**C# Document**

**Varialbes:**

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

byte, short, int, long : Integer Values

char : Character values

true or false : Boolean values

float, double : Floating values

**Rules for defining variables**

* A variable can have alphabets, digits and underscore
* A variable name can start with alphabet and underscore only. It can't start with digit.
* No white space is allowed within variable name

Ex: int x=10;

Float f=2.4f;

Char c=’x’;

**Data Types :**

A data type specifies the type of data that a variable can store such as integer, floating, character etc.

1.Value Data types :

i**. Predefined Data Types**- such as Integer, Boolean, Float, etc.

**ii. User defined Data Types**- such as Structure, Enumerations, etc.

The memory size of data types may change according to 32 or 64 bit operating system.

2.Reference type Data types :

i. **Predefined Types** - such as Objects, String.

ii. User defined types -  such as Classes, Interface.

3.pointer defined types :

& - Address Operator : used to determine a address of a variable

\* Indirection Operator : used to determine a value of a variable.

**Operators :**

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C# language.

* Arithmetic Operators : + , - , \* , / . %
* Relational Operators : < ,<= , >, >=,!=,==
* Logical Operators : && , || , !
* Bitwise Operators : & ,|, <<,>>,~,^
* Assignment Operators : =,+=,-=,\*=,%=,/=
* Unary Operators : ++,--
* Ternary Operators : ?:

**Control Statements :**

1.If statement :

*if statement* is used to test the condition. There are various types of if statements in C#.

* if statement
* if-else statement
* nested if statement
* if-else-if ladder

2. Switch statement :

Syntax:

**switch**(expression){

**case** value1:

//code to be executed;

**break**;

**case** value2:

 //code to be executed;

**break**;

**default**:

 //code to be executed if all cases are not matched;

**break**;

}

* Break is mandatory for Switch if we are not using this then the control can not go to next condition.

3.For Loop :

The C# *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop than while or do-while loops.

The C# for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

**Syntax:**

**for**(initialization; condition; incr/decr)

{

//code to be executed

}

4.While loop :

In C#, *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.

**Syntax:**

**while**(condition){

//code to be executed  }

5.Do While loop :

It is similar to while loop but before checking the condition it will perform the operation. The  do-while loop is executed at least once because condition is checked after loop body.

Syntax:

**do**{

//code to be executed

}**while**(condition);

6.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.

**Syntax:**

jump-statement;

**break**;

7. Continue statement :

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;

**continue**;

8. goto 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.

**Function :**

Function is a block of code that has a signature. Function is used to execute statements specified in the code block. A function consists of the following components:

**Function name:** It is a unique name that is used to make Function call.

**Return type:** It is used to specify the data type of function return value.

**Body:** It is a block that contains executable statements.

**Access specifier:** It is used to specify function accessibility in the application.

**Parameters:** It is a list of arguments that we can pass to the function during call.

C# Function Syntax

<access-specifier><**return**-type>FunctionName(<parameters>)

{

// function body

// return statement

}

Access-specifier, parameters and return statement are optional.

**Call by value :**

In C#, value-type parameters are that pass a copy of original value to the function rather than reference. It does not modify the original value. A change made in passed value does not alter the actual value. In the following example, we have pass value during function call.

**Call by ref :**

C# provides a **ref** keyword to pass argument as reference-type. It passes reference of arguments to the function rather than copy of original value. The changes in passed values are permanent and **modify** the original variable value.

**Out Parameter :**

C# provides **out** keyword to pass arguments as out-type. It is like reference-type, except that it does not require variable to initialize before passing. We must use **out** keyword to pass argument as out-type. It is useful when we want a function to return multiple values.

**Arrays :**

array is a group of similar types of elements that have contiguous memory location array is an object of base type **System.Array**. array index starts from 0. We can store only fixed set of elements in C# array.

Advantages :

* Code Optimization (less code)
* Random Access
* Easy to traverse data
* Easy to manipulate data
* Easy to sort data etc.

Disadvantages :

* Fixed size

There are 3 types of arrays in C# programming:

1. Single Dimensional Array
2. Multidimensional Array
3. Jagged Array

**1.Single Dimensional Array** :

To create single dimensional array, you need to use square brackets [] after the type.

Ex: int [] a=new int[5];

int a []=new int[5];//compile time error

There are 3 ways to initialize array at the time of declaration.

1. **int**[] arr = **new** **int**[5]{ 10, 20, 30, 40, 50 };

We can omit the size of array.

1. **int**[] arr = **new** **int**[]{ 10, 20, 30, 40, 50 };

We can omit the new operator also.

1. **int**[] arr = { 10, 20, 30, 40, 50 };

**2.Multi dimensional Arrays :**

The multidimensional array is also known as rectangular arrays. It can be two dimensional or three dimensional. The data is stored in tabular form (row \* column) which is also known as matrix.

To create multidimensional array, we need to use comma inside the square brackets.

Ex:

1. **int**[,] arr=**new** **int**[3,3]; //declaration of 2D array
2. **int**[,,] arr=**new** **int**[3,3,3];//declaration of 3D array

There are 3 ways to initialize multidimensional array in C# while declaration.

1. **int**[,] arr = **new** **int**[3,3]= { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

We can omit the array size.

1. **int**[,] arr = **new** **int**[,]{ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

We can omit the new operator also.

1. **int**[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

**3.Jagged Arrays :**

In C#, jagged array is also known as "array of arrays" because its elements are arrays. The element size of jagged array can be different.

Declaration :

Let's see an example to declare jagged array that has two elements.

1. **int**[][] arr = **new** **int**[2][];

Initialization

Let's see an example to initialize jagged array. The size of elements can be different.

1. arr[0] = **new** **int**[4];
2. arr[1] = **new** **int**[6];

Initialization and filling elements in Jagged array

Let's see an example to initialize and fill elements in jagged arrayOOPs Concepts in Java

1. arr[0] = **new** **int**[4] { 11, 21, 56, 78 };
2. arr[1] = **new** **int**[6] { 42, 61, 37, 41, 59, 63 };

Here, size of elements in jagged array is optional. So, you can write above code as given below:

1. arr[0] = **new** **int**[] { 11, 21, 56, 78 };
2. arr[1] = **new** **int**[] { 42, 61, 37, 41, 59, 63 };

**Constructor :**

Constructor is mainly used to initialize the object. We are not returning any values. The constructor has the same name as class or struct.

There can be two types of constructors

* Default constructor :- A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.
* Parameterized constructor :- A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

**Destructor :**

A destructor works opposite to constructor, It destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

destructor cannot have parameters. Moreover, modifiers can not be applied on destructors.

Destructor can't be public. We can't apply any modifier on destructors.

**This Keyword** :

This is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword.

* It can be used **to refer current class instance variable**. It is used if field names (instance variables) and parameter names are same, that is why both can be distinguish easily.
* It can be used **to pass current object as a parameter to another method**.
* It can be used **to declare indexers**.

**Static :**

In C#, static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members. In C#, static can be field, method, constructor, class, properties, operator and event. **Indexers and destructors cannot be static.**

**Advantages :**

1.Memory Efficient

**Static Field** :

A field which is declared as static, is called static field. Unlike instance field which gets memory each time whenever you create object, there is only one copy of static field created in the memory. It is shared to all the objects.

It is used to refer the common property of all objects

If you change the value of static field, it will be applied to all the objects.

**Static Class :**

The C# static class is like the normal class but it cannot be instantiated. It can have only static members. The advantage of static class is that it provides you guarantee that instance of static class cannot be created.

Points to remember for C# static class

* static class contains only static members.
* static class cannot be instantiated.
* static class is sealed.
* static class cannot contain instance constructors.

**Static Constructor :**

C# static constructor is used to initialize static fields. It can also be used to perform any action that is to be performed only once. It is invoked automatically before first instance is created or any static member is referenced.

Note:

1. C# static constructor cannot have any modifier or parameter.

2. C# static constructor is invoked implicitly. It can't be called explicitly.

**Structs :**

In C#, classes and structs are blueprints that are used to create instance of a class. Structs are used for lightweight objects such as Color, Rectangle, Point etc.

Unlike class, structs in C# are value type than reference type. It is useful if you have data that is not intended to be modified after creation of struct.

Note: **Struct doesn't support inheritance. But it can implement interfaces.**

**Enum :**

Enum in C# is also known as enumeration. It is used to store a set of named constants such as season, days, month, size etc. The enum constants are also known as enumerators. Enum in C# can be declared within or outside class and structs.

Enum constants has default values which starts from 0 and incremented to one by one. But we can change the default value.

Note :

* enum has fixed set of constants
* enum improves type safety
* enum can be traversed

Oops Concepts :

Oops stands for Object Oriented Programming.

There are 4 types of Oops concepts.

1.Inheritance

2.Encapsulation

3.Abstraction

4.Polymorphism

**Inheritance :**

Inheritance is defined as acquiring the properties of super class from the sub class is called as inheritance. In such way, you can reuse, extend or modify the attributes and behaviors which is defined in other class.

The class which inherits the members of another class is called **derived class** and the class whose members are inherited is called **base** class. The derived class is the specialized class for the base class.

Advantages : Code Reusability

There are 4 types of Inheritances

1.Single level Inheritance

2.Multilevel Inheritance

3.Heirarichal level Inheritance

4.Multiple Inheritance

**Aggregation :**

aggregation is a process in which one class defines another class as any entity reference. It is another way to reuse the class. It is a form of association that represents HAS-A relationship.

**Overloading :**

If we create two or more members having same name but different in number or type of parameter, it is known as member overloading. In C#, we can overload:

* methods,
* constructors
* indexed properties

**Method Overloading :**

Having two or more methods with same name but different in parameters, is known as method overloading in C#.

The **advantage** of method overloading is that it increases the readability of the program because you don't need to use different names for same action.

You can perform method overloading by two ways:

1. By changing number of arguments
2. By changing data type of the arguments

**Method Overriding :**

If derived class defines same method as defined in its base class, it is known as method overriding in C#. It is used to achieve runtime polymorphism. It enables you to provide specific implementation of the method which is already provided by its base class.

To perform method overriding in C#, you need to use **virtual** keyword with base class method and **override** keyword with derived class method.

**Polymorphism :**

The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a greek word. In object-oriented programming, we use 3 main concepts: inheritance, encapsulation and polymorphism.

There are two types of polymorphism in C#:

1.Compile time polymorphism

2.Runtime polymorphism.

Compile time polymorphism is achieved by method overloading and operator overloading in C#. It is also known as static binding or early binding. Runtime polymorphism in achieved by method overriding which is also known as dynamic binding or late binding.

Note: Runtime Polymorphism can't be achieved by data members.

**Sealed** :

Sealed keyword is used to give restrictions. It is used for data members, methods and Classes also.

1.If we use for data members we **can’t change** that value.

2.If we use for methods then that methods are can not be **overridden**

3.If we use for classes then the classes are not **inherited**

**Abstraction :**

Abstraction is defined as Hiding the functionality and providing necessary functionality. Abstraction can be achieved in 2 ways

1.Abstarct classes

2.Interface

**Abstract class** :

In abstract classes consisting of both concrete and abstract methods. It can not be instantiated

Abstarct method : A method which is declared abstract and has no body is called abstract method. It can be declared inside the abstract class only. Its implementation must be provided by derived classes

Note: An abstract method is internally a virtual method so it can be overridden by the derived class.

You can't use static and virtual modifiers in abstract method declaration.

**Interface** :

Interface in C# is a blueprint of a class. It is like abstract class because all the methods which are declared inside the interface are abstract methods. It cannot have method body and cannot be instantiated.

It is used to achieve multiple inheritance which can't be achieved by class. It is used to achieve fully abstraction because it cannot have method body.

Its implementation must be provided by class or struct. The class or struct which implements the interface, must provide the implementation of all the methods declared inside the interface.

Note : Interface methods are public and abstract by default. You cannot explicitly use public and abstract keywords for an interface method.

using System;

public interface Drawable

{

    public abstract void draw();//Compile Time Error

}

**Access Modifiers :**

Access modifiers are used to give visibility and reachability

There are 4 types of Access modifiers

1.Public modifiers

2.Protected modifiers

3.Internal modifiers

4.Protected internal

5. Private modifiers

**Public Modifiers :**

It makes data accessible publicly. It does not restrict data to the declared block.

**Protected Modifiers :**

It is accessible within the class and has limited scope. It is also accessible within sub class or child class, in case of inheritance.

**Internal Modifiers :**

The internal keyword is used to specify the internal access specifier for the variables and functions. This specifier is accessible only within files in the same assembly.

**Protected Internal Modifiers :**

Variable or function declared **protected internal** can be accessed in the assembly in which it is declared. It can also be accessed within a derived class in another assembly.

**Private Modifiers :**

When a private modifier is used then that is visible with in the class in which it is declared . We can’t access from outside the class.

**Encapsulation :**

It is a phenomenon where wrapping a data members and a functions into a single unit cell is called as a Encapusaltion. It is also called as data hiding. To achieve Encapsulation we can make that variables private so we can not access them directly.To access those variables we are having getters and setters

The purpose of encapsulation is to prevent alteration of data from outside. This data can only be accessed by getter functions of the class.

A fully encapsulated class has getter and setter functions that are used to read and write data. This class does not allow data access directly.

**Strings :**

In C#, string is an object of **System.String** class that represent sequence of characters. We can perform many operations on strings such as concatenation, comparision, getting substring, search, trim, replacement etc.

## **string vs String**

In C#, string is keyword which is an alias for System.String class. That is why string and String are equivalent. We are free to use any naming convention.

1. string s1 = "hello";//creating string using string keyword
2. String s2 = "welcome";//creating string using String class

**Generics :**

Generic is a concept that allows us to define classes and methods with placeholder. C# compiler replaces these placeholders with specified type at compile time. The concept of generics is used to create general purpose classes and methods.

o define generic class, we must use angle **<>** brackets. The angle brackets are used to declare a class or method as generic type. In the following example, we are creating generic class that can be used to deal with any type of data.

C# allows us to create generic methods also. In the following example, we are creating generic method that can be called by passing any type of argument.

**Delegates** :

In C#, delegate is a reference to the method. It works like function pointer in C and C++. But it is objected-oriented, secured and type-safe than function pointer.

For static method, delegate encapsulates method only. But for instance method, it encapsulates method and instance both.

The best use of delegate is to use as event.

Internally a delegate declaration defines a class which is the derived class of **System.Delegate**.