" files of your application.
- In MVC, the .cshtml file is a file where we need to write the HTML code along with the C# code.



- The Views folder also includes separate folders for each and every controller for your application.
- For example: all the .cshtml files of the HomeController will be in the View => Home folder.
- We also have the Shared folder under the Views folder.
- The Shared Folder contains all the views which are needed to be shared by different controllers.
- For example: error files,  layout files, etc.

# UNDERSTANDING DATATYPES & VARIABLES WITH CONVERSION

## 6.1 DATA TYPE:

- It specify type of data.
- In C# we have following types of datatypes
  - Value Data Type
  - Reference Data Type
  - Pointer Data Type

### 6.1.1 VALUE DATA TYPE:

- It directly store value in a memory.
- It accepts both signed and unsigned literals.
- It belongs to **System.ValueType** namespace.

**Signed and Unsigned Integer Data Type:**

| Alias | Type Name | Type | Size(bits) | Range | Default Value |
|---|---|---|---|---|---|
| **sbyte** | System.Sbyte | signed integer | 8 | -128 to 127 | 0 |
| **short** | System.Int16 | signed integer | 16 | -32768 to 32767 | 0 |
| **Int** | System.Int32 | signed integer | 32 | $-2^{31}$ to $2^{31}-1$ | 0 |
| **long** | System.Int64 | signed integer | 64 | $-2^{63}$ to $2^{63}-1$ | 0L |
| **byte** | System.byte | unsigned integer | 8 | 0 to 255 | 0 |
| **ushort** | System.UInt16 | unsigned integer | 16 | 0 to 65535 | 0 |
| **uint** | System.UInt32 | unsigned integer | 32 | 0 to $2^{32}$ | 0 |
| **ulong** | System.UInt64 | unsigned integer | 64 | 0 to $2^{63}$ | 0 |

**Floating Point Type:**

| Alias | Type name | Size(bits) | Range (aprox) | Default Value |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| **float** | System.Single | 32 | $\pm1.5 \times 10^{-45}$ to $\pm3.4 \times 10^{38}$ | 0.0F |
| **double** | System.Double | 64 | $\pm5.0 \times 10^{-324}$ to $\pm1.7 \times 10^{308}$ | 0.0D |

**Character Type:**

| Alias | Type name | Size In(Bits) | Range | Default value |
|---|---|---|---|---|
| **char** | System.Char | 16 | U +0000 to U +ffff | '\0' |

**Boolean Type:**

| Alias | Type name | Values |
|---|---|---|
| **bool** | System.Boolean | True / False |

### 6.1.2 Reference Data Type:

- The reference data type will contain memory address of data because reference don't store value of data directly.
- The built in reference type are…
  - String
  - Object

**String:**

- It represents a sequence of Unicode characters and its type name is **System.String**. So, string and String are equivalent.

```
string s1 = "Dhruvil"; // creating through string keyword
String s1 = "Dobariya"; // creating through String class
```

**Object:**

- In C#, all types, predefined and user-defined, reference types and value types, inherit directly or indirectly from Object.
- So basically it is the base class for all the data types in C#.
- Before assigning values, it needs type conversion.
- **Boxing:** When a variable of a value type is converted to object, it's called boxing.

- **Unboxing:** When a variable of type object is converted to a value type, it's called unboxing.
- Its type name is **System.Object**.

```
using System;

namespace DatatypeDemo
{
    class BoxingUnboxing
    {
        static void Main(string[] args)
        {
            //Intialization
            int a = 10;
            Console.WriteLine(a.GetType());

            //Boxing
            Object boxingA = a;
            Console.WriteLine(boxingA.GetType());

            //Unboxing
            int unboxingA = (int)(boxingA);
            Console.WriteLine(unboxingA.GetType());

        }
    }
}
```

```
Output :

System.Int32
System.Int32
System.Int32
```

- **GetType()** method give datatype of variable.

### 6.1.3 Pointer Datatype:

- The Pointer Data Types will contain a memory address of the variable value.
- To get the pointer details we have a two symbols
  - ampersand (&)
  - asterisk (*).

- Ampersand (&): It is Known as Address Operator. It is used to determine the address of a variable.
- Asterisk (*): It also known as Indirection Operator. It is used to access the value of an address.

**Syntax:**

```
type* identifier;
```

**Example:**

```
int* p1, p;   // Valid syntax
int *p1, *p;   // Invalid
```

**Program:**

```
// Error: Unsafe code requires the `unsafe'
// command line option to be specified
// For its solution:
// Go to your project properties page and
// check under Build the checkbox Allow
// unsafe code.
using System;

namespace DatatypeDemo
{
    class Pointer
    {
        static void Main(string[] args)
        {
            unsafe
            {

                // declare variable
                int n = 10;

                // store variable n address
                // location in pointer variable p
                int* p = &n;
                Console.WriteLine("Value :{0}", n);
                Console.WriteLine("Address :{0}", (int)p);
            }
        }
```

```
    }
}
```

```
Output :

Value :10
Address :-805837108
```

## 6.2 DATATYPE CONVERSION

- We have two type of technique to type conversion
  - Implicit type conversion
  - Explicit type conversion

**Implicit type conversion:**

- These conversions are performed by C# in a type-safe manner.
- For example: conversions from smaller to larger integral types and conversions from derived classes to base classes.

**Explicit type conversion:**

- These conversions are done explicitly by users using the pre-defined functions.
- Explicit conversions require a cast operator.

```
using System;

namespace DatatypeDemo
{
    class TypeConversion
    {
        static void Main(string[] args)
        {
            //Initialization
            int a = 20;
            Console.WriteLine(a);
            Console.WriteLine(a.GetType());

            //Implicit conversion
            double implicitlyA = a;
            Console.WriteLine(implicitlyA);
```

```
            Console.WriteLine(implicitlyA.GetType());

            //Explicit conversion
            int explicitA = (int)(implicitlyA);
            Console.WriteLine(explicitA);
            Console.WriteLine(explicitA.GetType());

        }
    }
}
```

```
Output :

20
System.Int32
20
System.Double
20
System.Int32
```

**Data Type Conversion Method:**

| Sr.No. | Methods & Description |
|---|---|
| 1 | **ToBoolean**<br>Converts a type to a Boolean value, where possible. |
| 2 | **ToByte**<br>Converts a type to a byte. |
| 3 | **ToChar**<br>Converts a type to a single Unicode character, where possible. |
| 4 | **ToDateTime**<br>Converts a type (integer or string type) to date-time structures. |
| 5 | **ToDecimal**<br>Converts a floating point or integer type to a decimal type. |

| | |
|---|---|
| 6 | **ToDouble**<br>Converts a type to a double type. |
| 7 | **ToInt16**<br>Converts a type to a 16-bit integer. |
| 8 | **ToInt32**<br>Converts a type to a 32-bit integer. |
| 9 | **ToInt64**<br>Converts a type to a 64-bit integer. |
| 10 | **ToSbyte**<br>Converts a type to a signed byte type. |
| 11 | **ToSingle**<br>Converts a type to a small floating point number. |
| 12 | **ToString**<br>Converts a type to a string. |
| 13 | **ToType**<br>Converts a type to a specified type. |
| 14 | **ToUInt16**<br>Converts a type to an unsigned int type. |
| 15 | **ToUInt32**<br>Converts a type to an unsigned long type. |
| 16 | **ToUInt64**<br>Converts a type to an unsigned big integer. |

```
using System;

namespace DatatypeDemo
{
    class DataTypeConversionMethod
    {
        static void Main(string[] args)
        {
            //Initialization
            int a = 20;
            double b = 21.20;
            bool c = true;
            Console.WriteLine("On Initialization");
            Console.WriteLine("Type of a: " + a.GetType());
            Console.WriteLine("Type of b: " + b.GetType());
            Console.WriteLine("Type of c: " + c.GetType());
            Console.WriteLine();

            //Covert into string
            Console.WriteLine("After Conversation in String");
            string stringA = a.ToString();
            string stringB = b.ToString();
            string stringC = c.ToString();

            Console.WriteLine("Type of stringA: " + stringA.GetType());
            Console.WriteLine("Type of stringB: " + stringB.GetType());
            Console.WriteLine("Type of stringC: " + stringC.GetType());
            Console.WriteLine();

            //Convert into main datatype
            Console.WriteLine("After Conversation in main datatype");

            Console.WriteLine("Type of stringA: " +
Convert.ToInt32(stringA).GetType());
            Console.WriteLine("Type of stringB: " +
Convert.ToDouble(stringB).GetType());
            Console.WriteLine("Type of stringC: " +
Convert.ToBoolean(stringC).GetType());

        }
    }
}
```

```
Output :

On Initialization
Type of a: System.Int32
Type of b: System.Double
Type of c: System.Boolean

After Conversation in String
Type of stringA: System.String
Type of stringB: System.String
Type of stringC: System.String

After Conversation in main datatype
Type of stringA: System.Int32
Type of stringB: System.Double
Type of stringC: System.Boolean
```

# UNDERSTANDING DECISION MAKING & STATEMENTS

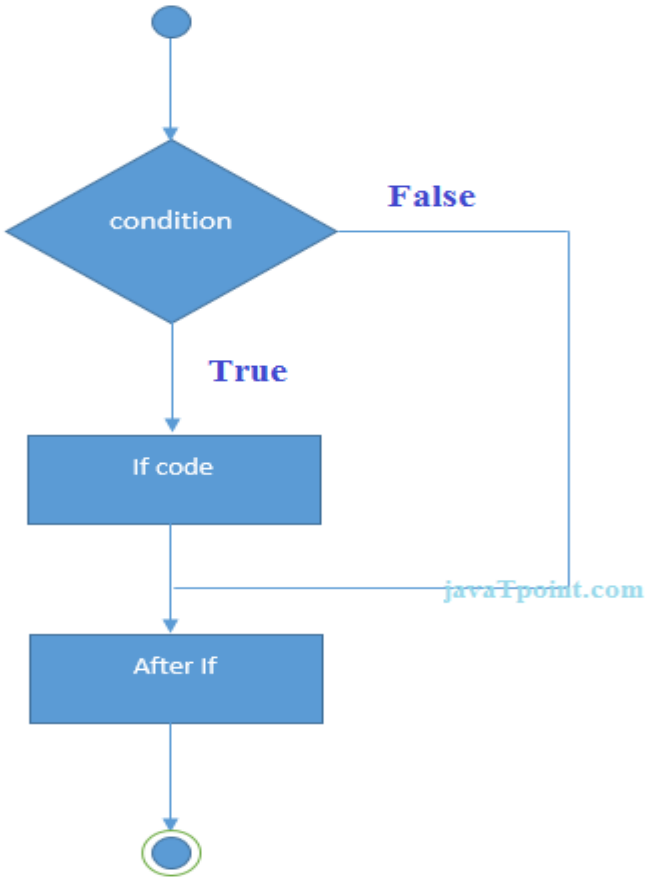
## 7.1 IF ELSE

- We have various type of if else statement:
  - if statement
  - if else statement
  - nested if statement
  - if else ladder

### 7.1.1 IF STATEMENT:

- C# test the condition, if condition is true then execute body of if.

**Flowchart:**



**Syntax:**

```
if(condition){
    // code...
}
```

**Program:**

```
using System;

namespace DecisionMaking
{
    class IfDemo
    {
        static void Main(string[] args)
        {
            int a = 10;
            if(a%2 == 0)
            {
                Console.WriteLine(a + " is a even");
            }
        }
    }
}
```

```
Output :

10 is a even
```

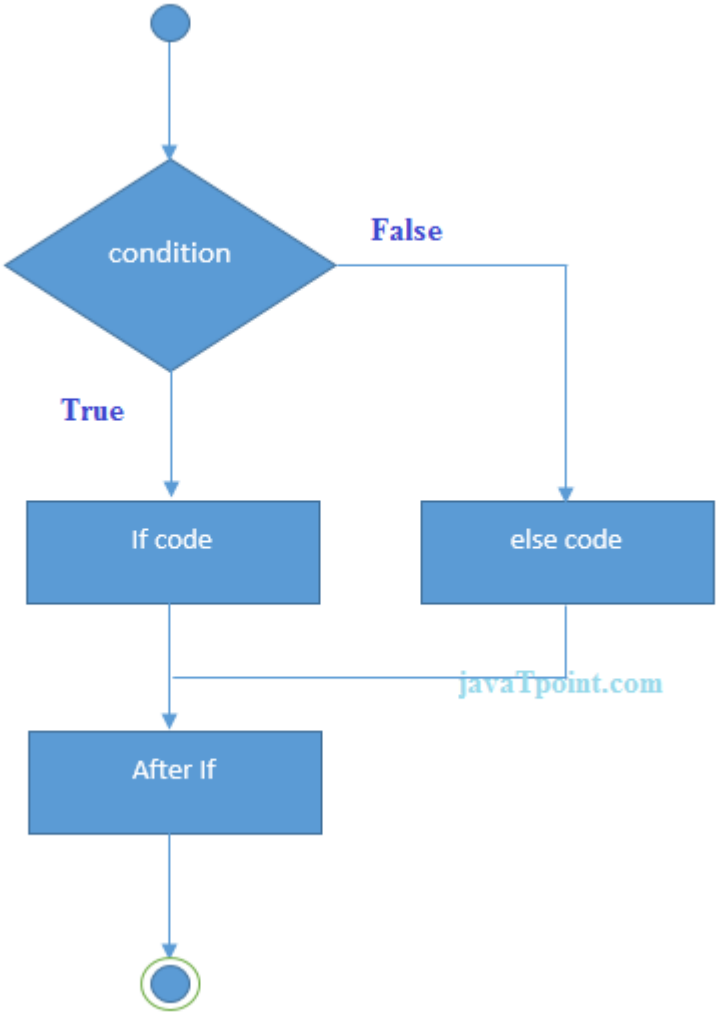
### 7.1.2 If Else Statement:

- C# test the condition if condition is true the execute body of **"if"** otherwise execute body of **"else"**.

**Syntax:**

```
if(condition){
    // code execute when condition is true
}
else{
    // code execute when condition is false
}
```

**Flowchart:**




**Program:**

```
using System;

namespace DecisionMaking
{
    class IfElseDemo
    {
        static void Main(string[] args)
        {
            int a = 21;
            if (a % 2 == 0)
            {
                Console.WriteLine(a + " is a even");
```

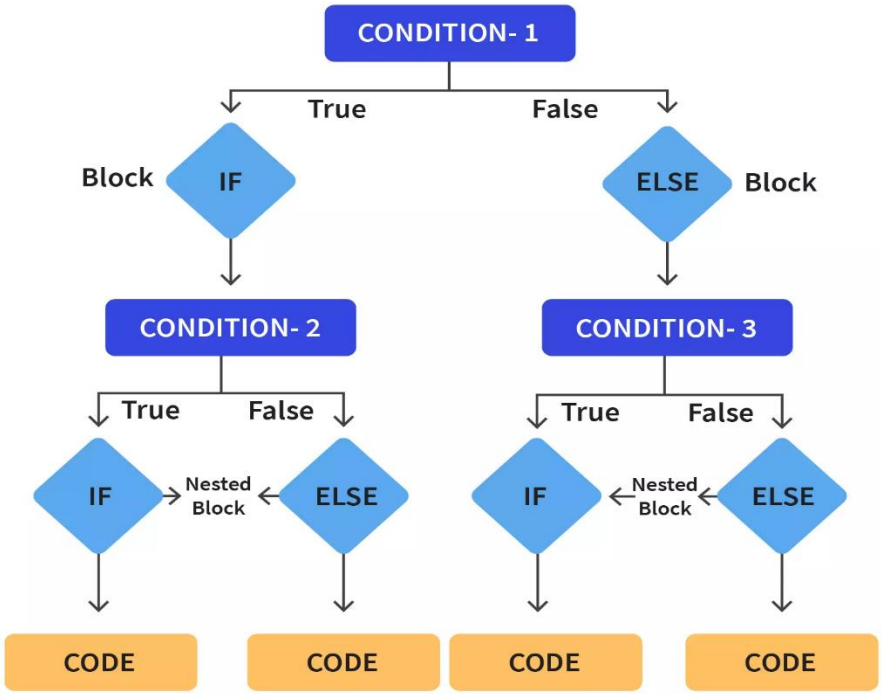
```
            }
            else
            {
                Console.WriteLine(a + " is a odd");
            }
        }
    }
}
```

```
Output :

21 is a odd
```

### 7.1.3 Nested If Statement:

- It is a nesting of if else statement.

**Flowchart:**




SCALER Topics

**Syntax:**

```
if(condition1){
    // code execute if condition1 is true
    if(condition2){
        // code execute if condition1 and condition2 both is true
    }
    else{
        // code execute if condition1 is true but condition2 if false
    }
}
else{
    // code execute if condition1 is false
    if(condition3){
        // code execute if condition1 is false and condition2 is true
    }
    else{
        // code execute if condition1 and condition2 both is false
    }
}
```

**Program:**

```
using System;

namespace DecisionMaking
{
    class NestedIFDemo
    {
        static void Main(string[] args)
        {
            int a = 1;
            int b = 2;
            int c = 3;

            if (a < b)
            {
                if (c < b)
                {
                    Console.WriteLine(b + " is grater then " + a + " and " +
c);
                }
                else
```

```
                {
                    Console.WriteLine(c + " is grater then " + a + " and " +
b);
                }
            }
            else
            {
                if (c < a)
                {
                    Console.WriteLine(a + " is grater then " + b + " and " +
c);
                }
                else
                {
                    Console.WriteLine(c + " is grater then " + a + " and " +
b);
                }
            }
        }
    }
}
```

```
Output :

3 is grater then 1 and 2
```

### 7.1.4 If Else Ladder Statement:

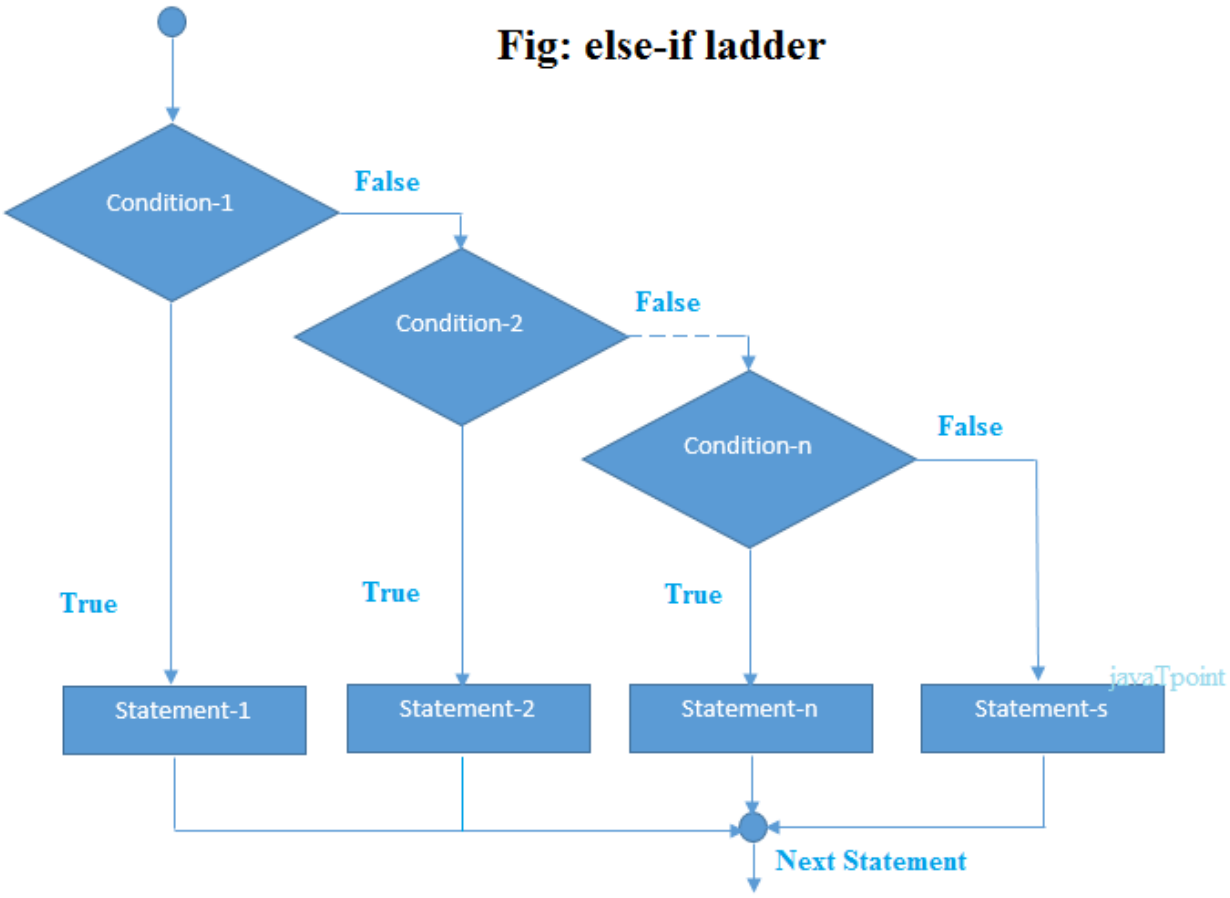
- C# test first condition, if it is true then execute it's body else it test second condition and this test run until one condition become true or reach the condition and execute **"else"** body.

**Syntax:**

```
if(condition1){
    // code execute if condition1 id true
}
else if(condition2){
    // code execute if condition2 is true
}
else if(conditionN){
    // code execute if conditionN is true.
}
```

```
else{
    // code execute if all condition id false.
}
```

**Flowchart:**




**Program:**

```
using System;

namespace DecisionMaking
{
    class IfElseLadderDemo
    {
        static void Main(string[] args)
        {
```

```
            int a = 1;
            int b = 2;
            int c = 3;

            if (a > b && a > c)
            {
                Console.WriteLine(a + " is grater then " + b + " and " + c);
            }
            else if (b > a && b > c)
            {
                Console.WriteLine(b + " is grater then " + a + " and " + c);
            }
            else
            {
                Console.WriteLine(c + " is grater then " + a + " and " + b);
            }
        }
    }
}
```

```
Output :

3 is grater then 1 and 2
```

## 7.2 SWITCH

- C# test one statement on multiple condition.
- It is similar like to if else ladder.
- It is a faster then if else ladder.
- Because, if switch contains more than five items, it's implemented using a lookup table or a hash list.
- This means that all items get the same access time.
- But in if else ladder where the last item takes much more time to reach as it has to evaluate every previous condition first.

**Syntax:**

```
switch(statement){
    case value1:
    // code execute if value1 and statement is same.
```

```
    break;

    case value2:
    // code execute if value2 and statement is same.
    break;

    case valueN:
    // code execute if valueN and statement is same.
    break;

    default:
    // code execute if no one match the statment value.
    break;
}
```

**Flowchart:**

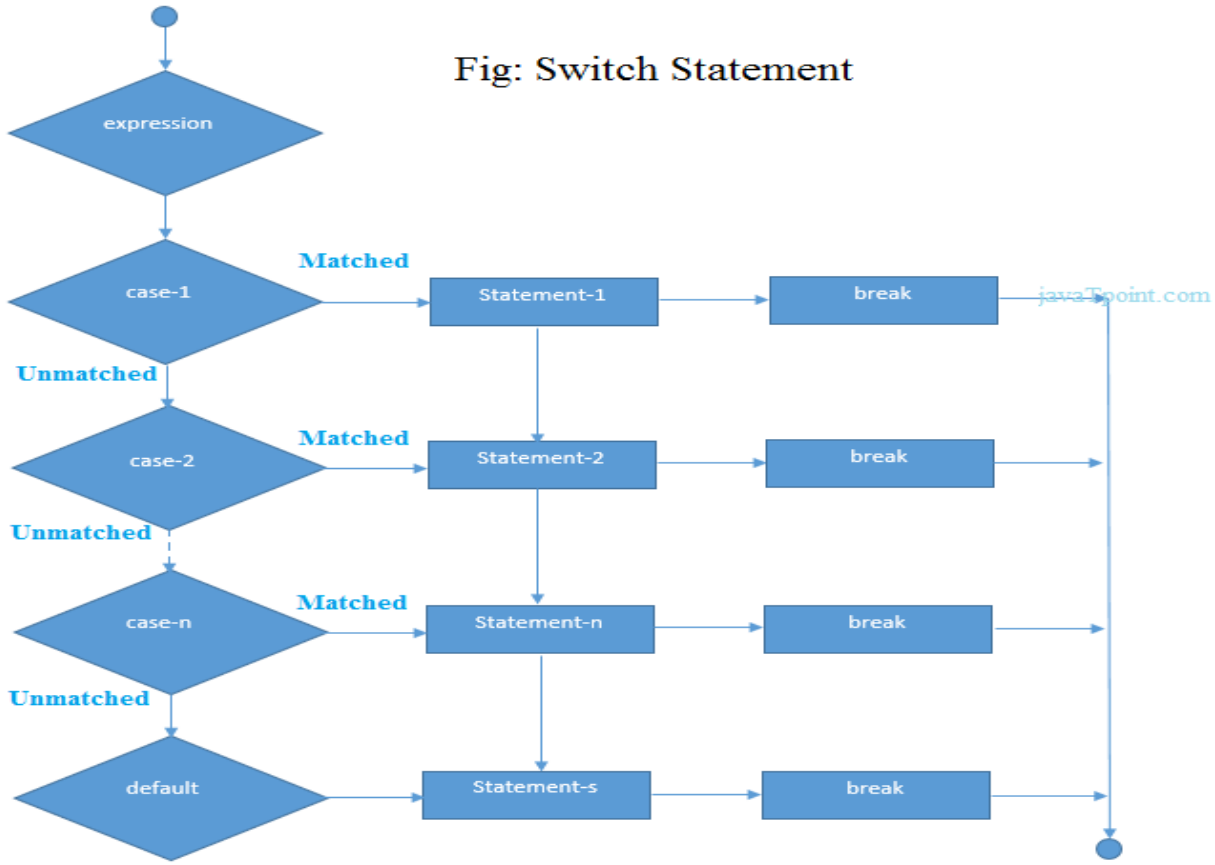



Fig: Switch Statement

**Program:**

```
using System;

namespace DecisionMaking
{
    class SwitchDemo
    {
        static void Main(string[] args)
        {
            int day = 4;

            switch (day)
            {
                case 1:
                    Console.WriteLine("Mon");
                    break;
                case 2:
                    Console.WriteLine("Tue");
                    break;
                case 3:
                    Console.WriteLine("Wed");
                    break;
                case 4:
                    Console.WriteLine("Thu");
                    break;
                case 5:
                    Console.WriteLine("Fri");
                    break;
                case 6:
                    Console.WriteLine("Sat");
                    break;
                case 7:
                    Console.WriteLine("Sun");
                    break;
                default:
                    Console.WriteLine("You entered wrong day");
                    break;

            }
        }
    }
}
```

```
Output :

Thu
```

# OPERATORS AND EXPRESSIONS

## 8.1 OPERATORS

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.
- C# have a following types of operators.
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Miscellaneous Operators

### 8.1.1 ARITHMETIC OPERATORS:

- Let say A = 10 and B = 20

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands | A + B = 30 |
| - | Subtracts second operand from the first | A - B = -10 |
| * | Multiplies both operands | A * B = 200 |
| / | Divides numerator by de-numerator | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division | B % A = 0 |
| ++ | Increment operator increases integer value by one | A++ = 11 |
| -- | Decrement operator decreases integer value by one | A-- = 9 |

### 8.1.2 COMPETITION / RELATION OPERATORS :

- Let say A = 10 and B = 20

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |

| | | |
|---|---|---|
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

### 8.1.3 Logical Operators:

- Let say A = true and B = false

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non zero then condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non zero then condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true. |

### 8.1.4 Bitwise Operator:

- Bitwise operator works on bits and perform bit by bit operation.

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) = 12, which is 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) = 61, which is 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) = 49, which is 0011 0001 |

| | | |
|---|---|---|
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) = -61, which is 1100 0011 in 2's complement due to a signed binary number. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 = 240, which is 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 = 15, which is 0000 1111 |

- For Example:
- Let say A = 60 and B = 13
- Then in a binary format A = 0011 1100 and B = 0000 1101

| p | q | p & q | p \| q | p ^ q |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

- So we got this:
- A & B = 0000 1100
- A | B = 0011 1101
- A ^ B = 0011 0001
- ~A = 1100 0011

### 8.1.5 Assignment Operators:

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator, Assigns values from right side operands to left side operand | C = A + B assigns value of A + B into C |

| | | |
|---|---|---|
| **+=** | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand | C += A is equivalent to C = C + A |
| **-=** | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand | C -= A is equivalent to C = C - A |
| ***=** | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand | C *= A is equivalent to C = C * A |
| **/=** | Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand | C /= A is equivalent to C = C / A |
| **%=** | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand | C %= A is equivalent to C = C % A |
| **<<=** | Left shift AND assignment operator | C <<= 2 is same as C = C << 2 |
| **>>=** | Right shift AND assignment operator | C >>= 2 is same as C = C >> 2 |
| **&=** | Bitwise AND assignment operator | C &= 2 is same as C = C & 2 |
| **^=** | bitwise exclusive OR and assignment operator | C ^= 2 is same as C = C ^ 2 |
| **\|=** | bitwise inclusive OR and assignment operator | C \|= 2 is same as C = C \| 2 |

### 8.1.6 Miscellaneous Operators:

| Operator | Description | Example |
|---|---|---|

| | | |
|---|---|---|
| **sizeof()** | Returns the size of a data type. | sizeof(int), returns 4. |
| **typeof()** | Returns the type of a class. | typeof(StreamReader); |
| **&** | Returns the address of an variable. | &a; returns actual address of the variable. |
| ***** | Pointer to a variable. | *a; creates pointer named 'a' to a variable. |
| **? :** | Conditional Expression | If Condition is true ? Then value X : Otherwise value Y |
| **is** | Determines whether an object is of a certain type. | If( Ford is Car) // checks if Ford is an object of the Car class. |
| **as** | Cast without raising an exception if the cast fails. | Object obj = new StringReader("Hello");<br>StringReader r = obj as StringReader; |

# LOOP ITERATION

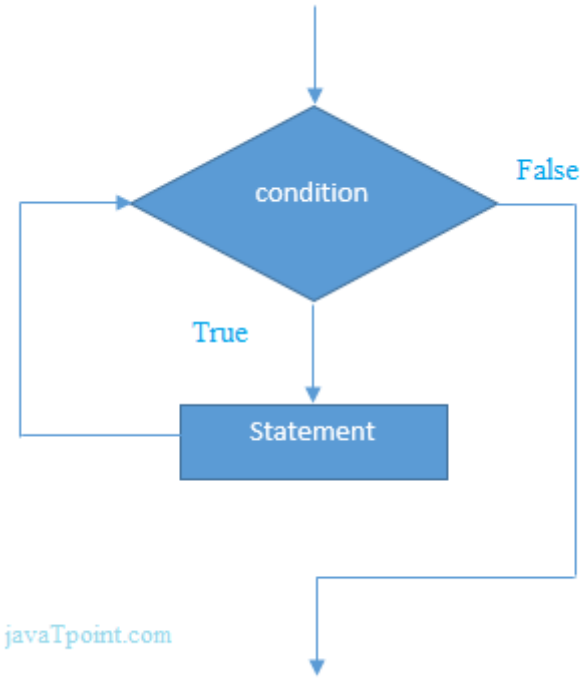
## 9.1 WHILE

- In C#, while loop is used to iterate a part of the program several times.
- If the number of iteration is not fixed, then we should use while loop.

**Syntax:**

```
while (condition)
{
    // code...
}
```

**Flowchart:**




**Program:**

```
using System;

namespace LoopIteration
{
    class WhileDemo
```

```
    {
        static void Main(string[] args)
        {
            int a = 1;

            while (a <= 10)
            {
                Console.WriteLine(a);
                a++;
            }
        }
    }
}
```

```
Output :

1
2
3
4
5
6
7
8
9
10
```

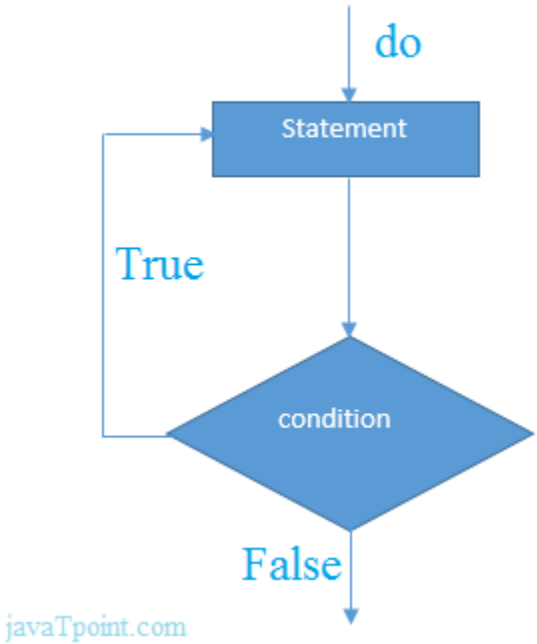
## 9.2 Do While

- In C#, while loop is used to iterate a part of the program several times.
- If the number of iteration is not fixed, then we should use while loop.
- It execute at least one time.
- Because it execute first and then check condition.

**Syntax:**

```
do
{
    // code...
} while (condition);
```

**Flowchart:**




**Program:**

```
using System;

namespace LoopIteration
{
    class DoWhileDemo
    {
        static void Main(string[] args)
        {
            int a = 1;

            do
            {
                Console.WriteLine(a);
                a++;
            } while (a <= 10);
        }
    }
}
```

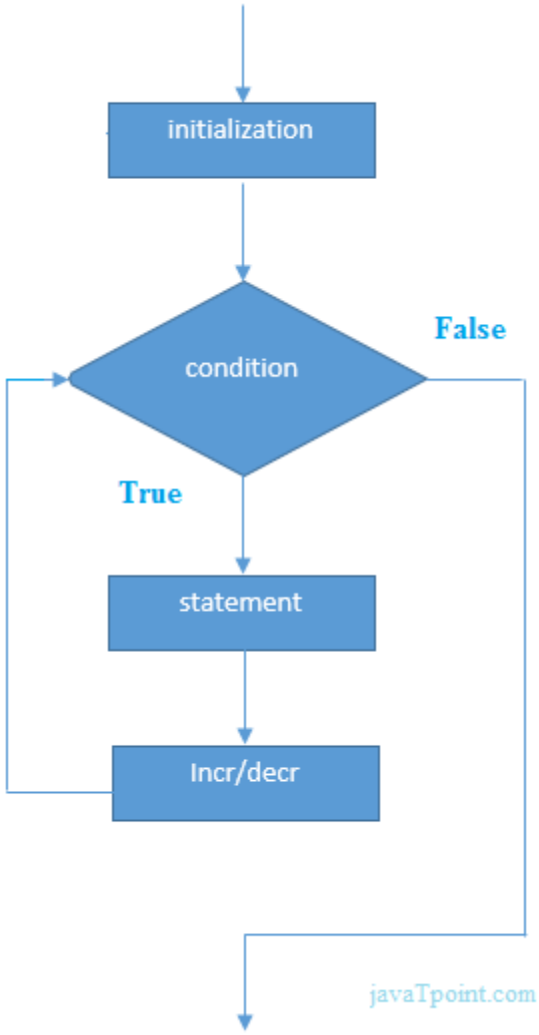
```
Output :

1
2
3
4
5
6
```

```
7
8
9
10
```

## 9.3 FOR

- The C# for loop is used to iterate a part of the program several times.
- If the number of iteration is fixed, then we should use for loop.

**Flowchart:**



**Syntax:**

```
for (initialization; condition; incriment/decrement)
{
    // code...
}
```

**Program:**

```
using System;

namespace LoopIteration
{
    class ForDemo
    {
        static void Main(string[] arge)
        {
            for (int i = 0; i <= 10; i++)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

```
Output :

1
2
3
4
5
6
7
8
9
10
```

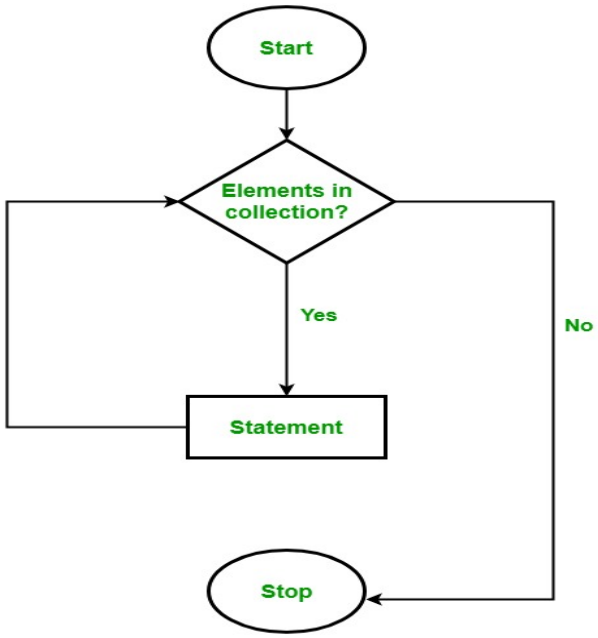
## 9.4 For Each

- In C#, while loop is used to iterate a part of the program several times.
- If the number of iteration is not fixed, then we should use while loop.
- It is high level language feature.

**Syntax:**

```
foreach (datatype element in collection)
{
    // code...
}
```

**Flowchart:**




**Program:**

```
using System;
using System.Collections.Generic;

namespace LoopIteration
{
    class ForEachDemo
    {
        static void Main(string[] args)
        {
            List<string> students = new List<string>() { "Dhruvil", "Dhaval",
"Bhargav", "Jenil", "Jash", "Dhruv", "Yash" };

            foreach (string std in students)
            {
                Console.WriteLine(std);
            }
        }
    }
}
```

```
Output :
```

```
Dhruvil
Dhaval
Bhargav
Jenil
Jash
Dhruv
Yash
```

## 9.5 Break Statement

- The C# break is used to break loop or switch statement.
- It breaks the current flow of the program at the given condition.
- In case of inner loop, it breaks only inner loop.

**Flowchart:**

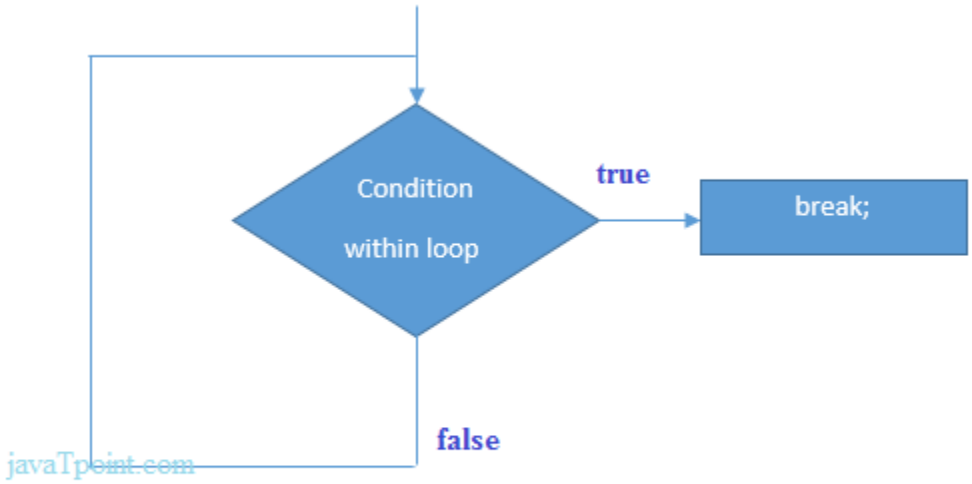



**Figure: Flowchart of break statement**

**Syntax:**

```
jump-statement;
break;
```

**Program:**

```
using System;

namespace LoopIteration
{
    class BreakDemo
    {
        static void Main(string[] args)
        {
            for (int i = 0; i <= 10; i++)
            {
                if (i == 5)
                {
                    break;
                }
                Console.WriteLine(i);
            }
        }
    }
}
```

```
Output :

0
1
2
3
4
```

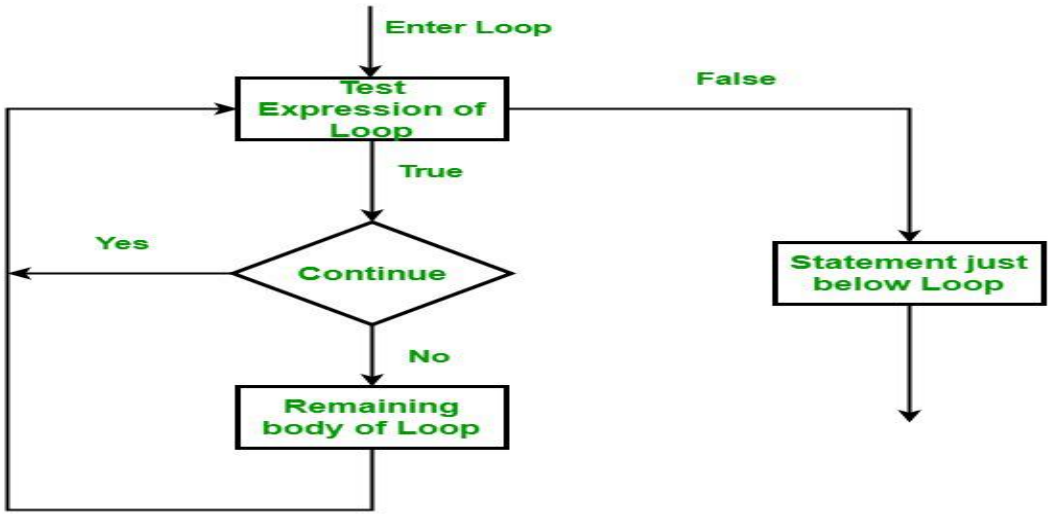
## 9.6 CONTINUE STATEMENT

- The C# continue statement is used to continue loop.
- It continues the current flow of the program and skips the remaining code at specified condition.
- In case of inner loop, it continues only inner loop.

**Syntax:**

```
jump-statement;
break;
```

**Flowchart:**




**Program:**

```
using System;

namespace LoopIteration
{
    class ContinueDemo
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 10; i++)
            {
                if (i % 2 != 0)
                {
                    continue;
                }
                Console.WriteLine(i);
            }
        }
    }
}
```

```
Output :

0
2
4
6
8
```

## 9.7 GO TO STATEMENT

- The C# goto statement is also known jump statement.
- It is used to transfer control to the other part of the program.
- It unconditionally jumps to the specified label.
- It can be used to transfer control from deeply nested loop or switch case label.
- Currently, it is avoided to use goto statement in C# because it makes the program complex.

**Program:**

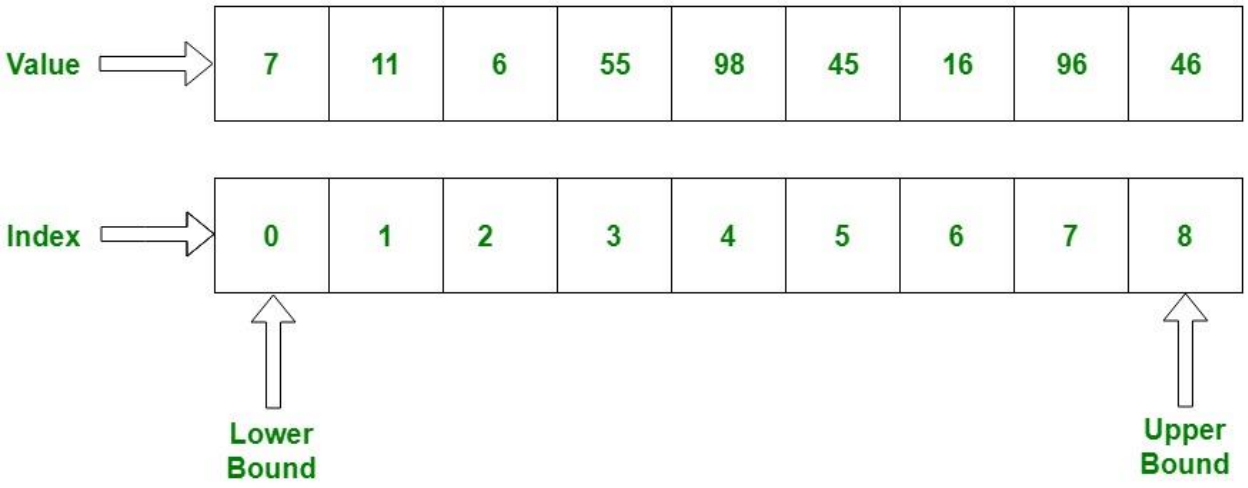
```
using System;

namespace LoopIteration
{
    class GoToDemo
    {
        static void Main(string[] args)
        {
            ineligible:
            Console.WriteLine("You are ineligible for voting!");

            Console.Write("Enter your age : ");
            int a = Convert.ToInt32(Console.ReadLine());

            if (a < 18)
            {
                goto ineligible;
            }
            else
            {
                Console.WriteLine("You are eligible for voting.");
            }
        }
    }
}
```

```
Output :

You are ineligible for voting!
Enter your age : 16
You are ineligible for voting!
Enter your age : 20
You are eligible for voting.
```

# UNDERSTANDING ARRAYS

## 10.1 INTRODUCTION

- Array is a sequential collection of same datatype variable.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.




- C# have three type of array.
  - Single dimensional array
  - Multidimensional array
  - Jugged array

## 10.2 DECLARING ARRAY

**Syntax:**

```
datatype[] array;
```

- Here, datatype is used to specify the type of elements in the array.
- [ ] specifies the rank of the array. The rank specifies the size of the array.

- array specifies the name of the array.
- Array is a reference type, so we need to use **new** keyword to create instance of array.

## 10.3 SINGLE DIMENSIONAL ARRAY

**Declaration:**

```
int[] array = new int[10];
```

**Initialization:**

- We should initialized array using three types.

```
namespace ArrayClass
{
    class InitializationSDArray
    {
        static void Main(string[] args)
        {
            // Method 1:
            // defining an array
            int[] array1 = new int[5];

            //initializing an array
            array1[0] = 1;
            array1[2] = 2;
            array1[3] = 3;
            array1[4] = 4;
            array1[5] = 5;

            // Method 2:
            // initialized all at declaration time
            int[] array2 = new int[5] { 1, 2, 3, 4, 5 };

            // Method 3:
            // declaring an array
            int[] array3;

            // initializing an array
            array3 = new int[5] { 1, 2, 3, 4, 5 };
        }
    }
}
```

**Accessing value:**

- We can accessing array using two types, using index of array and using foreach or any other loop.

```
using System;

namespace ArrayClass
{
    class AccessingValueSDArray
    {
        static void Main(string[] args)
        {
            int[] a = new int[5] { 1, 2, 3, 4, 5 };

            // Using index of array
            Console.WriteLine(a[0]);
            Console.WriteLine(a[1]);
            Console.WriteLine(a[2]);
            Console.WriteLine(a[3]);
            Console.WriteLine(a[4]);

            //using foreach loop
            foreach (int element in a)
            {
                Console.WriteLine(element);
            }
        }
    }
}
```

```
Output :

1
2
3
4
5
1
2
3
4
5
```

## 10.4 MULTIDIMENSIONAL ARRAY

- Array have more than one dimensional.
- It would be 2d array and 3d array and it would be nd array.

**Initialization:**

```
namespace ArrayClass
{
```

```
    class InitializationMDarray
    {
        static void Main(string[] args)
        {
            // Two dimensional array
            // Method 1:
            int[,] array1 = new int[,]
            {
                { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 }, { 9, 10 }
            };

            // Method 2:
            int[,] array2 = new int[5, 2]
            {
                { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 }, { 9, 10 }
            };


            // Three dimensional array
            // Method 1:
            int[,,] array3 = new int[,,]
            {
                { { 1, 2 }, { 3, 4 } },
                { { 5, 6 }, { 7, 8 } },
                { { 9, 10 }, { 1, 2 } }
            };

            // Method 2:
            int[,,] array4 = new int[3, 2, 2]
            {
                { { 1, 2 }, { 3, 4 } },
                { { 5, 6 }, { 7, 8 } },
                { { 9, 10 }, { 1, 2 } }
            };
        }
    }
}
```

**Accessing value:**

- We can accessing array using two types, using index of array and using foreach or any other loop.

```
using System;

namespace ArrayClass
{
    class AccessingValueMDArray
    {
        static void Main(string[] args)
        {
```

```
            int[,] array1 = new int[2, 4] { { 1, 2, 3, 4 }, { 5, 6, 7, 8 } };
            // here the length of array is 8

            // Method 1:
            Console.Write(array1[0, 1] + " ");
            Console.Write(array1[0, 0] + " ");
            Console.Write(array1[0, 2] + " ");
            Console.Write(array1[0, 3] + " ");

            Console.Write(array1[1, 0] + " ");
            Console.Write(array1[1, 1] + " ");
            Console.Write(array1[1, 2] + " ");
            Console.Write(array1[1, 3] + " ");

            Console.WriteLine();

            // Method 2:

            // using foreach

            foreach(int a in array1)
            {
                Console.Write(a + " ");
            }

            Console.WriteLine();
            // using for
            for (int i = 0; i <= array1.GetUpperBound(0); i++)
            {
                for (int j = 0; j <= array1.GetUpperBound(1); j++)
                {
                    Console.Write(array1[i, j] + " ");
                }
            }

            Console.WriteLine();
            // if we have 3d array then

            int[,,] array2 = new int[2, 2, 3]
            {
                { { 1, 2, 3 }, { 4, 5, 6 } },
                { { 7, 8, 9 }, { 0, 1, 2 } }
            };
            for (int i = 0; i <= array2.GetUpperBound(0); i++)
            {
                for (int j = 0; j <= array2.GetUpperBound(1); j++)
                {
                    for (int k = 0; k <= array2.GetUpperBound(2); k++)
                    {
                        Console.Write(array2[i, j, k] + " ");
                    }
                }
            }
        }
```

```
    }
}
```

```
Output :

2 1 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9 0 1 2
```

## 10.5 JUGGED ARRAY

- It is array of array.
- A jagged array is an array whose elements are arrays.
- It can be array of single array.
- It can be array of multidimensional array.

**Initialization:**

- We can initialize jugged array using three different methods.

```
namespace ArrayClass
{
    class InitializationJDArray
    {
        static void Main(string[] args)
        {
            // Method 1:
            int[][] array1 = new int[3][];

            array1[0] = new int[2] { 1, 2 };
            array1[1] = new int[2] { 3, 4 };
            array1[2] = new int[3] { 5, 6, 7 };

            // Method 2:
            int[][] array2 = new int[][]
            {
                new int[] { 1, 2 },
                new int[] { 3, 4 },
                new int[] { 5, 6, 7 }
            };

            // Method 3:
            int[][] array3 =
            {
                new int[] { 1, 2 },
                new int[] { 3, 4 },
                new int[] { 5, 6, 7 }
```

```
            };

            // Multidimensional array in jugged array
            int[][,] array4 =
            {
                new int[2,2] { { 1, 2 }, { 3, 4 } },
                new int[2,3] { { 1, 2, 3 }, { 4, 5, 6 } }
            };

        }
    }
}
```

**Accessing value:**

- We can access value of array using two method, one is using index of array and second is using loops.

```
using System;

namespace ArrayClass
{
    class AccessingValueJDArray
    {
        static void Main(string[] args)
        {
            int[][] array1 =
            {
                new int[] { 1, 2 },
                new int[] { 3, 4 },
                new int[] { 5, 6, 7 }
            };

            int[][,] array2 =
            {
                new int[,] { { 1, 2 }, { 3, 4 } },
                new int[,] { { 1, 2, 3 }, { 4, 5, 0 } }
            };

            // Method 1:
            Console.Write(array1[0][0] + " ");
            Console.Write(array1[1][0] + " ");
            Console.Write(array1[2][1] + " ");

            Console.Write(array2[0][0, 1] + " ");
            Console.Write(array2[1][1, 0] + " ");

            Console.WriteLine();

            // Method 2:
            foreach (int[] a1 in array1)
            {
                foreach (int a2 in a1)
```

```
                {
                    Console.Write(a2 + " ");
                }
            }

            Console.WriteLine();

            foreach (int[,] a3 in array2)
            {
                foreach(int a4 in a3)
                {
                    Console.Write(a4 + " ");
                }
            }
        }
    }
}
```

```
Output :

1 3 6 2 4
1 2 3 4 5 6 7
1 2 3 4 1 2 3 4 5 0
```

## 10.6 Array Class

- C# provides an Array class to deal with array related operations.
- It provides methods for creating, manipulating, searching, and sorting elements of an array.
- This class works as the base class for all arrays in the .NET Technology.

**Signature:**

- In C#, Array is not part of collection but considered as collection because it is based on the IList interface.

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public abstract class Array : ICloneable, IList, ICollection, IEnumerable,
IStructuralComparable, IStructuralEquatable
```

**Properties:**

| Sr.No. | Property & description |
|---|---|

| | |
|---|---|
| 1 | **IsFixedSize**<br>Gets a value indicating whether the Array has a fixed size. |
| 2 | **IsReadOnly**<br>Gets a value indicating whether the Array is read-only. |
| 3 | **Length**<br>Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array. |
| 4 | **LongLength**<br>Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array. |
| 5 | **Rank**<br>Gets the rank (number of dimensions) of the Array. |

**Methods:**

| Sr.No. | Methods & Description |
|---|---|
| 1 | **Clear**<br>Sets a range of elements in the Array to zero, to false, or to null, depending on the element type. |
| 2 | **Copy(Array, Array, Int32)**<br>Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer. |
| 3 | **CopyTo(Array, Int32)**<br>Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index. The index is specified as a 32-bit integer. |
| 4 | **GetLength** |

| | |
|---|---|
| | Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array. |
| 5 | **GetLongLength**<br>Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array. |
| 6 | **GetLowerBound**<br>Gets the lower bound of the specified dimension in the Array. |
| 7 | **GetType**<br>Gets the Type of the current instance. (Inherited from Object.) |
| 8 | **GetUpperBound**<br>Gets the upper bound of the specified dimension in the Array. |
| 9 | **GetValue(Int32)**<br>Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer. |
| 10 | **IndexOf(Array, Object)**<br>Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array. |
| 11 | **Reverse(Array)**<br>Reverses the sequence of the elements in the entire one-dimensional Array. |
| 12 | **SetValue(Object, Int32)**<br>Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer. |
| 13 | **Sort(Array)**<br>Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array. |

| | |
|---|---|
| 14 | **ToString**<br>Returns a string that represents the current object. (Inherited from Object.) |
| 15 | **Empty<T>()**<br>It is used to return an empty array. |

# DEFINING AND CALLING METHODS

## 11.1 INTRODUCTION

- A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.
- To use a method, you need to:
  - Define the method
  - Call the method

## 11.2 DEFINING THE METHOD

```
[Access Specifier] [return datatype] [Method Name]([datatype of args] args)
{
    //Code
    return [return member]

}
```

**Access Specifier:**

- This determines the visibility of a variable or a method from another class.

**Return type:**

- A method may return a value.
- The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.

**Method name:**

- Method name is a unique identifier and it is case sensitive.
- It cannot be same as any other identifier declared in the class.

**Parameter list:**

- Enclosed between parentheses, the parameters are used to pass and receive data from a method.
- The parameter list refers to the type, order, and number of the parameters of a method.
- Parameters are optional, It may possible that method don't contain any parameter.

**Method body:**

- This contains the set of instructions needed to complete the required activity.

**Program:**

```
amespace MethodDeclarationCalling
{
    public class MethodDemo
    {
        public static void Main(string[] args)
        {

        }

        public int Sum(int a, int b) // Method Declaration
        {
            return a + b;
        }
    }
}
```

## 11.3 Calling Method

- You can call a method using the name of the method.

**Program:**

```
using System;

namespace MethodDeclarationCalling
{
    class MethodDemo
    {
        static void Main(string[] args)
        {
            MethodDemo md = new MethodDemo(); // Create an instance of the
MethodDemo class.
            int ans = md.Sum(2, 3); // Calling the Method.
            Console.WriteLine(ans);
        }
        public int Sum(int a, int b) // Method Declaration
        {
            return a + b;
        }
    }
}
```

```
Output :
```

## 11.4 Passing Parameter to the method

- We should pass the three types of parameter.
  - Value Parameter
  - Reference Parameter
  - Output Parameter

### 11.4.1 Value Parameter:

- This is the default mechanism for passing parameters to a method.
- In this mechanism, when a method is called, a new storage location is created for each value parameter.
- The values of the actual parameters are copied into them.
- So, the changes made to the parameter inside the method have no effect on the argument.

### 11.4.2 Reference Parameter:

- Reference parameter is a reference to a memory location of a variable. \
- When we pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters.
- The reference parameters represent the same memory location as the actual parameters that are supplied to the method.
- You can declare the reference parameters using the **"ref"** keyword.
- So, the changes made to the parameter inside the method have effect on the argument.

### 11.4.3 Output Parameter:

- A return statement can be used for returning only one value from a function.
- However, using output parameters, you can return two values from a function.
- Output parameters are similar to reference parameters, except that they transfer data out of the method rather than into it.

# WORKING WITH STRINGS

## 12.1 INTRODUCTION

- In C#, a string is a sequence of Unicode characters or array of characters.
- The range of Unicode characters will be U+0000 to U+FFFF. The array of characters is also termed as the text.
- So the string is the representation of the text. A string is represented by a class **'System.String'**.
- The String class is defined in the .NET base class library. In other words, a String object is a sequential collection of **'System.Char'** objects which represent a string.
- The maximum size of the String object in memory can be 2GB or about 1 billion characters.
- Working with string in C#, We have a two class:
  - String
  - StringBuilder

## 12.2 STRING

- The **'System.String'** class is immutable, So it can't change it self, it should create a new instance.

**Constructor:**

| Sr.No. | Constructor & Description |
|---|---|
| 1 | **String(Char*)**<br>Arg 1: pointer of array of char |
| 2 | **String(Char*, Int32, Int32)**<br>Arg 1: pointer array of char, Arg 2: starting index, Arg 2: length from starting index |
| 3 | **String(Char, Int32)**<br>Arg 1: char, Arg 2: number of time occurred |
| 4 | **String(Char[])**<br>Arg 1: array of char |

| | |
|---|---|
| 5 | **String(Char[], Int32, Int32)**<br>Arg 1: array of char, Arg 2: starting index, Arg 2: length from starting index |
| 6 | **String(SByte*)**<br>Arg 1: pointer to an array of 8-bit signed integers |
| 7 | **String(SByte*, Int32, Int32)**<br>Arg 1: pointer to an array of 8-bit signed integers, Arg 2: starting index, Arg 3: length from starting index |
| 8 | **String(SByte*, Int32, Int32, Encoding)**<br>Arg 1: pointer to an array of 8-bit signed integers, Arg 2: starting index, Arg 3: length from starting index, Arg 4: Encoding object |

**Properties:**

| Sr.No. | Constructor & Description |
|---|---|
| 1 | **Chars[Int32]**<br>Arg 1: Index of character<br>Gets the Char object at a specified position in the current String object. |
| 2 | **Length**<br>Gets the number of characters in the current String object. |

## 12.3 StringBuilder

- StringBuilder is a mutable class, So it adopt change in it self.
- It's namespace is **"System.Text"**.

**Constructor:**

| Sr.No. | Constructor & Description |
|---|---|

| | |
|---|---|
| 1 | **StringBuilder()** |
| 2 | **StringBuilder(String)**<br>Arg 1: String |
| 3 | **StringBuilder(Int32)**<br>Arg 1: Current Capacity of String(Number of Characters exist in string) |
| 4 | **StringBuilder(String, Int32)**<br>Arg 1: String, Arg 2: Current Capacity of String |
| 5 | **StringBuilder(Int32, Int32)**<br>Arg 1: Current Capacity of String, Arg 2: Set Mex Capacity of String, if we put string which length is grater then max capacity then it throws exception. |
| 6 | **StringBuilder(String, Int32, Int32, Int32**<br>Arg 1: String, Arg 2: Starting Index of string, Arg 3: Length of string from starting index, Arg 4: Capacity of String |

**Properties:**

| Sr.No. | Constructor & Description |
|---|---|
| 1 | **Capacity**<br>The maximum number of characters that can be contained in the memory allocated by the current instance. Its value can range from Length to MaxCapacity. |
| 2 | **MaxCapacity**<br>The maximum number of characters this instance can hold. |
| 3 | **Chars[]**<br>The position of the character. |

| | |
|---|---|
| 4 | **Length**<br>The length of this instance. |

**Methods:**

| Sr.No. | Constructor & Description |
|---|---|
| 1 | **Append(String)**<br>Appends the string representation of a specified object to this instance. |
| 2 | **Insert(Int32, String)**<br>Inserts the string representation of a specified object into this instance at a specified character position. |
| 3 | **Remove(Int32, Int32)**<br>Removes the specified range of characters from this instance. |
| 4 | **Replace(String, String)**<br>Replaces all occurrences of a specified character or string in this instance with another specified character or string. |
| 5 | **CopyTo(Int32, Char[], Int32, Int32)**<br>Copies the characters from a specified segment of this instance to a specified segment of a destination Char array. |
| 6 | **Equals(String)**<br>Return: is string is match from another string then it will return true otherwise it will return false. |
| 7 | **Clear()**<br>Removes all characters from the current StringBuilder instance. |

# WORKING WITH DATETIMES

## 13.1 INTRODUCTION

- We used the DateTime when there is a need to work with the dates and times in C#.
- It is a structure not a class.
- Namespace is : **"System"**.
- Represents an instant in time, typically expressed as a date and time of day.

```
public readonly struct DateTime : IComparable, IComparable<DateTime>,
IConvertible, IEquatable<DateTime>, IParsable<DateTime>, ISpanFormattable,
ISpanParsable<DateTime>, System.Runtime.Serialization.ISerializable
```

## 13.2 DATETIME

**Constructor:**

| Sr.No. | Constructor & Description |
|---|---|
| 1 | **DateTime(Int64)**<br>Initializes a new instance of the DateTime structure to a specified number of ticks. |
| 2 | **DateTime(Int64, DateTimeKind)**<br>Initializes a new instance of the DateTime structure to a specified number of ticks and to Coordinated Universal Time (UTC) or local time. |
| 3 | **DateTime(Int32, Int32, Int32)**<br>Initializes a new instance of the DateTime structure to the specified year, month, and day. |
| 4 | **DateTime(Int32, Int32, Int32, Calendar)**<br>Initializes a new instance of the DateTime structure to the specified year, month, and day for the specified calendar. |
| 5 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32)** |

| | |
|---|---|
| | Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second. |
| 6 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and Coordinated Universal Time (UTC) or local time. |
| 7 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Calendar)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, and second for the specified calendar. |
| 8 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond. |
| 9 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time. |
| 10 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, and millisecond for the specified calendar. |
| 11 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Int32)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar. |
| 12 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)** |

| | |
|---|---|
| | Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar. |
| 13 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar. |
| 14 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar. |
| 15 | **DateTime(Int32, Int32, Int32, Int32, Int32, Int32, Int32, Calendar, DateTimeKind)**<br>Initializes a new instance of the DateTime structure to the specified year, month, day, hour, minute, second, millisecond, and Coordinated Universal Time (UTC) or local time for the specified calendar. |

**Properties:**

| Sr.No. | Constructor & Description |
|---|---|
| 1 | **Date**<br>Gets the date component of this instance. |
| 2 | **Day**<br>Gets the day of the month represented by this instance. |
| 3 | **DayOfWeek**<br>Gets the day of the week represented by this instance. |

| | |
|---|---|
| 4 | **DayOfYear**<br>Gets the day of the year represented by this instance. |
| 5 | **Hour**<br>Gets the hour component of the date represented by this instance. |
| 6 | **Kind**<br>Gets a value that indicates whether the time represented by this instance is based on local time, Coordinated Universal Time (UTC), or neither. |
| 7 | **Microsecond**<br>The microseconds component, expressed as a value between 0 and 999. |
| 8 | **Millisecond**<br>Gets the milliseconds component of the date represented by this instance. |
| 9 | **Minute**<br>Gets the minute component of the date represented by this instance. |
| 10 | **Month**<br>Gets the month component of the date represented by this instance. |
| 11 | **Nanosecond**<br>The nanoseconds component, expressed as a value between 0 and 900 (in increments of 100 nanoseconds). |
| 12 | **Now**<br>Gets a DateTime object that is set to the current date and time on this computer, expressed as the local time. |

| | |
|---|---|
| 13 | **Second**<br>Gets the seconds component of the date represented by this instance. |
| 14 | **Ticks**<br>Gets the number of ticks that represent the date and time of this instance. |
| 15 | **DayOfTime**<br>Gets the time of day for this instance. |
| 16 | **Today**<br>Gets the current date. |
| 17 | **UTCNow**<br>Gets a DateTime object that is set to the current date and time on this computer, expressed as the Coordinated Universal Time (UTC). |
| 18 | **Year**<br>Gets the year component of the date represented by this instance. |

**Methods:**

| Sr.No. | Constructor & Description |
|---|---|
| 1 | **Add(TimeSpan)**<br>Returns a new DateTime that adds the value of the specified TimeSpan to the value of this instance. |
| 2 | **AddDays(Double)**<br>Returns a new DateTime that adds the specified number of days to the value of this instance. |
| 3 | **AddHours(Double)** |

| | |
|---|---|
| | Returns a new DateTime that adds the specified number of hours to the value of this instance. |
| 4 | **AddMicroseconds(Double)**<br>Returns a new DateTime that adds the specified number of microseconds to the value of this instance. |
| 5 | **AddSeconds(Double)**<br>Returns a new DateTime that adds the specified number of seconds to the value of this instance. |
| 6 | **AddMilliseconds(Double)**<br>Returns a new DateTime that adds the specified number of milliseconds to the value of this instance. |
| 7 | **AddMinutes(Double)**<br>Returns a new DateTime that adds the specified number of minutes to the value of this instance. |
| 8 | **AddMonths(Int32)**<br>Returns a new DateTime that adds the specified number of months to the value of this instance. |
| 9 | **AddTicks(Int64)**<br>Returns a new DateTime that adds the specified number of ticks to the value of this instance. |
| 10 | **AddYears(Int32)**<br>Returns a new DateTime that adds the specified number of years to the value of this instance. |
| 11 | **Compare(DateTime, DateTime)**<br>Compares two instances of DateTime and returns an integer that indicates whether the first instance is earlier than, the same as, or later than the second instance. |

| | |
|---|---|
| 12 | **CompareTo**<br>Compares the value of this instance to a specified DateTime value and indicates whether this instance is earlier than, the same as, or later than the specified DateTime value. |
| 13 | **DaysInMonth(Int32, Int32)**<br>Returns the number of days in the specified month and year. |
| 14 | **Equals**<br>Returns a value indicating whether two DateTime objects, or a DateTime instance and another object or DateTime, have the same value. |
| 15 | **FromBinary(Int64)**<br>Deserializes a 64-bit binary value and recreates an original serialized DateTime object. |
| | **FromFileTime(Int64)**<br>Converts the specified Windows file time to an equivalent local time. |
| | **FromFileTimeUtc(Int64)**<br>Converts the specified Windows file time to an equivalent UTC time. |
| | **FromOADate(Double)**<br>Returns a DateTime equivalent to the specified OLE Automation Date. |
| | **GetDateTimeFormats**<br>Converts the value of this instance to all the string representations supported by the standard date and time format specifiers. |
| | **GetHashCode**<br>Returns the hash code for this instance. |

| | |
|---|---|
| | **GetTypeCode**<br>Returns the TypeCode for value type DateTime. |
| | **IsDaylightSaving**<br>Indicates whether this instance of DateTime is within the daylight saving time range for the current time zone. |
| | **IsLeapYear(Int32)**<br>Returns an indication whether the specified year is a leap year. |
| | **Parse**<br>Converts the string representation of a date and time to its DateTime equivalent. |
| | **ParseExact**<br>Converts the specified string representation of a date and time to its DateTime equivalent. The format of the string representation must match a specified format exactly or an exception is thrown. |
| | **SpecifyKind(DateTime, DateTimeKind)**<br>Creates a new DateTime object that has the same number of ticks as the specified DateTime, but is designated as either local time, Coordinated Universal Time (UTC), or neither, as indicated by the specified DateTimeKind value. |
| | **Subtract**<br>Returns the value that results from subtracting the specified time or duration from the value of this instance. |
| | **ToBinary**<br>Serializes the current DateTime object to a 64-bit binary value that subsequently can be used to recreate the DateTime object. |

| | |
|---|---|
| | **ToFileTime**<br>Converts the value of the current DateTime object to a Windows file time. |
| | **ToFileTimeUtc**<br>Converts the value of the current DateTime object to a Windows file time. |
| | **ToLocalTime**<br>Converts the value of the current DateTime object to local time. |
| | **ToLongDateString**<br>Converts the value of the current DateTime object to its equivalent long date string representation. |
| | **ToLongTimeString**<br>Converts the value of the current DateTime object to its equivalent long time string representation. |
| | **ToOADate**<br>Converts the value of this instance to the equivalent OLE Automation date. |
| | **ToShortDateString**<br>Converts the value of the current DateTime object to its equivalent short date string representation. |
| | **ToShortTimeString**<br>Converts the value of the current DateTime object to its equivalent short time string representation. |
| | **ToString**<br>Converts the value of the current DateTime object to its equivalent string representation. |

| | |
|---|---|
| | **ToUniversalTime**<br>Converts the value of the current DateTime object to Coordinated Universal Time (UTC). |
| | **TryFormat**<br>Tries to format the value of the current datetime instance into the provided span of characters. |
| | **TryParse**<br>Converts the specified string representation of a date and time to its DateTime equivalent and returns a value that indicates whether the conversion succeeded. |
| | **TryParseExact**<br>Converts the specified string representation of a date and time to its DateTime equivalent. The format of the string representation must match a specified format exactly. The method returns a value that indicates whether the conversion succeeded. |