2. Basics of C#

2.1 Introduction to C#

- It is an object-oriented programming language created by Microsoft that runs on the .NET Framework.
- C# has roots in the C family, and the language is close to other popular languages like C++ and Java.
- C# is an object-oriented language that gives a clear structure to programs and allows code to be reused, lowering development costs.
- Uses of C# :

1. Mobile applications
2. Desktop applications
3. Web applications
4. Web services
5. Web sites
6. Games
7. VR
8. Database applications, etc

2.2 Understanding C# program structure

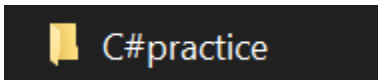
```
using System;

class Hello
{
    static void Main()
    {
        // This line prints "Hello, World"
        Console.WriteLine("Hello, World");
    }
}
```

- C# program starts with a using directive that references the System namespace.
- class is a container for data and methods, which brings functionality to your program. Every line of code in C# must be inside a class.
- Main method is a must for any C# program because it's an entry point of execution.

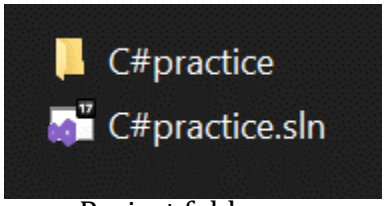
- Console is a class of the System namespace, which has a WriteLine() method that is used to output/print text.
- C# is case-sensitive and every statement ends with semi-colon.
- In C# single line comments starts with // and multi-line comments defines between /* ..... */.

2.3 Working with Code files, Projects & Solutions

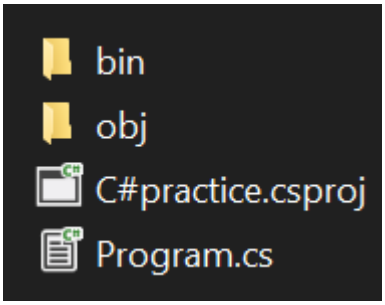
- In visual studio code we can create a c# project and can set path at which solution folder should be created and we can also give solution name.
- This solution folder is of same name that we given as solution while creating project.
- Solution folder contains solution file for vs code as well as another folder named same as project name given by us.
- Usually project folder contains bin folder, obj folder, project, program files.
- One solution can have multiple project based on application complexity that we are developing.



Solution folder



Project folder
Solution file



Project folder

2.4 Datatypes & Variables with conversion

Data Types

- C# supports int, long, string, bool, float, double, char etc data types, We can use them as per our requirements.
- Example :

```
int myNum = 5;
long myLongNum = 11001010010L;
double myDoubleNum = 5.99D;
float myFloatNum = 5.23F;
char myLetter = 'D';
bool myBool = true;
string myText = "Hello";
```

Variables

- We can declare variable as const and we can also define multiple variables in single line.
- Example :
  Const int age = 20;
  int num1 = 10, num2 = 20;

Type Casting

- Type casting is when you assign a value of one data type to another type.
- There is two type of type casting :
  1. Implicit type casting (automatically)
     - Implicit casting is done automatically when passing a smaller size type to a larger size type.
     - Example :
       int myInt = 9;
       double myDouble = myInt;
  2. Explicit type casting (manually)
     - Explicit casting must be done manually by placing the type in parentheses in front of the value.
     - Example :
       double myDouble = 9.78;
       int myInt = (int) myDouble;

2.5 Operators & Expressions

- Operators are used to perform operations on variables and values.
- There are different types of operators as follows :

1. Arithmetic operators

   Arithmetic operators are used to perform common mathematical operations:

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*****	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

2. Assignment operators

   Assignment operators are used to assign values to variables.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

3. Comparison operators

   Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions. The return value of a comparison is either True or False.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

4. Logical operators

   Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns True if both statements are true	x < 5 && x <10
||	Logical or	Returns True if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns False if the result is true	!(x < 5 && x < 10)

- An expression in C# is a combination of operands (variables, literals, method calls) and operators that can be evaluated to a single value. To be precise, an expression must have at least one operand but may not have any operator.
- Let's look at the example below:

```
double temperature;

temperature = 42.05;
```

- Here, 42.05 is an expression. Also, temperature = 42.05 is an expression too.

2.6 Statements

- A statement is a basic unit of execution of a program. A program consists of multiple statements.
- For example:

```
int age = 21;
Int marks = 90;
```

- In the above example, both lines above are statements.
- There are different types of statements in C#. Mainly focus on two of them here :
  1. Declaration Statement
  2. Expression Statement
- Declaration Statement

  Declaration statements are used to declare and initialize variables.

  For example:

  char ch;

  int maxValue = 55;

  Both char ch; and int maxValue = 55; are declaration statements.
- Expression Statement

  An expression followed by a semicolon is called an expression statement.

  For example:

```
/* Assignment */
area = 3.14 * radius * radius;
/* Method call is an expression*/
System.Console.WriteLine("Hello");
```

- Here, 3.14 * radius * radius is an expression and area = 3.14 * radius * radius; is an expression statement.
- Likewise, System.Console.WriteLine("Hello"); is both an expression and a statement.
- Beside declaration and expression statement, there are:
  - Selection Statements (if...else, switch)
  - Iteration Statements (do, while, for, foreach)
  - Jump Statements (break, continue, goto, return, yield)

- Exception Handling Statements (throw, try-catch, try-finally, try-catch-finally)

2.7 Understanding Arrays

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- An array can be single-dimensional, multidimensional, or jagged.
- The number of dimensions are set when an array variable is declared. The length of each dimension is established when the array instance is created. These values can't be changed during the lifetime of the instance.
- A jagged array is an array of arrays, and each member array has the default value of null.
- Arrays are zero indexed: an array with n elements is indexed from 0 to n-1.
- Array elements can be of any type, including an array type.
- Array types are reference types derived from the abstract base type Array.

The following example creates single-dimensional, multidimensional, and jagged arrays:

```
// Declare a single-dimensional array of 5 integers.
int[] array1 = new int[5];

// Declare and set array element values.
int[] array2 = { 1, 2, 3, 4, 5, 6 };

// Declare a two dimensional array.
int[,] multiDimensionalArray1 = new int[2, 3];

// Declare and set array element values.
int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

// Declare a jagged array.
int[][] jaggedArray = new int[3][];

// Set the values in the jagged array structure.
jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };
jaggedArray[1] = new int[] { 0, 2, 4, 6 };
jaggedArray[2] = new int[] { 11, 22 };
```

2.8 Define & calling of Methods

- A method is a code block that contains a series of statements. A program causes the statements to be executed by calling the method and specifying any required method arguments.
- Method signatures

  Methods are declared in a class, record, or struct by specifying:

  1. An optional access level, such as public or private. The default is private.
  2. Optional modifiers such as abstract or sealed.
  3. The return value, or void if the method has none.
  4. The method name.
  5. Any method parameters. Method parameters are enclosed in parentheses and are separated by commas. Empty parentheses indicate that the method requires no parameters.

  These parts together form the method signature.
- Calling a method

  Calling a method is like accessing a field.

  When a caller invokes the method, it provides concrete values, called arguments, for each parameter. The arguments must be compatible with the parameter type, but the argument name, if one is used in the calling code, doesn't have to be the same as the parameter named defined in the method.
- Example :

```
public class SquareExample
{
        public static void Main()
        {
                // Call with an int variable.
                int num = 4;
                int productA = Square(num);
                // Call with an integer literal.
                int productB = Square(12);
                // Call with an expression that evaluates to int.
                int productC = Square(productA * 3);
        }
        static int Square(int i)
```

```
    {
        // Store input argument in a local variable.
        int input = i;
        return input * input;
    }
}
```

2.9 Understanding classes & OOP concepts

- C# is an object-oriented programming language. The four basic principles of object-oriented programming are:
- Abstraction Modeling the relevant attributes and interactions of entities as classes to define an abstract representation of a system.
- Encapsulation Hiding the internal state and functionality of an object and only allowing access through a public set of functions.
- Inheritance Ability to create new abstractions based on existing abstractions.
- Polymorphism Ability to implement inherited properties or methods in different ways across multiple abstractions.
- A class is a template for objects, and an object is an instance of a class.
- When the individual objects are created, they inherit all the variables and methods from the class.
- Example of class :

```
public class Animal{
  public static int noOfFeet = 4;
  public string? animalType;
  // Displays sound of animal
  public void Sound()
  {
    Console.WriteLine("Animal sound...");
  }
}
```

2.10 Interface & Inheritance

- Another way to achieve abstraction in C#, is with interfaces.
- An interface is a completely "abstract class", which can only contain abstract methods and properties (with empty bodies):
- Example of Interface :

```
interface Animal {
    void animalSound();
    void run();
}
```

- Example of Inheritance :

```
public class Animal{
   public static int noOfFeet = 4;
   public string? animalType;
   // Displays sound of animal
   public void Sound(){
      Console.WriteLine("Animal sound...");
   }
}
public class Dog : Animal{
   public string bark;
   public Dog() { }
   public Dog(string animalType, string bark) : base(){
      this.animalType = animalType;
      this.bark = bark;
   }
   // Overrides sound() method and displays sound of Dog
   public void Sound(){
      Console.WriteLine("Dog sound : " + this.bark);
   }
}
```

2.11 Scope & Accessibility modifier

- Access modifiers in C# are used to specify the scope of accessibility of a member of a class or type of the class itself.
- In C# there are 6 different types of Access Modifiers.

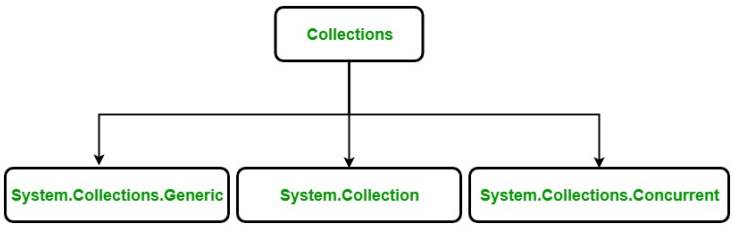
Modifier	Description
public	There are no restrictions on accessing public members.
private	Access is limited to within the class definition. This is the default access modifier type if none is formally specified
protected	Access is limited to within the class definition and any class that inherits from the class
internal	Access is limited exclusively to classes defined within the current project assembly
protected internal	Access is limited to the current assembly and types derived from the containing class. All members in current project and all members in derived class can access the variables.
private protected	Access is limited to the containing class or types derived from the containing class within the current assembly.

2.12 Working with Collections

- Collections standardize the way of which the objects are handled by your program. In other words, it contains a set of classes to contain elements in a generalized manner.

With the help of collections, the user can perform several operations on objects like the store, update, delete, retrieve, search, sort etc.

- C# divide collection in several classes, some of the common classes are shown below:



2.13 Enumerations

- An enum is a special "class" that represents a group of constants (unchangeable/read-only variables).
- To create an enum, use the enum keyword (instead of class or interface), and separate the enum items with a comma:
- Example :

```
enum Level {
    Low,
    Medium,
    High
}
```

- You can access enum items with the dot syntax:

```
Level myVar = Level.Medium;
Console.WriteLine(myVar);
```

2.14 Data Table

- The DataTable class in C# ADO.NET is a database table representation and provides a collection of columns and rows to store data in a grid form. The code sample in this artilce explains how to create a DataTable at run-time in C#.
- Example :

```
public static DataTable CreateOrderDetailTable(){
    DataTable orderDetailTable = new DataTable("OrderDetail");
    DataColumn[] cols ={
        new DataColumn("OrderDetailId", typeof(Int32)),
        new DataColumn("OrderId", typeof(String)),
        new DataColumn("Product", typeof(String)),
        new DataColumn("UnitPrice", typeof(Decimal)),
        new DataColumn("OrderQty", typeof(Int32)),
        new DataColumn("LineTotal", typeof(Decimal),
        "UnitPrice*OrderQty")
    };
    orderDetailTable.Columns.AddRange(cols);
    orderDetailTable.PrimaryKey = new DataColumn[] {
    orderDetailTable.Columns["OrderDetailId"] };
```

```
        return orderDetailTable;
}
public static void InsertOrderDetails(DataTable orderDetailTable){
        Object[] rows = {
                new Object[] { 1, "O0001", "Mountain Bike", 1419.5, 36 },
                new Object[] { 2, "O0001", "Road Bike", 1233.6, 16 },
                new Object[] { 3, "O0001", "Touring Bike", 1653.3, 32 },
                new Object[] { 4, "O0002", "Mountain Bike", 1419.5, 24 },
                new Object[] { 5, "O0002", "Road Bike", 1233.6, 12 },
                new Object[] { 6, "O0003", "Mountain Bike", 1419.5, 48 },
                new Object[] { 7, "O0003", "Touring Bike", 1653.3, 8 },
        };

        foreach (Object[] row in rows){
                orderDetailTable.Rows.Add(row);
        }
}
```

2.15 Exception Handling

- When an error occurs, C# will normally stop and generate an error message. The technical term for this is: C# will throw an exception (throw an error).
- The try statement allows you to define a block of code to be tested for errors while it is being executed.
- The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The finally statement lets you execute code, after try...catch, regardless of the result.
- Example :

```
try{
  int[] myNumbers = {1, 2, 3};
  Console.WriteLine(myNumbers[10]);
}
catch (Exception e){
  Console.WriteLine("Something went wrong.");
}
finally{
  Console.WriteLine("The 'try catch' is finished.");
}
```

2.16 Different Project types

Project Type	Description

Class library	Component library with no user interface
Console application	Command line application
Database project	SQL script storage
Device application	Windows application for a smart device
Empty project	Blank project
SQL Server project	Management of stored procedures and SQL Server objects
Web service	ASP.NET Web application with no user interface; technically, no longer a project type
Web site	ASP.NET Web application; technically, no longer a project type
Windows	Windows application with a user interface application
Windows service	Windows application with no user interface
WPF Browser Application	Windows Presentation Foundation browser application

2.17 Working with String Class

- 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.
- Some methods :

Method	Description
Clone()	Returns a reference to this instance of String.
Compare()	Used to compare the two string objects.
CompareOrdinal(String, Int32, String, Int32, Int32)	Compares substrings of two specified String objects by evaluating the numeric values of the corresponding Char objects in each substring.
CompareOrdinal(String, String)	Compares two specified String objects by evaluating the numeric values of the corresponding Char objects in each string.
CompareTo()	Compare the current instance with a specified

2.18 Working with DateTime Class

- C# DateTime is a structure of value Type like int, double etc. It is available in System namespace and present in mscorlib.dll assembly. It implements interfaces like IComparable, IFormattable, IConvertible, ISerializable, IComparable, IEquatable.

- Example :

```
// Creating TimeSpan object of one month(as 30 days)
System.TimeSpan duration = new System.TimeSpan(30, 0, 0, 0);
System.DateTime newDate1 = DateTime.Now.Add(duration);
System.Console.WriteLine(newDate1); // 1/19/2016 11:47:52 AM
// Adding days to a date
DateTime today = DateTime.Now; // 12/20/2015 11:48:09 AM
```

```
DateTime newDate2 = today.AddDays(30); // Adding one month(as 30 days)
Console.WriteLine(newDate2); // 1/19/2016 11:48:09 AM
// Parsing
string dateString = "Wed Dec 30, 2015";
DateTime dateTime12 = DateTime.Parse(dateString); // 12/30/2015 12:00:00
AM
// Date Difference
System.DateTime date1 = new System.DateTime(2015, 3, 10, 2, 15, 10);
System.DateTime date2 = new System.DateTime(2015, 7, 15, 6, 30, 20);
System.DateTime date3 = new System.DateTime(2015, 12, 28, 10, 45, 30);

// diff1 gets 127 days, 04 hours, 15 minutes and 10 seconds.
System.TimeSpan diff1 = date2.Subtract(date1); // 127.04:15:10
// date4 gets 8/23/2015 6:30:20 AM
System.DateTime date4 = date3.Subtract(diff1);
// diff2 gets 166 days 4 hours, 15 minutes and 10 seconds.
System.TimeSpan diff2 = date3 - date2; //166.04:15:10
// date5 gets 3/10/2015 2:15:10 AM
System.DateTime date5 = date2 - diff1;
// Universal Time
DateTime dateObj = new DateTime(2015, 12, 20, 10, 20, 30);
Console.WriteLine("mans" + dateObj.ToUniversalTime()); // 12/20/2015
4:50:30 AM
```

2.19 Basic File operations

- The following table describes some commonly used classes in the System.IO namespace.

Class Name	Description
FileStream	It is used to read from and write to any location within a file
BinaryReader	It is used to read primitive data types from a binary stream
BinaryWriter	It is used to write primitive data types in binary format
StreamReader	It is used to read characters from a byte Stream

StreamWriter	It is used to write characters to a stream.
StringReader	It is used to read from a string buffer
StringWriter	It is used to write into a string buffer
DirectoryInfo	It is used to perform operations on directories
FileInfo	It is used to perform operations on files