**Part 2**

**Basic of C#**

|  |  |
| --- | --- |
| **2.1** | **Introduction to C#** |
|  | C# (pronounced "C-sharp") is a modern, versatile, and object-oriented programming language developed by Microsoft. It is widely used for developing a variety of software applications, from Windows desktop applications to web services, games, and mobile apps.  Visual Studio, Microsoft's integrated development environment (IDE), is commonly used for C# development.  C# applications are typically built on the .NET Framework or .NET Core. |
| **2.2** | **Create first C# program ‘Hello world’** |
|  |  |
| **2.3** | **Understanding C# program structure** |
|  | Consider the following example of C#:  using System;  namespace HelloWorld {  class Hello {  static void Main(string[] args) {  /\* first program in C# \*/  Console.WriteLine("Hello World!");  }  }  }  • The first line of the program using System; - the using keyword is used to include the System namespace in the program.  • The next line has the namespace declaration. A namespace is a collection of classes. The HelloWorld namespace contains the class Hello.  • The next line has a class declaration, the class Hello contains the data and method definitions that our program uses. Classes generally contain multiple methods. However, the HelloWorld class has only one method Main.  • The next line defines the Main method, which is the entry point for all C# programs. The Main method represents what the class does when executed.  • The next line /\*...\*/ is ignored by the compiler and it is put to add comments in the program.  • The Main method specifies its behavior with the statement Console. WriteLine("Hello World"); |
| **2.4** | **Working with Code files, Projects & Solutions** |
|  | **Understanding structure of project (Windows app, class library , Console App)**  **• Structure of Windows forms app**   * File name is same as app name and [appname].csproj file contains references of other projects and of packages used and also version of project etc. * It has a .cs file which contains code of windows form that we create. * It also contains Program.cs file which has Main method as required to run app as console application. * The configuration for app is in CreateHostBuilder Method called in Main method.   **• Structure of Class Library**   * File name is same as app name and [appname].csproj file contains references of other projects and of packages used and also version of project etc. * It has a .cs file which contains code of a particular class that we define in it.   **• Console Application:**   * This project type is used to create console-based applications that run in a command-line interface. |
| **2.5** | **Datatypes & Variables with conversion** |
|  | The datatypes in C#, are categorized into the following types −  1. Value types  2. Reference types  3. Pointer types   * **Value type** variables can be assigned a value directly. They are derived from the class System.ValueType. * Some examples are int, char, and float, double, long, sbyte, short, ulong etc. * The **reference types** do not contain the actual data stored in a variable, but they contain a reference to the variables. * Example of built-in reference types are: object, dynamic, and string. * The **Object Type** is the ultimate base class for all data types in CTS (commen type system) * Any type of value can be stored in the **dynamic data type** variable. * The **String Type** allows you to assign any string values to a variable. * **Pointer type** variables store the memory address of another type.   **Datatype Conversion:**  Two types of conversion:   1. Implicit conversion 2. Explicit conversion   **IMPLICIT CONVERSION:**   * It is done by the compiler. * It is done when there is no loss of information. * It there is no possibility of throwing exception during the conversion. * For example: converting int to float will not lose any data and no exception will be thrown. * But while converting float to int, we lose the fractional part and also possibility of overflow exception. * Therefore in this case explicit conversion is required. * **EXPLICIT CONVERSION:** * It is done by the user. * Convert class can be used for explicit conversion. * For example : Convert.ToInt32() can be used to convert string value to integer. * Int.parse() or float.parse() is used for explicit conversion. |
| **2.6** | **Operators & Expressions** |
|  | **Expressions**: An expression in C# is a combination of operands (variables, literals, method call) and operators that can be evaluated to a single value.  An expression must have at least one operand but may not have any operator.  **Operators:** An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.  C# has a rich set of built-in operators and provides the following type of operators.  • **Arithmetic Operators:** +,-,\*,/,%,++,--  • **Relational Operators:** ==,!=,>,=,<=  • **Logical Operators:** &&,||,!  • **Bitwise Operators:** `,&,|,^,<>  • **Assignment Operators:** =,+=,-=,\*=,/=,%=,<<=,>>=,&=,^=,|=  • **Miscellaneous Operators:** sizeof(), typeof(), ?:  **Arithmetic Operators:** The arithmetic operators perform arithmetic operations on all the numeric type operands.  **Relational Operators:** Relational Operators are useful to check the relation between two operands like we can determine whether two operand values equal or not, etc.,  **Logical Operators:** Logical Operators are useful to perform the logical operation between two operands like AND, OR, and NOT based on our requirements.  **Bitwise Operators:** It will work on bits, and these are useful to perform bit-by-bit operations such as Bitwise AND (&), Bitwise OR (|), etc.  **Assignment Operators:** The assignment operator assigns the value of its right-hand operand to a variable.  **Miscellaneous Operators:** It performs some miscellaneous tasks like Returns the size of a data type, Returns the type of a class, etc |
| **2.7** | **Statements** |
|  | **if** Statement contains a Boolean Condition which if it is true than the code inside its block is executed.  **else if** Statement contains a Boolean Condition which if it is true than the code inside its block is executed but also the condition above this statement have to be false for it to happen.  **else** Statement code block is executed automatically after all the above conditions are found to be false   **Syntax -** if (condition 1)  {  // code block to be executed when if condition1 evaluates to true  }  else if(condition2)  {  // code block to be executed when  //condition1 evaluates to false  //condition2 evaluates to true  }  else if(condition3)  {  // code block to be executed when  // condition1 evaluates to false  // condition2 evaluates to false  //condition3 evaluates to true  }  else  {  // code block to be executed when  // condition1 evaluates to false  // condition2 evaluates to false  // condition3 evaluates to false  // any other condition is True  }   **Switch case** – It is an alternative used for if…else statement and there will be an expression or variable in switch bracket the result of which will be tested by value in each case and wherever it is equal to the condition the corresponding case’s code block will be executed and then it will break. It also has a default case which performs similarly to else statement.   **Syntax :**  switch(match expression/variable)  {  case value:  statement(s) to be executed;  break;  default:  statement(s) to be executed; |
| **2.8** | **Understanding Arrays** |
|  | * An array is a group of like-typed variables that are referred to by a common name. * And each data item is called an element of the array. * The data types of the elements may be any valid data type like char, int, float, etc. and the elements are stored in a contiguous location. * The length of the array specifies the number of elements present in the array. * The variables in the array are ordered and each has an index beginning from 0. * C# array is an object of base type System.Array   Array Declaration:  < Data Type > [ ] < Name\_Array >  Here,  < Data Type >: It defines the element type of the array.  [ ]: It defines the size of the array.  < Name\_Array >: It is the Name of the array.    Array Initialization:  < Data Type > [ ] < Name\_Array > = new < datatype > [size];  Here,  • New will allocate memory to an array according to its size.  • Type specifies the type of data being allocated, size specifies the number of elements in the array, and Name\_Array is the name of an array variable.  Types of Array:  1. One Dimensional Array  2. Multidimensional Array  3. Jagged Array  One Dimensional:  Array One dimensional array contains only one row for storing the values.  All values of this array are stored contiguously starting from 0 to the array size.  For example, declaring a single-dimensional array of 5 integers :  int[] arrayint = new int[5];  Multidimensional Array  The multi-dimensional array contains more than one row to store the values.  It is also known as a Rectangular Array because it’s each row length is the same.  It can be a 2D-array or 3D-array or more. To storing and accessing the values of the array, one required the nested loop.  The multi-dimensional array declaration, initialization, and access are as follows :  // creates a two-dimensional array of four rows and two columns.  int[, ] intarray = new int[4, 2];  Jagged Array  An array whose elements are arrays is known as Jagged arrays it means “array of arrays”.  The jagged array elements may be of different dimensions and sizes. |
| **2.9** | **Define & calling of Methods** |
|  | Define method and use   * Methods are generally the block of codes or statements in a program. * This gives the user the ability to reuse the same code which ultimately saves the excessive use of memory, acts as a time saver, and more importantly, it provides better readability of code. * A method is a collection of statements that perform some specific task and return the result to the caller. * A method can also perform some specific task without returning anything.   <Access\_Modifier> <return\_type> <method\_name>([<param\_list>])  {  // Body  }  different types of parameters in the method:  Value Type Parameters:   * It is a normal value parameter in a method or you can say the passing of value types by value. * So when the variables are passed as value types they contain the data or value, not any reference. * If you will make any changes in the value type parameter then it will not reflect the original value stored as an argument.   Reference Type Parameters:   * The ref is a keyword that is used for passing the value types by reference. * Or we can say that if any changes made in this argument in the method will reflect in that variable when the control return to the calling method. * In ref parameters, the parameters must initialize before it passes to ref. * The passing of value through the ref parameter is useful when the called method also needs to change the value of the passed parameter.   Default or Optional Type Parameters:   * As the name suggests optional parameters are not compulsory parameters, they are optional. * It helps to exclude arguments for some parameters. * Or we can say in optional parameters, it is not necessary to pass all the parameters in the method. * Here, every optional parameter contains a default value which is part of its definition. * If we do not pass any arguments to the optional parameters, then it takes its default value. * The optional parameters are always defined at the end of the parameter list. |
| **2.10** | **Understanding classes & OOP concepts** |
|  | Classes are the user-defined data types that represent the state and behaviour of an object. State represents the properties and behaviour is the action that objects can perform.  **Types of Classes**  1. Abstract class  2. Partial class  3. Sealed class  4. Static class  **Abstract Class :**   * An Abstract class is a class that provides a common definition to the subclasses and this is the type of class whose object is not created. * Abstract classes are declared using the abstract keyword. * We cannot create an object of an abstract class. * If you want to use it then it must be inherited in a subclass. * An Abstract class contains both abstract and non-abstract methods. * The methods inside the abstract class can either have an implementation or no implementation. * We can inherit two abstract classes; in this case the base class method implementation is optional. * An Abstract class has only one subclass. * Methods inside the abstract class cannot be private. * If there is at least one method abstract in a class then the class must be abstract.   **Partial Class:**   * A Partial class provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled.   + A partial class is created by using a partial keyword.   + This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.   + When you want to chop the functionality of the class, method, interface, or structure into multiple files, then you should use partial keyword and all the files are mandatory to be available at compile time for creating the final file.   + The partial modifier can only present instantly before the keywords like struct, class, and interface. * Every part of the partial class definition should be in the same assembly and namespace, but you can use a different source file name. * Every part of the partial class definition should have the same accessibility as private, protected, etc. * If any part of the partial class is declared as an abstract, sealed, or base, then the whole class is declared of the same type. * The user is also allowed to use nested partial types. * Dissimilar parts may have dissimilar base types, but the final type must inherit all the base types.     **Sealed Class**:   * Sealed classes are used to restrict the users from inheriting the class. * A class can be sealed by using the sealed keyword. * Access modifiers are not applied to a sealed class.   **Static Class:**   * A class with static keyword that contains only static members is defined as static class. * Static classes are sealed, means you cannot inherit a static class from another class. * Static class cannot instantiated using a new keyword. * A static method can only contain static variables and can only access other static items. * Static items share the resources between multiple users. * Static members are allocated in a high frequency heap area of the memory. * The methods of the static class can be called using the class name without creating the instance.   Oop concept:  Classes and Objects:   * A class is a blueprint or template for creating objects. * An object is an instance of a class. Objects encapsulate data and behavior.   Inheritance:   * Inheritance allows a class to inherit properties and behavior from another class. * The base class (parent) provides a common set of attributes and methods that can be reused by derived classes (children).   Encapsulation:   * Encapsulation is the bundling of data (fields) and methods (functions) that operate on the data within a single unit (class). * It helps in hiding the internal details of an object and exposing only what is necessary.   Polymorphism:   * Polymorphism allows objects to be treated as instances of their base class, even when they are instances of derived classes. * It enables methods to be called on objects of different types through a common interface.   Abstraction:   * Abstraction is the process of hiding the complex implementation details and showing only the essential features of an object. * Abstract classes and interfaces are used to achieve abstraction. |
| **2.11** | **Interface & Inheritance** |
|  | **Interface:**   * An interface looks like a class, but has no implementation. * The only thing it contains are declarations of events, indexers, methods or properties. * The reason interfaces only provide declarations is because they are inherited by structs and classes, that must provide an implementation for each interface member declared. * It is used to achieve multiple inheritance which can't be achieved by class. * Interfaces can’t have private members. * By default all the members of interfaces are public and abstract.   **Inheritance:**  Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.  **Types of Inheritance in C#:**   * Single Inheritance * Multilevel Inheritance * Hierarchical Inheritance * Multiple Inheritance (Through Interfaces) * Hybrid Inheritance (Through Interfaces) |
| **2.12** | **Scope & Accessibility modifier** |
|  | Accessibility Modifiers:  Public:   * Least restrictive. * Accessible from any code that can reference the assembly.   Protected Internal:   * Accessible within the same assembly or by derived classes.   Private:   * The most restrictive modifier. * Only accessible within the same class.   Protected:   * Accessible within the same class or by derived classes.   Internal:   * Accessible within the same assembly (executable or DLL). |
| **2.13** | **Namespace & .Net Library** |
|  | Namespace:   * Used to organize the classes. * To access class use namespace.classname. * For eg: System.Console.WriteLine, where System is namespace, Console is classname and WriteLine is the method. |
| **2.14** | **Creating & Adding Reference to Assemblies** |
|  | **Assemblies are basically of two types.**  1. Private Assembly: It is an assembly that is used by a single application only.  2. Shared Assembly: It is an assembly that is used by more than one projects.  **Steps for reference to assembly**  • Open solution explorer by (ctrl + alt + L)  • Choose Add Reference  • Browsr of choose available assembly and click ok. |
| **2.15** | **Working with Collections** |
|  | * C# collection types are designed to store, manage and manipulate similar data more efficiently. * Data manipulation includes adding, removing, finding, and inserting data in the collection. * .NET supports two types of collections, generic collections and non-generic collections * The System.Collections namespace contains the non-generic collection types * System.Collections.Generic namespace includes generic collection types.   **Generic Collection**    **Non-Generic Collection** |
| **2.16** | **Enumerations** |
|  | Enumerations (enums) in C# provide a way to define named integral constants. Enums make code more readable and self-explanatory by allowing you to use symbolic names to represent values.  You can define an enumeration using the enum keyword |
| **2.17** | **Data Table** |
|  | * In C#, a DataTable is a part of the ADO.NET library and is used to represent an in-memory relational data table. It provides a way to store and manipulate data in a tabular format, similar to a database table. * a new instance of the DataTable class is created * columns are added to the dataTable with the specified names and data types . Each column is represented by a DataColumn object. * rows are added to the dataTable, and the data for each row corresponds to the columns defined earlier. The Add method takes the values for each column in the order they were added. |
| **2.18** | **Exception Handling** |
|  | Exception handling in C# provides a way to gracefully handle errors and unexpected situations that may occur during program execution.  1. Try-Catch Block  2. Multiple Catch Blocks  3. Finally Block  4. Throw Statement  5. Custom Exceptions |
| **2.19** | **Working with String Class** |
|  | * With the help of the length property, it provides the total number of characters present in the given string. * String objects can include a null character which counts as the part of the string’s length. * It allows empty strings. Empty strings are the valid instance of String objects that contain zero characters. * It also supports searching strings, comparison of strings, testing of equality, modifying the string, copying of strings, etc. |
| **2.20** | **Working with DateTime Class** |
|  | * We used the DateTime when there is a need to work with the dates and times. * We can format the date and time in different formats by the properties and methods of the DateTime. * DateTime helps developers to find out more information about Date and Time like getting month, day, year, weekday. * It also helps to find date differences, add several days to a date, etc. |
| **2.21** | **Basic File operations** |
|  | * A file is a collection of data stored in a disk with a specific name and a directory path. * In C# Provides static methods for the creation, copying, deletion, moving, and opening of a single file. |