

Phase-1

Name : Kotadiya Amisha K.

College : Government Engineering College, Rajkot

Branch : Computer Engineering

Understanding

Module-1

➤ Visual Studio 2019 IDE Overview

- Different types of windows (solution exp.,properties etc.)
- Solution,Project
- Code editor features
- Shortcuts

1) Different types of windows (solution exp.,properties etc.)

The IDE has two basic window types, *tool windows* and *document windows*.

Tool windows

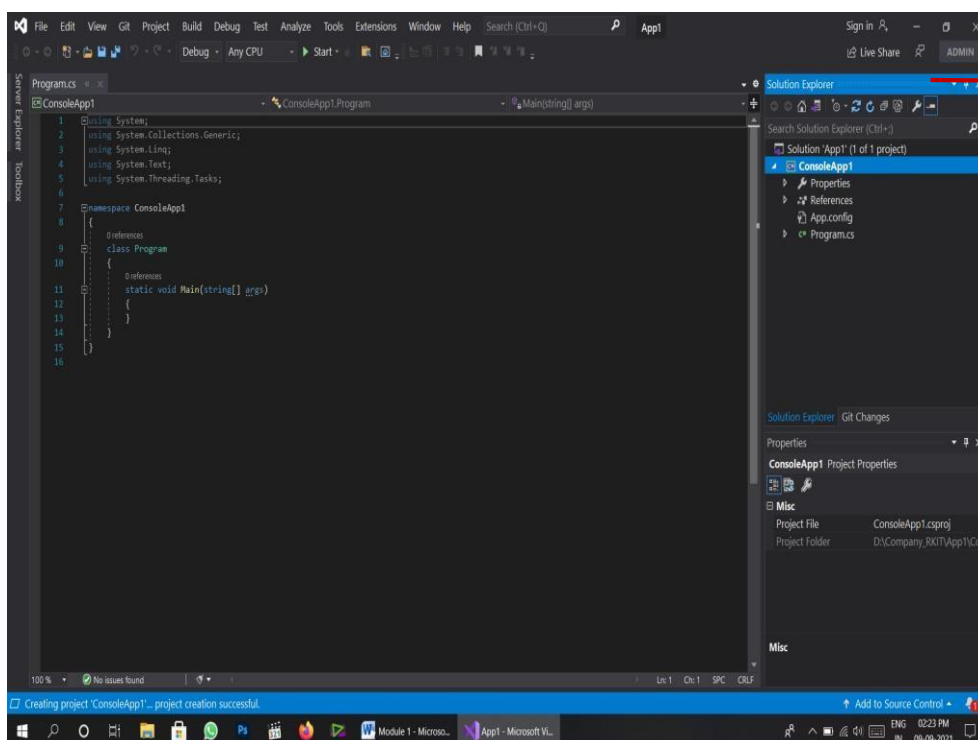
Tool windows include Solution Explorer, Server Explorer, Output Window, Error List, the designers, the debugger windows, and so on.

Tool windows can be resized and dragged by their title bar.

Document windows

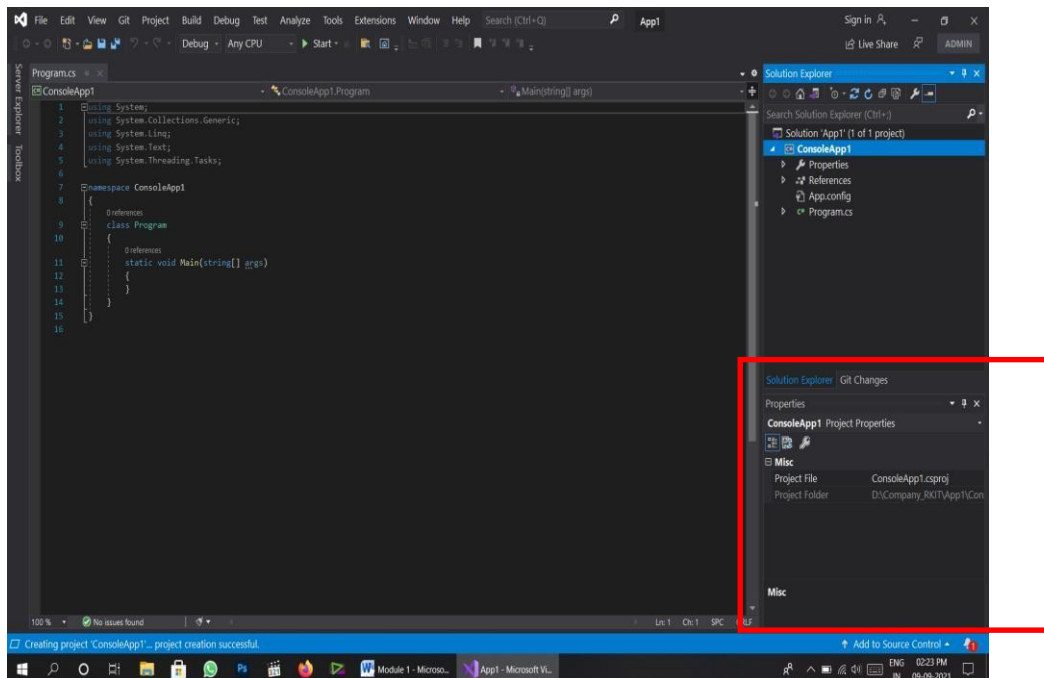
Document windows contain source code files, arbitrary text files, config files, and so on.

Solution Explorer:



Solution Explorer is a window in IDE as shown in the figure, that enables you to manage solutions, projects, and files. It provides a complete view of the files in a project, and it enables you to add or remove files and to organize files into subfolders. It helps in manage your code files and also provides navigation. It groups similar projects or project files under single solution.

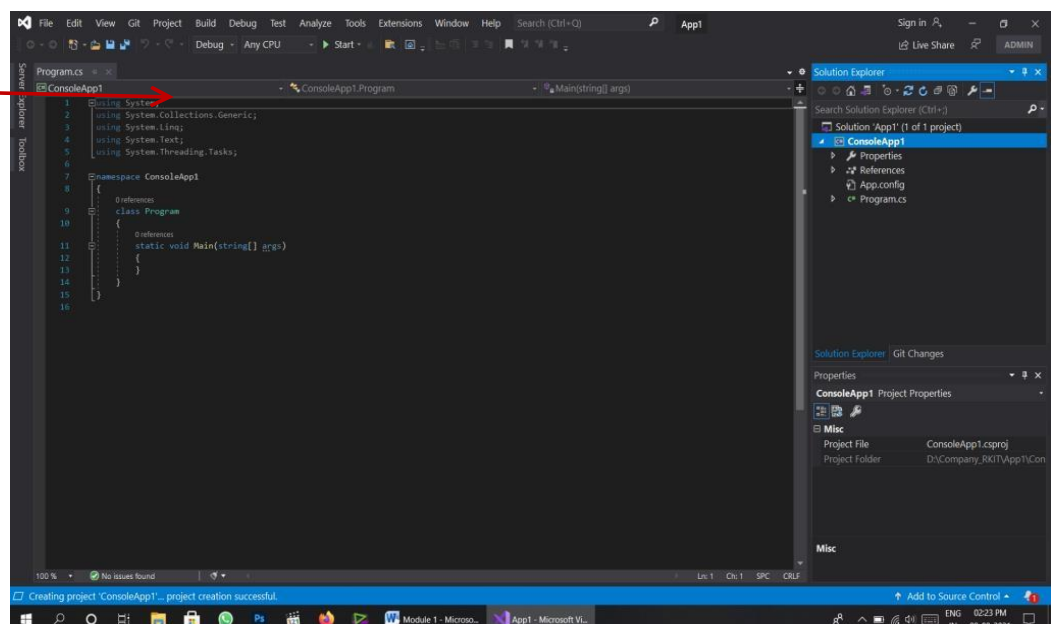
Properties Window:



The Properties window displays different types of editing fields, depending on the needs of a particular property. These edit fields include edit boxes, drop-down lists, and links to custom editor dialog boxes. Properties shown in gray are read-only.

Editor Window:

Editor Window



This is where we write our code. It displays the file contents. We can edit the code or design a user interface like buttons or text boxes. It provides varieties of features which make it easier to write and manage the code. These features include syntax colouring, errors and warning marks, Bracematching, structure visualizer, line numbers, etc...

2) Solution,Project

Solution:

- A Solution contains group of similar projects. 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. A solution is described by a text file (extension .sln) with its own unique format; it's not intended to be edited by hand.
- Visual Studio uses two file types (.sln and .suo) to store settings for solutions:
- .sln - Visual Studio Solution-Organizes projects, project items, and solution items in the solution.
- .suo- Solution User Options- Stores user-level settings and customizations, such as breakpoints.

Project:

When you create an app or website in Visual Studio, you start with a *project*. In a logical sense, a project contains all files that are compiled into an executable, library, or website. Those files can include source code, icons, images, data files, and so on. A project also contains compiler settings and other configuration files that might be needed by various services or components that your program communicates with.

For customizing our projects in a certain way, we can create a custom project template that can be used to create new projects from. For more information, see Create project and item templates.

3) Code editor features

- **Syntax Colouring:**

Some syntax elements of code and mark-up files are coloured differently to distinguish them. For example, keywords are one colour, but types with another colour.

- **Error and warning mark:**

As we add code and build your solution, we may see different-coloured wavy underlines or light bulbs appearing in the code. Red wavy underlines denote syntax errors, blue denotes compiler errors, green denotes warnings, and purple denotes other types of errors. Quick Actions suggest fixes for problems and make it easy to apply the fix.

- **Brace Matching:**

When the insertion point is placed on an open brace in a codefile, both it and the closing brace are highlighted. This feature gives immediate feedback on misplaced or missing braces.

- **Line Numbers:**

Line numbers can be displayed in the left margin of the code window. It can be useful in case of debugging code editing task becomes easier with line numbers.

- **Change Tracking:**

The colour of the left margin allows you to keep track of the changes you have made in a file. Changes you have made since the file was opened but not saved are denoted by a yellow bar on the left margin (known as the selection margin). After you have saved the changes (but before closing the file), the bar turns green. If you undo a change after you have saved the file, the bar turns orange.

- **Structure Visualizer:**

Dotted lines connect matching braces in code files, making it easier to see opening and closing brace pairs.

- **Format Document:**

Sets the proper indentation of lines of code and moves curly braces to separate lines in the document.

- **Format Selection:**

Sets the proper indentation of lines of code and moves curly braces to separate lines in the selection.

- **Make Lowercase:**

Changes all characters in the selection to lowercase, or if there is no selection, changes the character at the insertion point to lowercase.

- **Word Wrap:**

Causes all the lines in a document to be visible in the codewindow.

- **Move selected Lines Up:**

Moves the selected line up one line. Shortcut: **Alt+Up Arrow**.

- **Comment Selection:**

Adds comment characters to the selection or the current line.

- **Uncomment Selection:**

Removes comment characters from the selection or the current line.

4) Shortcuts

We can navigate in Visual Studio more easily by using the shortcuts. These shortcuts include keyboard and mouse shortcuts as well as text you can enter to help accomplish a task more easily.

Shortcuts for some basic functioning are listed in table given below:

Commands	Shortcut Keys
Maximize floating window	Double-Click on title bar
Maximize/minimize windows	Win+Up arrow/Win+Down arrow
Close active document	Ctrl+F4
Show open file list	Ctrl+Alt+Down arrow
Show all floating windows	Ctrl+Shift+M
Switch between windows	Win+N
Solution Explorer search	Ctrl+;
Quick Find	Ctrl+F
Dismiss Find	Esc
Quick Replace	Ctrl+H
Go To All	Ctrl+T
Start debugging	F5
Stop debugging	Shift+F5
Restart debugging	Ctrl+Shift+F5
Toggle full screen	F11
New File	Ctrl+N
Open File	Ctrl+O
New window/instance	Ctrl+Shift+N
Close window/instance	Ctrl+Shift+W

Module-2

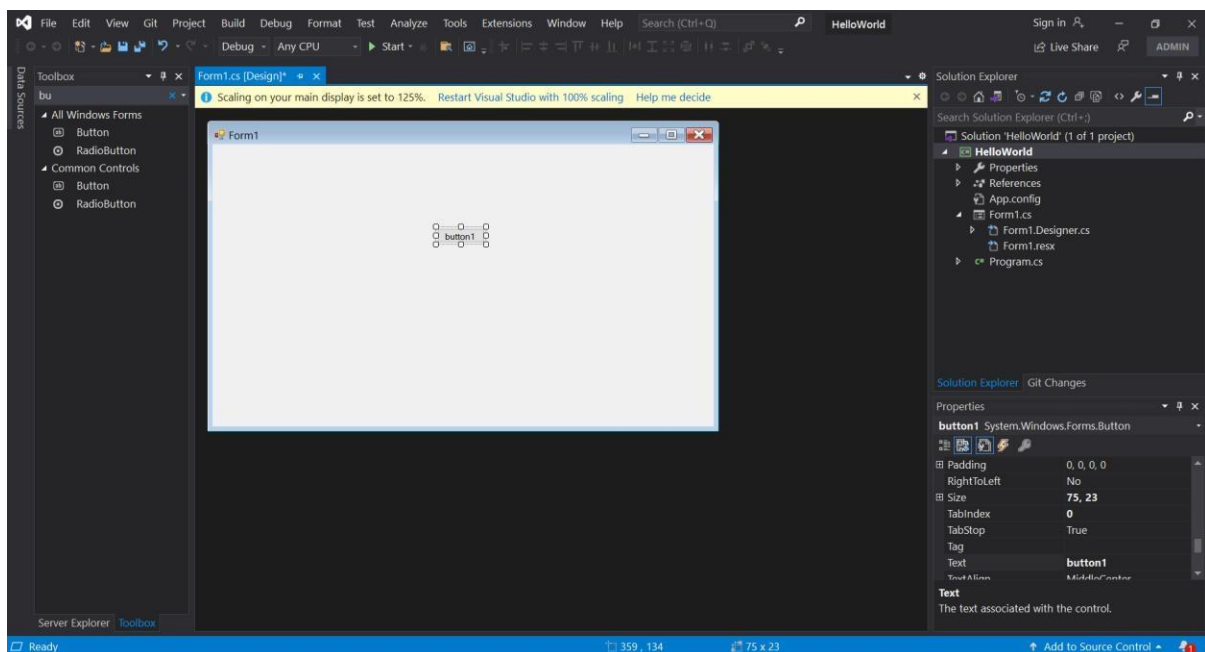
➤ Project Types

- Windows App, Class Library
- Web Application

1) Windows App, Class Library

Windows App:

- A program that is written to run under the Microsoft Windows operating system is called a "Windows app." Visual Studio 2019 IDE allows us to create windows app.
- First, you'll create a Visual Basic application project. The project type comes with all the template files you'll need, before you've even added anything.
- After you select your Visual Basic project template and name your file, Visual Studio opens a form for you. A form is a Windows user interface.
- Click Toolbox to open the Toolbox fly-out window. Then Click the Pin icon to dock the Toolbox window. Click the Button control and then drag it onto the form.



- In the Appearance section (or the Fonts section) of the Properties window, type Click This Button, and then press Enter. In the Design section of the Properties window, change the name from Button1 to btnHello, and then press Enter.
- Select the Label control from the Toolbox window, and then drag it onto the form and drop it beneath the Click This Button button. Then change the name of Label1 to lblHello, and then press Enter

Class Library:

The Class Library contains program code, data, and resources that can be used by other programs and are easily implemented into other Visual Studio projects. In visual studio we can implement code in project type as class library and then it can be referred by other programs or even in same program.

Simple program that demonstrates use of class library is shown below. I have prepared one class library project and another console application which uses method specified in the class library.

Class library code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ClassLibrary
{
    public sealed class Class1
    {
        public int add(int a, int b)
        {
            return a + b;
        }
        public int sub(int a, int b)
        {
            return a - b;
        }
    }
}
```

Console application which uses the class library method:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using ClassLibrary;

namespace use_library
```

```
{
class Program
{
static void Main(string[] args)
{
Class1 c = new Class1();
Console.WriteLine(c.add(15,75));
Console.WriteLine(c.sub(95, 75));
}
}
}
```

2) Web Application

A web application is a computer program that utilizes web browsers and web technology to perform tasks over the Internet. Microsoft Visual Studio is an effective development environment for those who work with Microsoft development services. It's the best solution to create web apps with .NET or ASP.NET. If you would like to create a small website or build a basic web app, then you can choose Visual Studio Code.

Here is the code of small web app prepared by me. I have taken project type as ASP.Net core web app. Target framework is ASP.Net Core 3.1.

ASP .NET is one of the popular web application frameworks developed by Microsoft that allows web developers to build the best dynamic websites. Well, you can use two programming languages such as VB or C# as per your interest.

The web applications are cloud-based, modern, and internet-based & are developed using various services and tools

Now let's check the tools:

Visual Studio - Visual Studio has always been a part of the web application development process. This tool is used by many developers to build effective web applications.

Below are some of the extensions of Visual Studio:

1. NuGet Package Manager
2. Resharper

3. Web Essentials

4. Version Control System - This tool is a management system that provides the facility to track the changes that were made.

Module-3

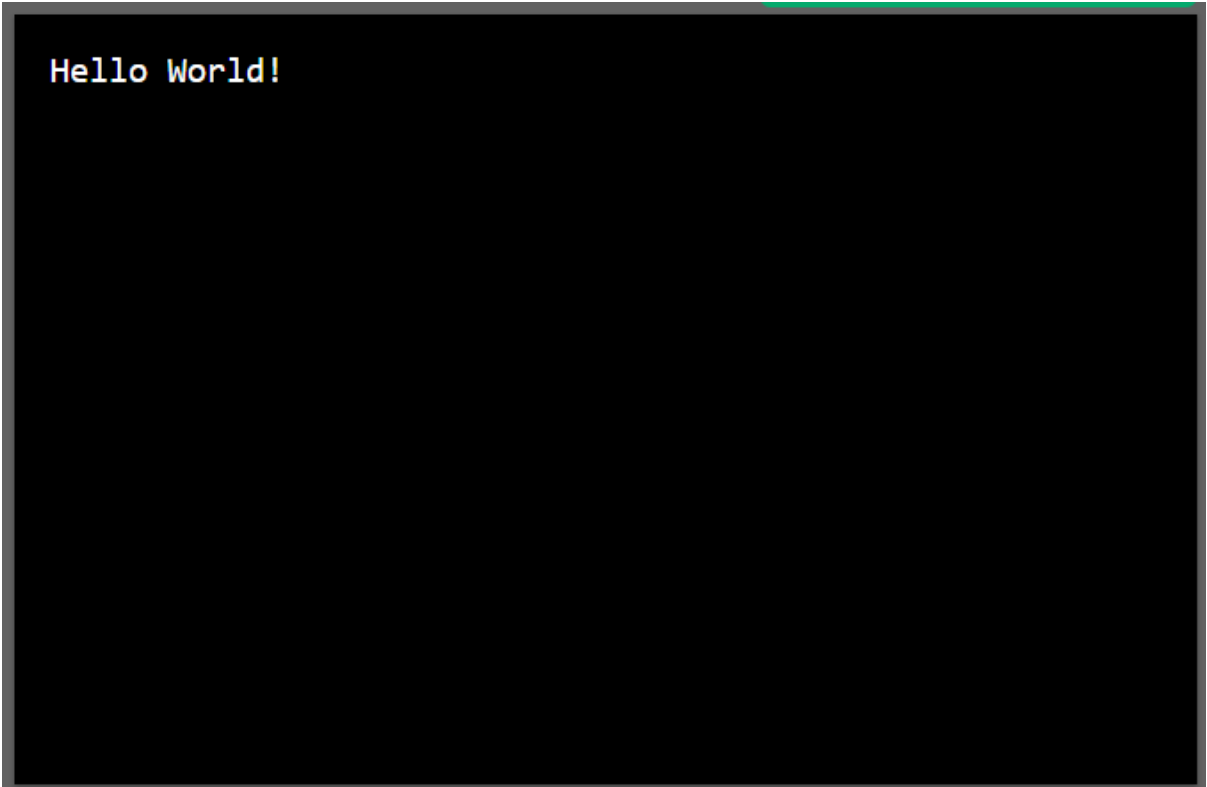
➤ Create First C# Program “Hello World”

- What is namespace?
- What is class?
- Variable & Method Declaration

A simple Console application for Hello World program in C# is as shown below:

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

Output:-

A screenshot of a console application window with a black background and a gray border. The text "Hello World!" is displayed in a monospaced font at the top left of the window.

Hello World!

1) What is namespace?

- Namespaces are used to organize the classes.
- It helps to control the scope of methods and classes in larger .Net programming projects.
- It provides a way to keep one set of names (like class names) different from other sets of names.
- The biggest advantage of using namespace is that the class names which are declared in one namespace will not clash with the same class names declared in another namespace.
- It is also referred as named group of classes having common features.
- The members of a namespace can be namespaces, interfaces, structures, and delegates.
- To define a namespace in C#, we will use the namespace keyword followed by the name of the namespace and curly braces containing the body of the namespace as follows:

```
namespace namespace_name {  
    // code declarations  
}
```

- The members of a namespace are accessed by using dot(.) operator. A class in C# is fully known by its respective namespace.
- Syntax:
[namespace_name].[member_name]
- C# provides a keyword “using” which help the user to avoid writing fully qualified names again and again.
- The user just has to mention the namespace name at the starting of the program and then he can easily avoid the use of fully qualified names.
- Syntax:
using [namespace_name][.][sub-namespace_name]

- You can also define a namespace into another namespace which is termed as the nested namespace. To access the members of nested namespace user has to use the dot(.) operator.
- For example, Generic is the nested namespace in the collections namespace as System.Collections.Generic.

Example:-

```
using System;
namespace First
{
    public class Hello
    {
        public void sayHello()
        {
            Console.WriteLine("Hello First Namespace");
        }
    }
}
namespace Second
{
    public class Hello
    {
        public void sayHello()
        {
            Console.WriteLine("Hello Second Namespace");
        }
    }
}
public class NamespaceEx
{
    public static void Main()
    {
        First.Hello h1 = new First.Hello();
        Second.Hello h2 = new Second.Hello();
        h1.sayHello();
        h2.sayHello();
    }
}
```

Output:-

```
Hello First Namespace  
Hello Second Namespace
```

2) What is class?

- Class is basically a blueprint for object. Objects are instances of a class. The methods and variables that constitute a class are called members of the class.
- Defining Class in C#: A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces.
- For eg:
class class_name{
//members of class
}
- For accessing members of class dot(.) operator is used.
- Default access modifier for class is internal. However, different access modifiers can be public, private, protected, protected internal & private protected.

Example:-

```
using System;
namespace ClassEx
{
    class Program
    {
        static void Main(string[] args)
        {
            Square s = new Square();
            s.length = 20;
            int area = s.length * s.length;
            Console.WriteLine("Area of square is : {0}", area);
        }
    }
    class Square
    {
        public int length;
    }
}
```

Output:-

```
Area of square is : 400
```

3) Variable & Method Declaration

Variable in C#

- A variable is a name given to a storage area that our programs can manipulate.
- Each variable in C# has a specific type, which determines the size and layout of the variable's memory the range of values that can be stored within that memory and the set of operations that can be applied to the variable.
- Basic value types provided in C# are:
 - 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
- C# also allows defining other value types of variable such as enum and reference types of variables such as class.
- How to define variable in C#?
Syntax:
<data_type><variable_list>;
- How to initialize variables in C#?
Syntax:
variable_name = value;
- However, Variables can be initialized in their declaration too. The initializer consists of an equal sign followed by a constant expression as –
<data_type><variable_name>= value;

Example:-

```
using System;
namespace VariableEx
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 17;
            float b = 09.15F;
        }
    }
}
```

```

double sum;
sum = a + b;
Console.WriteLine("sum = {0} + {1}\n = {2}", a,b,sum);
    }
}
}

```

Output:-

```

sum = 17 + 9.15
    = 26.1499996185303

```

Method Declaration in C#

- 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.

- How to define method in C#?

Syntax:

```

<Access Specifier><Return Type><Method Name>(Parameter List) {
Method Body
}

```

- 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; that is, a method may contain no parameters.
- Method body – This contains the set of instructions needed to complete the required activity.
- How to Call a Method in C#?
You can call a method using the name of the method.

Example:-

```
using System;

namespace MethodDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            int l = 4;
            int b = 7;
            int area = r.Area(l, b);

            Console.WriteLine("Area of rectangle = length {0} and breadth {1}\n\t\t = {2}", l,b,area);
            Console.Read();
        }
    }
    class Rectangle
    {
        public int Area (int length, int breadth)
        {
            int ans = length * breadth;
            return ans;
        }
    }
}
```

Output:-

```
Area of rectangle = length 4 and breadth 7  
                  = 28
```

Module-4

➤ Understanding C# Program

- Program Flow
- Understanding Syntax

1) Program Flow

To understand the flow of a C# program, let us begin by analyzing a very simple program in C#. Example shows a program written in C#, which simply displays a message on the user's screen.

Example :

```
001:  /* This is a simple program in C# */  
002:  
003:  using System;  
004:  class ProgramFlow  
005:  {  
006:      public static void Main()  
007:      {  
008:          Console.WriteLine("My simple program in C#");  
009:      }  
010; }
```

Line 001: This is a comment. Comments can be included in any part of a C# program. Here we have used a standard C/C++ style comment. The comment begins with “/*” and ends with “*/”. The comment can span multiple lines.

Line 003: This line, **using System**, is quite similar to the **#include** statement used in C/C++. The **#include** statement was used to include another header (source) file, so as to make the functions, present within that header file, a part of the current program. Similarly the keyword **using** imports the **System** class file and makes the methods present within it as a part of the program. But **System** here is known as a namespace and not as a header file. For now, just think of namespaces as a collection of classes. The **System** namespace contains the classes that most applications use for interacting with the operating system. The classes that are used more often are the ones required for basic Input/Output. Note that there is a semicolon at the end of this line. All lines of code in C# must end with a semicolon, just like in C/C++. As you can see for yourself, C# has strong roots in C and C++.

Line 004: This line defines a class.

Line 006: Each class has one static void **Main()** function. This function is the entry point of a C# program. This means that the **Main()** function is the first function that is called when program execution begins. It is declared as **public** to make it accessible from just about anywhere in the program. The keyword **public** can be ignored, as by default, just as in C++, in C# also the members of a class are public. The **Main()** function is declared as a **static** member (Static methods are the methods that can be called without creating an object of class.). Since in our program the **Main()** does not return any value its return type is declared as **void**. Do keep in mind the case of the keywords, all keywords on this line of code are in lower case except in case of the **Main()** function **M** is in upper case.

Line 007: Next we open the scope of the method (or function) using curly braces.

Line 008: Within the **Main()** function we call the **WriteLine** method of the **Console** class and pass the text “My simple program in C#” as its parameter. The **WriteLine** function displays text on the console or the DOS window. Note that the **WriteLine** method is a part of the **Console** class, which in turn is a part of the **System** namespace. If we had not specified the **using System** clause in **Line 003**, we would have had to write this line (Line 008) as:

```
System.Console.WriteLine("My simple program in C#");
```

Using a fully qualified name to refer objects can be error prone. To ease this burden, C# provides us with the **using** directive, which we specified at Line 003. Also remember that you can put more than one **using** directive, but they all must be specified at the beginning of the program.

Line 009: The **Main()** function is terminated using the closing braces.

When the program is executed it will just display the message “My simple program in C#”. Though it may seem like it has not accomplished much, do not think likewise. You have just understood the basic flow of program execution in C#!

A point to note is that as mentioned earlier, **C# is case sensitive!** For example, the “using” is not the same as “Using”!

2) Understanding Syntax

Comments in C#

Comments are ignored by the C# compiler. Therefore, comments are used for the explanation of the code. Comments helps in debugging the program.

Comments helps in reminding the code later. In C#, comments are of two types:

- single-line comments
- multi-line comments

Single-line Comments in C#

A single-line comments starts with `//`. That is, all the words in the same line after the `//` are referred as single-line comments. Here is an example of single-line comments in C# programming:

```
// this is single line comments
```

Multi-line Comments in C#

Multi-line comments starts with `/*` and ends with `*/`. No matter, how many lines you are using. Here is an example showing the multi-line comments in C# programming:

```
/* hello,  
 * i am multi-line  
 * comment */
```

Identifiers in C#

An identifier is a name simply used to identify a variable, class, function, or any other user-defined items. Here are the general and basic rules for naming an identifier:

- An identifier name must begin with a letter, followed by sequence of letters, digits (0 to 9) or underscore.
- An identifier name must not be a C# keyword

Keywords in C#

Keywords in C#, are the reserved words, having special meaning to the C# compiler.

Here, the following table lists, the reserved keywords in C# programming:

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

Module-5

➤ Working with code files, projects & solutions

- Understanding structure of solution
- Understanding structure of project(Win app, web app, web api, class library)
- Familiar with different type of file extensions

1) Understanding structure of solution

A solution contains a collection of projects, along with information on dependencies between those projects. The projects themselves contain files. This structure is illustrated in [Figure 1-1](#). You can have as many projects as you like in a solution, but there can be only one solution open at a time in a particular instance of VS.NET. (You can, of course, open multiple solutions by running multiple instances of VS.NET.)

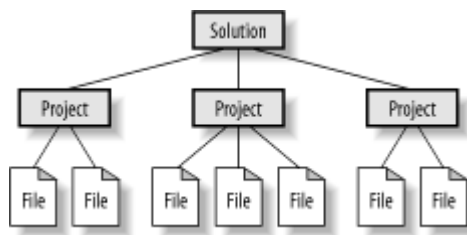


Figure 1-1. A solution, its projects, and their files

Solutions contain only projects—you cannot nest one solution inside another. However, projects can belong to multiple solutions, as [Figure 1-2](#) shows, which gives you great flexibility for organizing your builds, particularly in large applications.

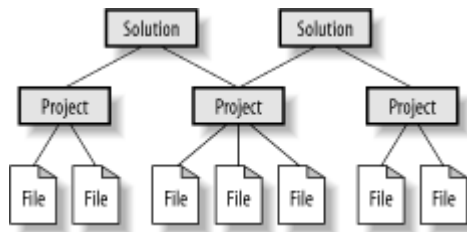


Figure 1-2. Projects that belong to multiple solutions

With Microsoft's previous generation of development tools, each language had its own integrated development environment. Now there is just one unified environment. In addition, there are no restrictions on the range of different project types any single solution can contain, so you can work on, say, an unmanaged C++ DLL in the same solution as a VB.NET Window Forms application, which can greatly simplify development, debugging, and deployment.

Figure 1-5 illustrates how the physical directory structure can reflect the logical structure of a project. Figure 1-6 shows how Visual Studio .NET will organize the directory structure if left to its own devices—the physical structure is less closely related to the logical structure. The solution file is located in an arbitrary project directory. (Specifically, it is in the first project that was created in the solution.) The project directories themselves may well be in the same directory as other, unrelated directories or files. So, to avoid the mess shown in Figure 1-6, be sure to check the Create directory for solution checkbox.

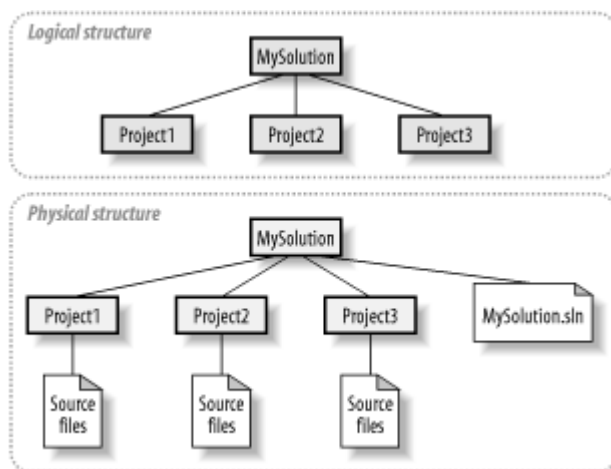


Figure 1-5. Solution structure and directory structure in harmony

Visual Studio .NET classifies projects by implementation language and then by project type in its New Project dialog box. However, many of the project types have a great deal in common despite using different languages, so although VS.NET 2003 Enterprise Edition lists more than 90 distinct types, most fall into one of six groups: managed local projects, managed web projects, Smart Device projects, unmanaged local projects, unmanaged web projects, and setup projects. **web api, class library**)

A project has two main jobs: to act as a container for our source files and to compile those files into some kind of component, typically either a Dynamic Link Library (DLL) or Windows Executable (EXE). We shall now run through the main types of projects supported by VS.NET.

A *managed local project* will create a .NET assembly. *Managed web projects* do the same, but the project output is intended to be accessed by a client over a network connection, typically using either a browser or a web

service proxy. Web projects are therefore always associated with a web application on a web server. And although managed web projects produce a .NET assembly just like a managed local project, with a web project, Visual Studio .NET will place the assembly on the web server as part of the build process.

Managed local

A managed local application could be written in C#, J#, VB.NET, or Managed C++ (MC++). VB.NET, C#, and J# all support the same local application types, which are shown in Table

Project template	Project output	Type of file built
Windows Application	A Windows Forms application	Managed EXE
Class Library	An assembly to be used by other .NET assemblies	Managed DLL
Windows Control Library	An assembly containing at least one class derived from System.Windows.Forms.Control	Managed DLL
Web Control Library	An assembly containing at least one class derived from System.Web.UI.Control	Managed DLL
Console Application	A command-line application	Managed EXE
Windows Service	A Windows Service	Managed EXE
Empty Project	Any kind of .NET assembly	Managed EXE or DLL

3) Familier with different type of file extentions

Types of file extensions :-

- Audio file formats by file extensions.
- Compressed file extensions.
- Disc and media file extensions.
- E-mail file extensions.
- Executable file extensions.
- Font file extensions.
- Image file formats by file extensions.
- Internet-related file extensions.
- Presentation file formats by file extensions.
- Programming files by file extensions.
- Spreadsheet file formats by file extensions.
- System related file formats and file extensions.
- Video file formats by file extensions.
- Word processor and text file formats by file extensions.

Data and database file extensions

A data file could be any file, but for this list, we've listed the most common data files that relate to data used for a database, errors, information, importing, and exporting.

- **.csv** - Comma separated value file
- **.dat** - Data file
- **.db** or **.dbf** - Database file
- **.log** - Log file
- **.mdb** - Microsoft Access database file
- **.sav** - Save file (e.g., game save file)
- **.sql** - SQL database file
- **.xml** - XML file

Executable file extensions

The most common executable file are files ending with the .exe file extension. However, other files can also be run by themselves or with the aid of an interpreter.

- **.apk** - Android package file
- **.bat** - Batch file
- **.bin** - Binary file
- **.cgi** or **.pl** - Perl script file
- **.com** - MS-DOS command file
- **.exe** - Executable file
- **.gadget** - Windows gadget
- **.jar** - Java Archive file
- **.msi** - Windows installer package
- **.py** - Python file
- **.wsf** - Windows Script File

Programming files by file extensions

Many file extensions are used for programs before they are compiled or used as scripts. Below is a list of the most common file extensions associated with programming.

- **.c** - C and C++ source code file
- **.cgi** and **.pl** - Perl script file.
- **.class** - Java class file
- **.cpp** - C++ source code file
- **.cs** - Visual C# source code file
- **.h** - C, C++, and Objective-C header file
- **.java** - Java Source code file
- **.php** - PHP script file.
- **.py** - Python script file.
- **.sh** - Bash shell script
- **.swift** - Swift source code file
- **.vb** - Visual Basic file

Spreadsheet file formats by file extension

Below are the most common file extensions used to save spreadsheet files to a computer.

- **.ods** - OpenOffice Calc spreadsheet file
- **.xls** - Microsoft Excel file
- **.xlsm** - Microsoft Excel file with macros
- **.xlsx** - Microsoft Excel Open XML spreadsheet file

System related file formats and file extensions

Like all other programs, your operating system uses files and has file extensions that are more common than others. Below is a list of the most common file extensions used on operating systems.

- **.bak** - Backup file
- **.cab** - Windows Cabinet file
- **.cfg** - Configuration file
- **.cpl** - Windows Control panel file
- **.cur** - Windows cursor file
- **.dll** - DLL file
- **.dmp** - Dump file
- **.drv** - Device driver file
- **.icns** - macOS X icon resource file
- **.ico** - Icon file
- **.ini** - Initialization file
- **.lnk** - Windows shortcut file
- **.msi** - Windows installer package
- **.sys** - Windows system file
- **.tmp** - Temporary file

Module-6

➤ Understanding datatype & variables with conversion

- Base datatype
- Datatype Conversion
- Boxing/Unboxing

1) Base datatype

Data types specify the type of data that a valid C# variable can hold. C# is a **strongly typed programming language** because in C#, each type of data (such as integer, character, float, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.

Data types in C# is mainly divided into three categories

- **Value Data Types**
- **Reference Data Types**
- **Pointer Data Type**

(1)Value Data Types : In C#, the Value Data Types will directly store the variable value in memory and it will also accept both signed and unsigned literals. The derived class for these data types are **System.ValueType**.

Following are **different Value Data Types** in C# programming language :

- **Signed & Unsigned Integral Types :** There are 8 integral types which provide support for 8-bit, 16-bit, 32-bit, and 64-bit values in signed or unsigned form.
- **Floating Point Types :** There are 2 floating point data types which contain the decimal point.
Float: It is **32-bit single-precision** floating point type. It has 7 digit Precision. To initialize a float variable, use the suffix f or F. Like, float x = 3.5F;. If the suffix F or f will not use then it is treated as double.
Double: It is **64-bit double-precision** floating point type. It has 14 – 15 digit Precision. To initialize a double variable, use the suffix d or D. But it is not mandatory to use suffix because by default floating data types are the double type.
- **Decimal Types :** The decimal type is a 128-bit data type suitable for financial and monetary calculations. It has 28-29 digit Precision. To initialize a decimal variable, use the suffix m or M. Like as, decimal x = 300.5m;. If the suffix m or M will not use then it is treated as double.
- **Character Types :** The character types represents a UTF-16 code unit or represents the 16-bit Unicode character.

- **Boolean Types :** It has to be assigned either true or false value. Values of type bool are not converted implicitly or explicitly (with casts) to any other type. But the programmer can easily write conversion code.

Example:-

```
using System;

namespace ValueDatatype
{
    class Program
    {
        static void Main(string[] args)
        {
            // declaring character
            char a = 'G';

            // Integer data type is generally used for numeric values
            int i = 89;
            short s = 56;

            // long uses Integer values which may signed or unsigned
            long l = 4564;

            // UInt data type is generally used for unsigned integer values
            uint ui = 95;
            ushort us = 76;

            // ulong data type is generally used for unsigned integer values
            ulong ul = 3624573;

            // by default fraction value is double in C#
            double d = 8.358674532;

            // for float use 'f' as suffix
            float f = 3.7330645f;

            // for decimal use 'm' as suffix
            decimal dec = 389.5m;

            Console.WriteLine("char: " + a);
            Console.WriteLine("integer: " + i);
```

```
Console.WriteLine("short: " + s);
Console.WriteLine("long: " + l);
Console.WriteLine("float: " + f);
Console.WriteLine("double: " + d);
Console.WriteLine("decimal: " + dec);
Console.WriteLine("Unsigned integer: " + ui);
Console.WriteLine("Unsigned short: " + us);
Console.WriteLine("Unsigned long: " + ul);
}
}
}
```

Output:-

```
char: G
integer: 89
short: 56
long: 4564
float: 3.733064
double: 8.358674532
decimal: 389.5
Unsigned integer: 95
Unsigned short: 76
Unsigned long: 3624573
```

(2)Reference Data Types : The Reference Data Types will contain a memory address of variable value because the reference types won't store the variable value directly in memory. The built-in reference types are **string, object**.

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

Example :

```
string s1 = "hello"; // creating through string keyword   String s2 =  
"welcome"; // 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. When a variable of a value type is converted to object, it's called **boxing**. When a variable of type object is converted to a value type, it's called **unboxing**. Its type name is **System.Object**.

Example:-

```
using System;  
  
namespace ReferenceDatatype  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
  
            // declaring string  
            string a = "Kotadiya ";  
  
            //append in a  
            a+="Amisha";  
            a = a+" Kishorbhai";  
            Console.WriteLine(a);  
  
            // declare object obj  
            object obj;  
            obj = 17;  
            Console.WriteLine(obj);  
  
            // to show type of object  
            // using GetType()  
            Console.WriteLine(obj.GetType());  
        }  
    }  
}
```


Output:-

```
Kotadiya Amisha Kishorbhai  
17  
System.Int32
```

(3)Pointer Data Type : The Pointer Data Types will contain a memory address of the variable value.

To get the pointer details we have a two symbols **ampersand (&)** and **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
```

Example:-

```
using System;

namespace PointerDatatype
{
    class Program
    {
        static void Main(string[] args)
        {
            unsafe
            {
                // declare variable
                int n = 17;

                // 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 :17
Address :1764744000
```

2) Datatype Conversion

Type casting is when you assign a value of one data type to another type.

In C#, there are two types of casting:

Implicit Casting (automatically)

converting a smaller type to a larger type size char -> int -> long -> float -> double

Implicit type conversion in C#, are performed in a type-safe manner. For instance, conversion from smaller to larger integral types.

```
using System;

namespace ImplicitConversion
{
    class Program
    {
        static void Main(string[] args)
        {

            char a = 'A';

            // automatic type conversion
            int i = a;

            // automatic type conversion
            long l = i;

            // automatic type conversion
            float f = l;

            // automatic type conversion
            double d = f;

            // Display Result
            Console.WriteLine("Char value " +a);
            Console.WriteLine("Int value " +i);
            Console.WriteLine("Long value " +l);
            Console.WriteLine("Float value " +f);
            Console.WriteLine("Double value " +d);
```

```
}  
}  
}
```

Output:-

```
Char value A  
Int value 65  
Long value 65  
Float value 65  
Double value 65
```

Explicit Casting (manually)

converting a larger type to a smaller size type double -> float -> long -> int -> char.

Explicit type conversion in C#, are done explicitly by users using the pre-defined functions. Explicit type conversion in C#, requires a cast operator.

```
using System;  
  
namespace ExplicitConversion  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {
```

```
// Explicit Type Casting
double d = 65.01;

// Explicit Type Casting
float f = (float)d;

// Explicit Type Casting
long l = (long)f;

// Explicit Type Casting
int i = (int)l;

// Explicit Type Casting
char c = (char)i;

// Display Result
Console.WriteLine("Value of double is " +d);
Console.WriteLine("Value of float is " +f);
Console.WriteLine("Value of long is " +l);
Console.WriteLine("Value of integer is " +i);
Console.WriteLine("Value of char is " +c);
}
}
}
```

Output:-

```
Value of double is 65.01
Value of float is 65.01
Value of long is 65
Value of integer is 65
Value of char is A
```

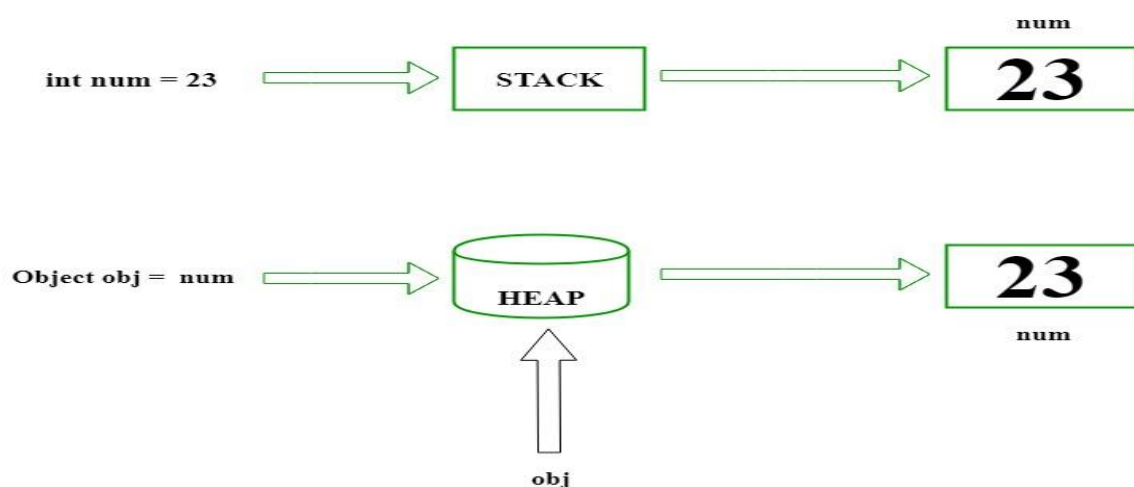
3) Boxing/Unboxing

Boxing In C# :-

- The process of Converting a Value Type (char, int etc.) to a Reference Type(object) is called **Boxing**.
- Boxing is implicit conversion process in which object type (super type) is used.
- The Value type is always stored in Stack. The Referenced Type is stored in Heap.
- Boxing is used to store value types in the garbage-collected heap.
- Example =
`int num = 23; // 23 will assigned to num`

`Object Obj = num; // Boxing`

- **Description :** First declare a value type variable (num), which is integer type and assigned it with value 23. Now create a references object type (obj) and applied Implicit operation which results in num value type to be copied and stored in object reference type obj as shown in below figure :



- Let's understand **Boxing** with a C# programming code :

```
using System;

namespace Boxing
{
    class Program
    {
        static void Main(string[] args)
```

```
{
    // assigned int value
    // 2020 to num
    int num = 2021;

    // boxing
    object obj = num;

    // value of num to be change
    num = 100;

    System.Console.WriteLine
    ("Value - type value of num is : {0}", num);
    System.Console.WriteLine
    ("Object - type value of obj is : {0}", obj);
}
}
```

Output:-

```
Value - type value of num is : 100
Object - type value of obj is : 2021
```

Unboxing In C# :-

- The process of converting reference type **into the** value type is known as **Unboxing**.
- It is explicit conversion process.
- **Example :**
`int num = 23; // value type is int and assigned value`
`Object Obj = num; // Boxing`
`int i = (int)Obj; // Unboxing`
- **Description :** Declaration a value type variable (num), which is integer type and assigned with integer value 23. Now, create a reference object type (obj). The explicit operation for boxing create an value type integer i and applied casting method. Then the referenced type residing on Heap is copy to stack as shown in below figure :



- Let's understand **Unboxing** with a C# programming code :

```
using System;

namespace Unboxing
{
    class Program
    {
        static void Main(string[] args)
        {
            // assigned int value
            // 23 to num
            int num = 17;

            // boxing
            object obj = num;

            // unboxing
            int i = (int)obj;
```



```
// Display result
Console.WriteLine("Value of ob object is : " + obj);
Console.WriteLine("Value of i is : " + i);
}
}
}
```

Output:-

```
Value of ob object is : 17
Value of i is : 17
```

Module-7

➤ Understanding Decision making & statements

- if else, switch

Decision Making in programming is similar to decision making in real life. In programming too, a certain block of code needs to be executed when some condition is fulfilled.

A programming language uses control statements to control the flow of execution of program based on certain conditions.

1) If Statement

The if statement checks the given condition. If the condition evaluates to be true then the block of code/statements will execute otherwise not.

Syntax:

```
if(condition)
{
    //code to be executed
}
```

Note: If the curly brackets { } are not used with if statements than the statement just next to it is only considered associated with the if statement.

Example:

```
using System;

namespace IfStatement
{
    class Program
    {
        static void Main(string[] args)
        {
            string name = "Amisha";
            if (name == "Amisha") {
                Console.WriteLine("Amisha Kotadiya");
            }
        }
    }
}
```

Output:-



```
Amisha Kotadiya
```

2) If else Statment

The if statement evaluates the code if the condition is true but what if the condition is not true, here comes the else statement. It tells the code what to do when the if condition is false.

Syntax:


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

Example:

```
using System;

namespace IfelseStatement
{
    class Program
    {
        static void Main(string[] args)
        {
            string name = "Amisha";
            if (name == "Amisa") {
                Console.WriteLine("Amisha Kotadiya");
            }
            else {
                Console.WriteLine("Amisha");
            }
        }
    }
}
```

Output:-

A screenshot of a console window with a black background and a green title bar. The word "Amisha" is printed in white text at the top left of the window.

Amisha

3) If else if Ladder

The if-else-if ladder statement executes one condition from multiple statements. The execution starts from top and checked for each if condition. The statement of if block will be executed which evaluates to be true. If none of the if condition evaluates to be true then the last else block is evaluated.

Syntax:

```
if(condition1)
{
    // code to be executed if condition1 is true
}
else if(condition2)
{
    // code to be executed if condition2 is true
}
else if(condition3)
{
    // code to be executed if condition3 is true
}
...
else
{
    // code to be executed if all the conditions are false
}
```

Example:

```
using System;

namespace IfelseifStatement
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 20;

            if (i == 10)
                Console.WriteLine("i is 10");
            else if (i == 15)
                Console.WriteLine("i is 15");
            else if (i == 20)
                Console.WriteLine("i is 20");
            else
                Console.WriteLine("i is not present");
        }
    }
}
```

Output:-

```
i is 20
```

4) Nested If

If statement inside an if statement is known as nested if. if statement in this case is the target of another if or else statement. When more than one condition needs to be true and one of the condition is the sub-condition of parent condition, nested if can be used.

Syntax:

```
if (condition1)
{
    // code to be executed
    // if condition2 is true
    if (condition2)
    {
        // code to be executed
        // if condition2 is true
    }
}
```

Example:

```
using System;

namespace NestedifStatement
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 10;

            if (i == 10)
            {
                // Nested - if statement
                if (i < 15)
                    Console.WriteLine("i is smaller than 15 too");
                else
                    Console.WriteLine("i is greater than 15");
            }
        }
    }
}
```



```
}  
}  
}
```

Output:-

```
i is smaller than 15 too
```

5) Switch

Switch statement is an alternative to long if-else-if ladders. The expression is checked for different cases and the one match is executed. **break** statement is used to move out of the switch. If the break is not used, the control will flow to all cases below it until break is found or switch comes to an end. There is **default case (optional)** at the end of switch, if none of the case matches then default case is executed.

Syntax:

```
switch (expression)  
{
```


Output:-



Cat

6) Nested Switch

Nested Switch case are allowed in C# . In this case, switch is present inside other switch case. Inner switch is present in one of the cases in parent switch.

Example:

```
using System;

namespace NestedswitchStatement
{
    class Program
    {
        static void Main(string[] args)
        {
            int j = 5;

            switch (j)
            {
                case 5: Console.WriteLine(5);
                    switch (j - 1)
```

```
        {
            case 4: Console.WriteLine(4);
                switch (j - 2)
                {
                    case 3: Console.WriteLine(3);
                        break;
                }
                break;
        }
        break;
    case 10: Console.WriteLine(10);
        break;
    case 15: Console.WriteLine(15);
        break;
    default: Console.WriteLine(100);
        break;
    }
}
}
```

Output:-

```
5
4
3
```