**C SHARP NOTES**

**Day-1**

1. **Casing in C#**

* **Camel Case:**

Camel case is a naming convention where each word in an identifier begins with a lowercase letter, and subsequent words are capitalized.

Typically, the first word starts with a lowercase letter, and the first letter of each subsequent word is capitalized.

Example: myVariableName, calculateTotalAmount, getUserInfo

* **Pascal Case:**

Pascal case is a naming convention where each word in an identifier begins with an uppercase letter, and subsequent words are also capitalized.

Typically, the first word starts with an uppercase letter, and the first letter of each subsequent word is capitalized.

Pascal case is often used for naming classes, types, and methods in C#.

Example: MyClass, CalculateTotalAmount, GetUserInfoq

**Example**:

public class MyClass

{

public int MyProperty { get; set; }

public void MyMethod()

{

int myLocalVariable = 42;

string myStringVariable = "Hello, World!";

}

}

**Q: For defining constant or global variable which type of casing we used ?**

**Ans:** For defining constants or global variables in C#, it is common practice to use Pascal case. This helps distinguish them from regular variables and makes it clear that they are constants or global variables.

Example:

public class ConstantsAndGlobals

{

public const int MaxItemCount = 100;

public static string GlobalString = "Hello, World!";

}

MaxItemCount is a constant and follows Pascal case.

Global String is a global variable and also follows Pascal case.

**Q: Difference between Variable and properties in class?**

**Ans:**

**Variables (Field Variables):**

* Variables, also known as field variables or member variables, are data storage locations within a class.
* They are typically declared at the class level and can hold values that are associated with the class's state.
* Variables can have various access modifiers (public, private, protected, etc.) to control their visibility and accessibility from other parts of the program.
* They are often used for storing and managing the internal state of an object.

**Example of a variable in a class:**

public class MyClass

{

private int myField; // Variable

}

**Properties:**

* Properties are a higher-level concept that provides controlled access to class fields (variables).
* Properties define how the values of the underlying fields can be read and modified by external code.
* They are used to encapsulate the internal state of an object and provide a level of abstraction and control over data access.
* Properties can have get and set accessors, allowing you to specify custom logic when reading or writing the property's value.
* Properties are often used to expose the state of an object to the outside world while enforcing validation or encapsulation rules.

**Example:**

public class MyClass

{

private int myField; // Variable

public int MyProperty

{

get

{

return myField; // Property

}

set

{

if (value >= 0)

{

myField = value;

}

}

}

}

**Note**: Property is a member of the class that provides an abstraction to set (write) and get (read) the value of a private field.

**Q: Difference between Functions and subroutines?**

**Ans:**

**Functions:**

A function is a block of code that can return a value to the caller. It performs some computation and provides a result.

We need to compute and return a value based on input parameters. Functions are typically called with an expression, and their return value can be used in that expression or assigned to a variable.

**Example**:

using System;

public class Program

{

public static int AddNumbers(int a, int b)

{

return a + b;

}

public static void Main()

{

int result = AddNumbers(5, 3);

Console.WriteLine(result); // Prints 8

}

}

**Subroutines:**

A subroutine (also known as a procedure or method) is a block of code that does not return a value. It is used for performing tasks or operations but doesn't produce a result that can be used in expressions.

Subroutines are used when you want to perform a series of actions or tasks without necessarily producing a value that needs to be returned. They are often used for code organization and reuse.

Subroutines are called as standalone statements or commands. They don't return values, so they are called for their side effects.

**Example:**

Using System;

public class Program

{

public static void Greet(string name)

{

Console.WriteLine ($"Hello, {name}!");

}

public static void Main()

{

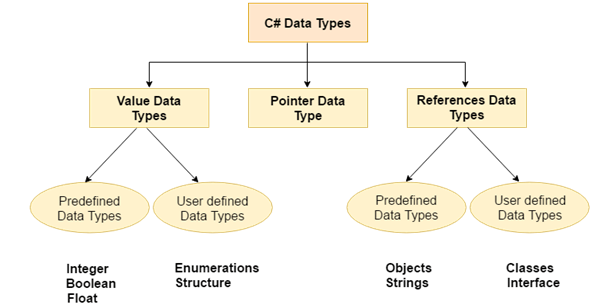
Greet("Alice"); // Prints "Hello, Alice!"

}

}

**Day-2**

**Q: Data Types in C sharp**



| Data Type | Size (in bits) | Range | Example |
| --- | --- | --- | --- |
| byte | 8 bits (1 byte) | 0 to 255 | byte age = 30; |
| sbyte | 8 bits (1 byte) | -128 to 127 | sbyte temperature = -10; |
| short | 16 bits (2 bytes) | -32,768 to 32,767 | short distance = 1000; |
| ushort | 16 bits (2 bytes) | 0 to 65,535 | ushort count = 65535; |
| int | 32 bits (4 bytes) | -2,147,483,648 to 2,147,483,647 | int population = 1000000; |
| uint | 32 bits (4 bytes) | 0 to 4,294,967,295 | uint score = 42; |
| long | 64 bits (8 bytes) | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | long bigNumber = 12345678901234567; |
| ulong | 64 bits (8 bytes) | 0 to 18,446,744,073,709,551,615 | ulong largeNumber = 18446744073709551615; |
| float | 32 bits (4 bytes) | Approximately ±1.5 x 10^-45 to ±3.4 x 10^38 | float height = 1.75f; |
| double | 64 bits (8 bytes) | Approximately ±5.0 x 10^-324 to ±1.7 x 10^308 | double pi = 3.141592653589793; |
| decimal | 128 bits (16 bytes) | Approximately ±1.0 x 10^-28 to ±7.9 x 10^28 | decimal money = 100.50m; |
| char | 16 bits (2 bytes) | Unicode characters (0 to 65,535) | char grade = 'A'; |
| bool | Not defined (implementation-dependent) | true or false | bool isStudent = true; |

## 

## Q: What is Type Casting?

Ans: Type casting is when we assign a value of one data type to another type.

There are two types of casting:

* **Implicit Casting** (automatically) - converting a smaller type to a larger type size  
  char -> int -> long -> float -> double

**Example:**

int integerNumber = 42;

double doubleNumber = integerNumber; // Implicit conversion from int to double

* **Explicit Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char

**Example:**

double doubleNumber = 42.5;

int integerNumber = (int)doubleNumber; // Explicit conversion from double to int

* **REFERENCE DATA TYPES:**

The reference data type in C# does not have the actual data stored in a variable, but they contain a reference to the variables.

* **Object Type**

The Object Type is the ultimate base class for all data types.. The object types can be assigned values of any other types, value types, and reference types, predefined or user-defined types.

**Example**

Object ob;

ob = 250;

* **Dynamic Type**

Store any type of value in the dynamic data type variable. Type checking for these types of variables takes place at run-time.

**Example**

dynamic d = 100;

* **String Type**

The String Type allows you to assign any string values to a variable. The string type is an alias for the System. String class. It is derived from object type.

**Example**

String val = "Welcome!”;

**// CREATE AN OBJECT OF STRING TYPE:**

String S1;

S1=new String (“Abc”);

* **Statement Types** :-

A statement is a basic unit of execution of a program. A program consist of multiple statement.

**Different types of statements are:**

* **Declaration statement**
* **Expression statement**
* **Selection statement**
* **Iteration statement**
* **Jump statement**
* **Operators Types :**

Operators are used to perform operations on variables and values.

* Arithmetic Operator
* Logical Operator
* Bit wise operator
* Assignment operator
* Assignment operator
* **Conditional statements :**

The conditional statements are as follows:

* If—else
* Boolean condition
* Ternary condition
* Switch condition

**Day: 3**

**Q: Code in c sharp that includes at least one bitwise operator, type conversion, data types .**

**Q: +12 and -12 time zone does exist if exist does they meet at same point ?**

**Ans**: Yes ,+12 and -12 time zones do exist, but they are on opposite sides of the Earth, and they do not meet at the same point. Because +12 Time Zone is often referred to as "UTC+12" or "GMT+12." It is 12 hours ahead of Coordinated Universal Time (UTC) and 12 Time Zone: This is often referred to as "UTC-12" or "GMT-12." It is 12 hours behind Coordinated Universal Time (UTC).

Since the +12 and -12 time zones are on opposite sides of the Earth, they are exactly 24 hours apart

* **Loops :**

The loops are as follows:

* For loop
* While loop
* Do-while loop
* For each loop
* **Methods**: Methods are used to perform certain actions, and they are also known as **functions**.

**Syntax:**

**class Abc**

**{**

**public void M()**

**{**

**//Block of code to be executed**

**}**

**Public static void main(String []args)**

**{**

**M();**

}

}

* **Pointers:** A pointer is nothing more than a variable that holds the address in memory of another variable.

**Syntax:**

int num=10;

int \* p;

\*p=& num;

**Unsafe keyword in Pointer :**

 Unsafe code allows us to manipulate memory directly, and are loosing garbage collection and whilst directly accessing memory.

Example:

using System;

class MyClass

{

public unsafe void Method()

{

int x = 10;

int y = 20;

int\* ptr1 = &x;

int\* ptr2 = &y;

Console.WriteLine((int)ptr1);

Console.WriteLine((int)ptr2);

Console.WriteLine(\*ptr1);

Console.WriteLine(\*ptr2);

}

}

class MyClient

{

public static void Main()

{

MyClass mc = new MyClass();

mc.Method();

}

}

**Day: 4**

**Q: Difference between for and for each loop?**

**Ans:**

**For Loop:**

* The **for** loop is a general-purpose loop that allows you to iterate over a range of values using a counter variable.
* It is typically used when you know the exact number of iterations you need or when you want to iterate over an index-based data structure like an array.
* We explicitly specify the initialization, condition, and iteration step of the loop.

Example:

for (int i = 0; i < 5; i++)

{

Console.WriteLine (i);

}

**For Each Loop :**

* The foreach loop is used to iterate over the elements of the collection.
* The **for each** loop is specifically designed for iterating over collections that implement the IEnumerable or IEnumerable<T> interface, such as arrays, lists, dictionaries, and other enumerable data structures.
* It simplifies the iteration process by automatically managing the iteration variable and looping until the end of the collection.
* We do not need to specify an index or manage the loop control variable manually.

Example:

int [] numbers = { 1, 2, 3, 4, 5 };

foreach (int number in numbers)

{

Console.WriteLine (number);

}

**Q: How many argument can pass in main method?**

**Ans:** We can pass multiple argument in main method in command line argument.

The Main method can accept command-line arguments as an array of strings.

**Q: Difference between Reference and Out?**

Ans:

**Ref Keyword :**

* ref keyword is used when a called method has to update the passed parameter.
* ref keyword is used to pass data in bi-directional way,
* Before passing a variable as ref, it is required to be initialized otherwise compiler will throw error.
* In called method, it is not required to initialize the parameter passed as ref.

**Out Keyword :**

* out keyword is used when a called method has to update multiple parameter passed.
* out keyword is used to get data in uni-directional way.
* No need to initialize variable if out keyword is used.
* In called method, it is required to initialize the parameter passed as out.

**Q: How many types of arguments can be pass?**

**Ans:** In C#, arguments can be passed to parameters either by value or by reference. C# types can be either reference types (class) or value types (struct).

Pass by value means **passing a copy of the variable** to the method.

Pass by reference means **passing access to the variable** to the method.

**Q: Difference between class and structure?**

**Ans:**

**CLASS:**

* Classes are of reference types
* All the reference types are allocated on heap memory.
* Allocation of large reference type is cheaper than allocation of large value type
* Class is generally used in large programs.
* Classes can contain constructor or destructor.
* Classes used new keyword for creating instances.
* Two variable of class can contain the reference of the same object and any operation on one variable can affect another variable.

**STRUCTURE :**

* Structs are of value types.
* All the value types are allocated on stack memory.
* Allocation and de-allocation is cheaper in value type as compare to reference type.
* Struct has limited features.
* Struct are used in small programs.
* Structure does not contain parameter less constructor or destructor, but can contain Parameterized constructor or static constructor.
* Each variable in struct contains its own copy of data(except in ref and out parameter variable) and any operation on one variable can not effect another variable.

# **Q: How to Return Multiple Values from a Function ?**

**Ans:** We can return multiple values from a function using the following 3 approaches:

* Reference parameters
* Output parameters
* Returning an Array

**Day: 5**

**Q: Make three project which consist of one .exe, two .dll files and the required information the files contains are:**

* Scientific Calculator (.exe file)
* ArithemticOperations (.Dll file)Operations are : +,-,\*,/,%
* Power Operations (.Dll file ) Operations are : square, squareroot,cube,cuberoot,x power y,x power 1/y .

Requirements:

1.Use template keyword for data types(int,float,decimal).