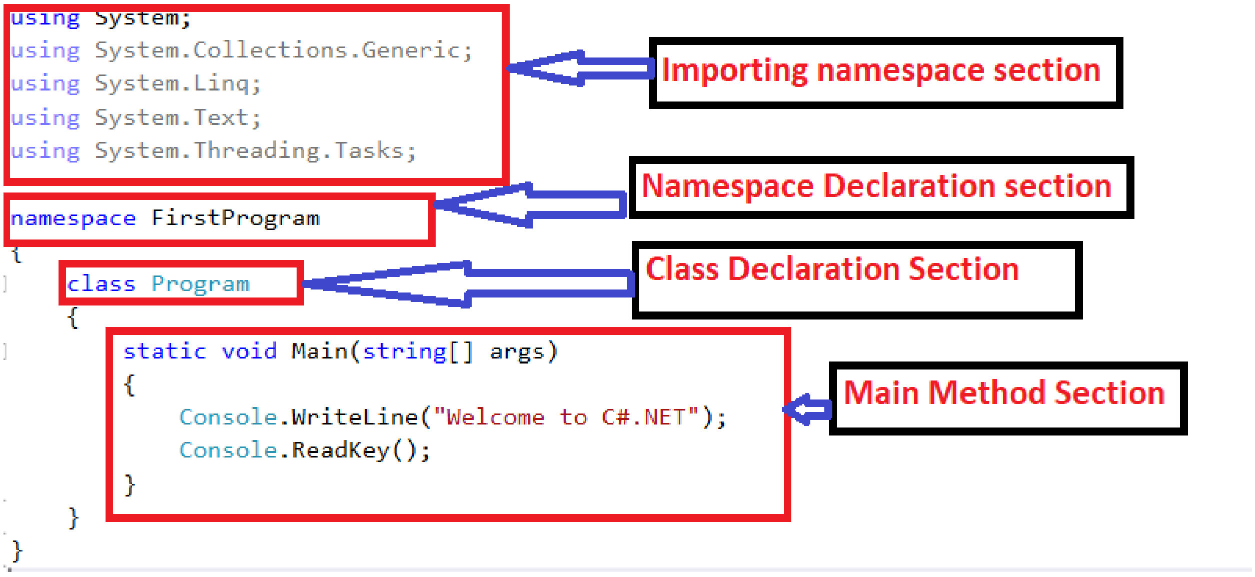


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstProgram
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to C#.NET");
            Console.ReadKey();
        }
    }
}
```

Importing namespace section
Namespace Declaration section
Class Declaration Section
Main Method Section

Importing section: This section contains importing statements that are used to import the BCL (Base Class Libraries). This is similar to the include statements in the C programming language.

Syntax: using namespace; Example: using System;

Note: If the required namespace is a member of another namespace we have to specify the parent and

child namespaces separated by a dot.

using System.Data;

using System.IO;

<u>**Namespace Declaration**</u>: Here a user-defined namespace is to be declared. In .NET applications, all classes related to the project should be declared inside one namespace.

Syntax: namespace NamespaceName { }

Generally, the namespace name will be the same as the project name.

Class Declaration: This is to declare the start-up class of the project. In every .NET application like console and windows applications, there should be a start-up class. A startup class is nothing but a class that contains a "Main()" method.

Note- C# code is compiled into Intermediate Language (IL). Code moves from preprocessor to compiler, to assembler to Linker.

Intermediate Language Code is a set of instructions that produces machine code for execution on a machine processor. It is partially compiled code.

JIT compiles IL code to machine language.

Managed Code- is code that executed by CLR (means every application code is totally dependent on .NET platform. Benefits: Garbage Collections, Exception Handling.

Unmanaged Code: Code executed by Runtime program that is not part of .NET platform. Unmanaged code cleaned using destructor/finalize.

CLR- Common Language Runtime. Handles program execution for various languages. Architecture of CLR handles memory management, garbage collection, security handling.

Garbage Collector- cleans the unused managed objects. To deallocate memory occupied by unused managed objects. GC internally uses destruction method to clean up the unused objects.

Private Finalize method- just before garbage collection, private finalize method is called. This method cleans unmanaged code, which is automatically triggered whenever an instance is not recalled.

Public Dispose method- is called explicitly by the user. This method is defined in IDisposable interface.

Main() method:

The main() method is the starting execution point of the application. When the application is executed.
the main method will be executed first. This method contains the main logic of the application.

Using is a keyword. Using this keyword we can refer to .NET BCL in C# applications.

Console class is available in the “System” namespace. This Console class provides some methods and properties using which we can implement the user interface in a console application.
All the properties and methods available in the console class are static. So we can access these members by using the Console class name i.e. we don’t require a Console class instance.

Methods of Console class:

Method	Description
Clear()	To clear the screen
Beep()	Play a beep sound using a PC speaker at runtime
Resetcolor()	Reset the background and foreground color to its default state
Write(“string”)	Display the specified message on the console window
WriteLine(“string”)	Same as the write method but automatically moves the cursor to the next line after printing the message.
Write(variable)	Displays the value of the given variable
WriteLine(variable)	Displays the value of the given variable along with moving the cursor to the next line after printing the value of the variable.
Read()	**Read a single character from the keyboard and returns its ASCII value**. The Datatype should be int as it returns the ASCII value.
ReadLine()	**Reads a string value from the keyboard and returns the entered value only.** As it returns the entered string value so the DataType is going to be a string.
ReadKey()	**This method reads a single character from the keyboard and returns that character**. The Datatype should be int as it returns the ASCII value. It is a STRUCT Data type which is ConsoleKeyInfo.

The ReadLine method always accepts the value in the form of a string. So we need to convert the values to the appropriate type. **WriteLine and Write are static methods of the Console class.**

Datatypes are basically used to store the data temporarily in the computer through a program.

In the real world, we have different types of data like integers, floating-point, characters, strings, etc. Datatypes are something that gives information about

1. Size of the memory location.
2. The range of data that can be stored inside that memory location
3. Possible legal operations that can be performed on that memory location.

There are 3 types of data types (PVR)

1. Value Data Type
2. Reference Data Type
3. Pointer Data Type

Value Data Type- data type which stores the value directly within its own memory is called the Value Data Type. They are derived from the class System.ValueType. The examples are int, char, and float which store numbers, alphabets, and floating-point numbers respectively. Value types hold a data value inside its memory space.

1. Predefined Data Types – Example includes Integer, Boolean, Float, etc.
2. User-defined Data Types – Example includes Structure, Enumerations, etc

Reference Data Type- is used to store the reference of a variable. In other words, we can say that the reference types do not store the actual data stored in a variable, rather they store the reference to the variables. These are stored in the HEAP. Reference type keep objects address where value is stored.

1. Predefined Types – Examples include Objects, String, and dynamic.
2. User-defined Types – Examples include Classes, Interface.

If you define a user-defined data type by using the struct keyword, it is value type.

If you define a user-defined data type by using the class keyword, it is reference type

Yes, Value types are sealed.

Note-

- Default value of reference type variable is null, when they are not initialized.
- Out, Ref keywords used to pass value type as reference type.
- **Ref Keywords** pass arguments by reference and not value. To use ‘ref’ keyword, need to explicitly menton ‘ref’.

Void method (ref int refArg){ refArg = refArg+10;}

int no=1; Method(ref no); Console.Write(no);   o/p-11

- **Out Keywords** pass arguments within methods and functions.

  Pointer Type- pointer in C# language is a variable, it is also known as a locator or indicator that points to an address of the value which means pointer type variables stores the memory address of another type.

Built-in Data Types-

1. Boolean type – Only true or false
2. Integral Types – sbyte, byte, short, ushort, int, uint, long, ulong, char
3. Floating Types – float and double
4. Decimal Types
5. String Type

Literals or Constants- Literals in C# are the fixed values and these fixed values are used by Variables and the values cannot be modified during the execution of the program.

1- Integer Literals   2- String Literals   3- Character Literals

4- Floating-point Literals   5- Boolean Literals

Integer Literals

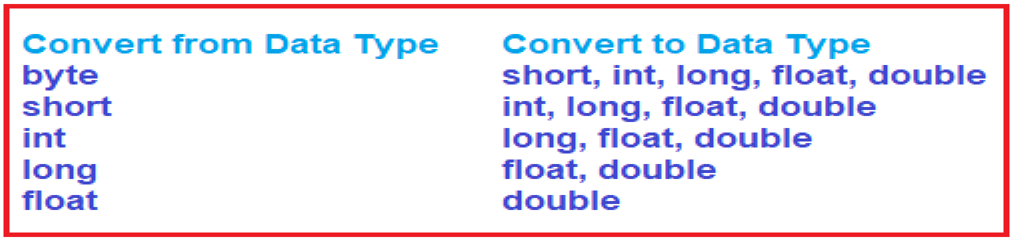
```
101; // decimal literal                 0146; // octal literal

0x123Face; // Hexa-decimal literal      0b1111; // Binary literal
```

Type Casting- we can say that Type Casting or Type Conversion is the process to change one data type into another data type.

Implicit Conversion or Implicit Type Casting is automatically done by the compiler and in this case, there will be no data loss. type conversion is done from a smaller data type to a larger data type and from derived classes to the base class. This type of type conversion is safe.
Generally, in the case of implicit Type Conversion, the smaller data types like int (having less memory size) are automatically converted to larger data types like double (having larger memory size).

Implicit Type Casting happens when:

1. The two data types are compatible.
2. When we assign a value of a smaller data type to a bigger data type.



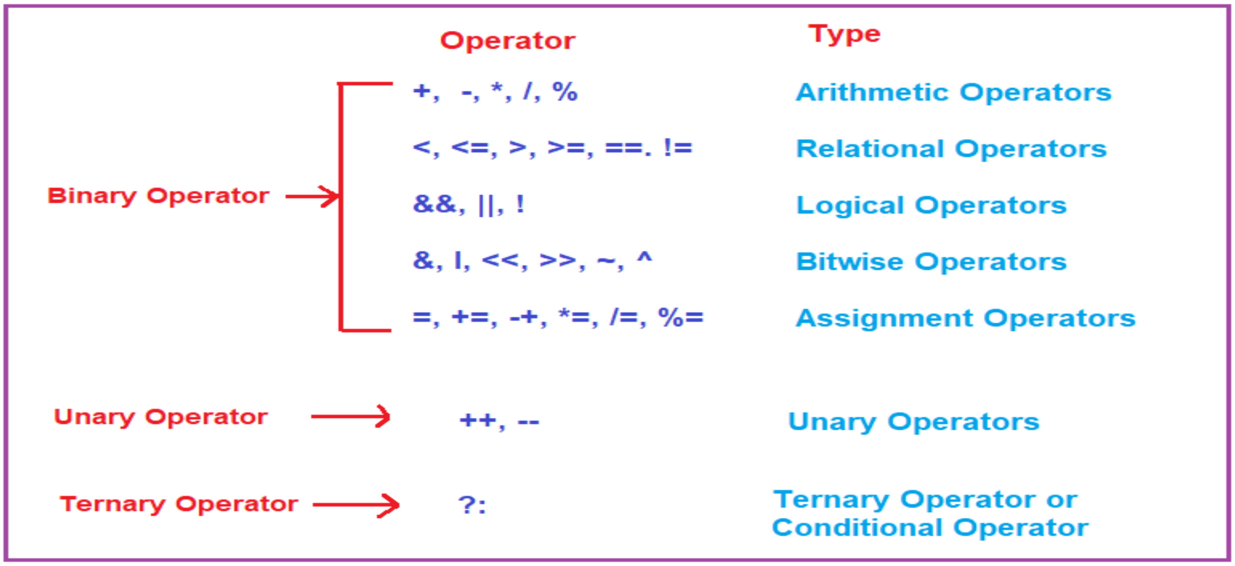
Convert from Data Type	Convert to Data Type
byte	short, int, long, float, double
short	int, long, float, double
int	long, float, double
long	float, double
float	double

Explicit Conversion or Explicit Type Casting in C# is done by using the Cast operator. It includes conversion of larger data types to smaller data types and conversion of base classes to derived classes. In the case of Explicit Conversion or Explicit Type Casting there is a chance that data might be lost or conversion might not be successful for some reason. This is an unsafe type of conversion.

Boxing- converts value type (int, char) to reference type (objects) which is an implicit conversion process using object value.
EXample- int num=23; object obj= num;

Unboxing- converts reference type (object) to value type (int, char) using explicit conversion process.
Example- int i = (int)obj;

Operators- Operators are used to perform operations on operands. Operators are used to manipulate the variables and values in a program. (BLAAR)



	Operator	Type
Binary Operator	+, -, *, /, %	Arithmetic Operators
	<, <=, >, >=, ==. !=	Relational Operators
	&&, ||, !	Logical Operators
	&, I, <<, >>, ~, ^	Bitwise Operators
	=, +=, -+, *=, /=, %=	Assignment Operators
Unary Operator	++, --	Unary Operators
Ternary Operator	?:	Ternary Operator or Conditional Operator

Variable- A variable is a name given to a storage area that is used to store values of various data types. Each variable needs to have a specific type, which determines the size and layout of the variable's memory.

A name that is given for any computer memory location is called a variable. The purpose of the variable is to store some data. The **user will access it by the variable name and the compiler will access it by the address**

Rules for variable declaration in C#:

1. A variable name must begin with a letter or underscore.
2. Variables in C# are case sensitive
3. They can be constructed with digits and letters.
4. No special symbols are allowed other than underscores.
5. sum, height, _value are some examples of the variable name

```
data_type variable_name = value;
```

Types of Variables:

1-Local Variable 2-Instance Variable(non-static) 3-Static Variable

4-Const Variable 5-Readonly Variable

Local Variables are declared inside the method of a class. The scope of the local variable is limited to the method, which means you cannot change the value of the local variable outside the method and you cannot even access it outside the method. The initialization of the local variable is mandatory.

1. Scope of the Local Variables: Within the block in which it is declared.
2. The lifetime of the Local Variable: Until the control leaves the block in which it is declared.

Instance variables are also known as non-static variables. It is always declared in a class but outside the method, block, or constructor. The instance variable is created when an object of the class is created and destroyed when the object is destroyed. Instance Variables can only be accessed by creating the objects of the class. The initialization of the instance variable is not required, it takes the default value based on the data type of the variable.

1. Scope of Instance Variable: Throughout the class except in static methods.
2. The lifetime of Instance Variable: Until the object is available in the memory.

Static variables in C# are also known as the Class Variable. The Static variables are created automatically at the start of the program execution and also destroyed automatically when program execution ends. These variables are declared similarly to instance variables, the difference is that static variables are declared using the static keyword within a class outside any method, constructor or block. Initialization of Static Variable is not mandatory, it takes the default value based on the variable data type. To access static variables, we need not create an object of that class, we can simply access the variable as class_name.variable_name;

1. Scope of the Static Variable: Throughout the class.
2. The Lifetime of Static Variable: Until the end of the program.

Instance variables vs Static variables- ***1-***In the case of Instance Variable, each object will have its own copy whereas We can only have one copy of a static variable irrespective of how many objects we create.

2- We can access the instance variables through object references whereas the Static Variables can be accessed directly by using the class name in C#.

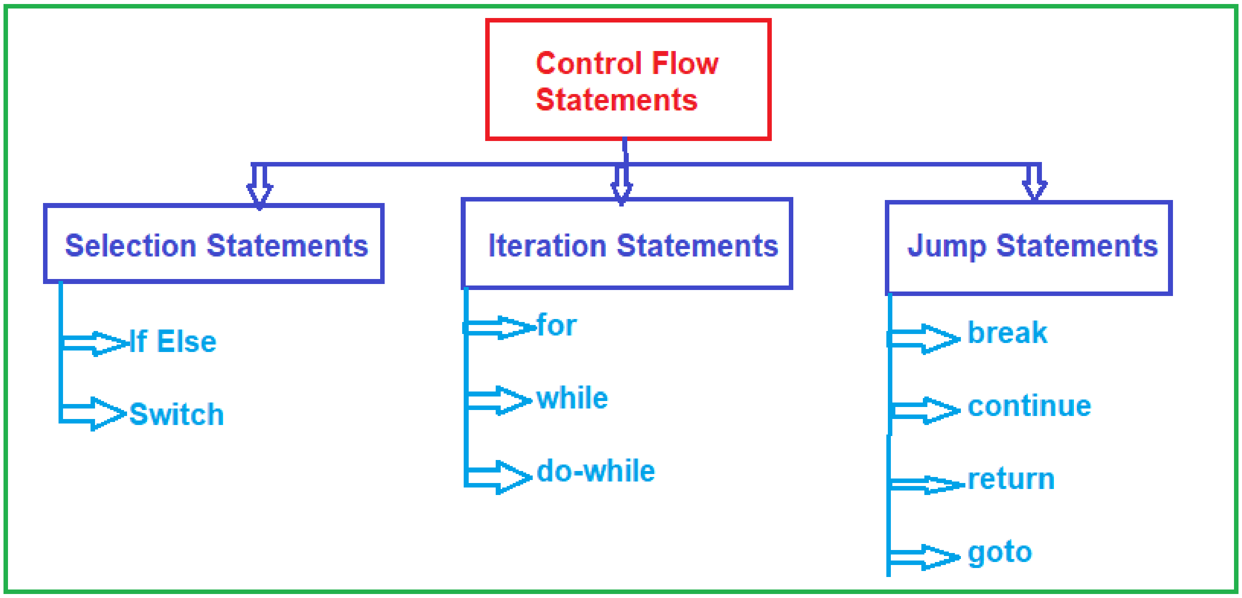
3- In the life cycle of a class, a static variable is initialized only once, whereas instance variables are initialized for 0 times if no instance is created and n times if n number of instances are created

Constants Variables- if we declare a variable by using the const keyword, then it is a constant variable and the value of the constant variable can't be modified once after its declaration. So, it is mandatory to initialize at the time of declaration only.

Read-Only Variables: When we declare a variable by using the readonly keyword, then it is known as a read-only variable and these variables can't be modified like constants but after initialization. That means it is not mandatory to initialize a read-only variable at the time of its declaration, they can also be initialized under the constructor. That means we can modify the read-only variable value only within a constructor.

The behavior of read-only variables will be similar to the behavior of non-static variables, i.e. initialized only after creating the instance of the class and once for each instance of the class created.

Control Flow Statements in C# are the statements that alter the flow of program execution and provide better control to the programmer on the flow of execution. The Control Flow Statements are useful to write better and more complex programs. A program executes from top to bottom except when we use control statements, we can control the order of execution of the program, based on logic and values.
Generally, the statements inside our C# programs are executed from top to bottom, in the order that they appear. The Control flow statements change or break the flow of execution by implementing decision making, looping, and branching our program to execute particular blocks of code based on the conditions**.**



Jump Statements are used to transfer control from one point to another point in the program due to some specified condition while executing the program. The Jump Statements are used to modify the behavior of conditional and iterative(while, for, do-while) statements. The Jump Statements in C# allow us to exit a loop, and start the next iteration, or explicitly transfer the program control to a specified location in your program. C# supports the following four jump statements:

1. **Break- used to skip the next statement of current iteration and come out of loop.**
2. **Continue- skip statement execution from the loop body.**
3. goto
4. return (In the Function section we will discuss the return statement)
5. throw (In the Exception Handling section we will discuss the throw statement)

Continue is a keyword. By using the continue keyword, we can skip the statement execution from the loop body.

Goto Statement in C# is used to transfer the control to the labeled statement in the program. The label is a valid identifier and placed just before the statement from where the control is transferred. That means the goto Statement provides an unconditional jump from the goto to a labeled statement in the same function.

Goto is a keyword and by using this goto keyword we can pass the control anywhere in the program in the local scope. It makes testing and debugging difficult.

function is a group of related instructions that performs a specific task. It can be a small task or a big task but the function will perform that task completely. Functions take some input as parameters and return the result as a return value.

Functions are useful for procedural programming or modular programming. If we write a function then we can reuse the function multiple times in the program.

In the C#, a function is a section of the program that contains a set of instructions or code. **A method is a set of statements that is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name**. Think of a method as a subprogram that acts on data and often returns a value. Each method has its own name. Functions allow us to reuse the code without retyping the code

1. Reusability: By using functions we can create re-usability blocks i.e. develop once and use multiple times.
2. Code Maintenance: When we are developing the application by using functions, then it is easy to maintain code for future enhancement.
3. Code Sharing: A function may be used by many other programs.
4. Flexible Debugging: It is easy to locate and isolate a faulty function for further investigations.
5. Data Protection: Functions can be used to protect data and local data. Local data is available only within a function when the function is being executed.
6. Code Reduced: Reduces the size of the code, duplicate statements are replaced by function calls.

- **Will the functions occupy space in memory?**

  Yes, the machine code of a function is kept code section.

- **Will a function occupy space even if it is not called?**

  Yes, if a function is defined in a program or included in the library, it will occupy space in the code section.

- **Where the memory for the variable of a function is created?**

  Memory for the variables used in a function is created in the stack.

- **When the memory for variables will be allocated?**

  Memory for the variables will be allocated at runtime when the function is called and deleted when the function ends.

- **Is the memory for variables allocated freshly for each call?**

  For “for each” call of a function memory for the variables is created freshly in the stack.

- **What is the return type of a function?**

When a function is called by passing parameters, it will compute and get the results. A function can return the result to a calling function. The **return type is the data type of a value returned by the function.**

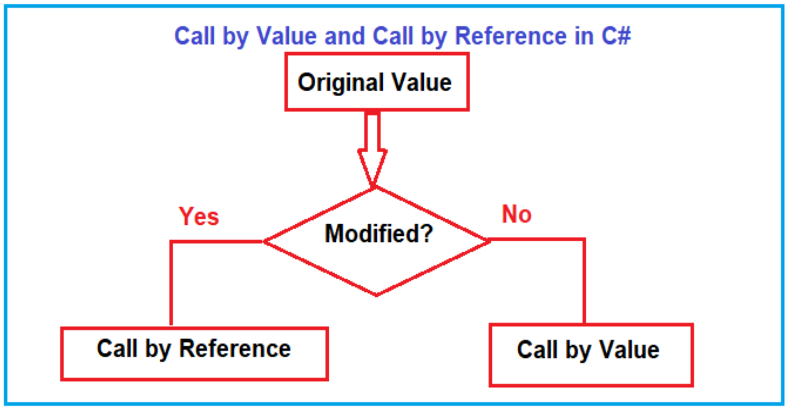
- **What is void?**

  If a function is not returning any value, then its return type is mentioned as void.

- **Difference between int main() and void main()**
  1. void main() means the main function is not returning any value.
  2. int main() means main function will return 0; 0 is a success code.
  3. The function has terminated successfully. The main() will return the value to the operating system, like windows.

https://dotnettutorials.net/lesson/user-defined-functions-in-csharp/

Functions can be invoked in two ways: **Call by Value** or **Call by Reference**. The parameters passed to the function are called ***actual parameters*** whereas the parameters received by the function are called ***formal parameters.***



Call By Value- In call by value method of parameter passing, the values of actual parameters are copied to the function's formal parameters.

There are two copies of parameters stored in different memory locations. One is the original copy and the other is the function copy. Any changes made inside functions are not reflected in the actual parameters of the caller. The memory location referred to by formal parameters and actual arguments is different. It doesn't require a ref or out keyword

Call By Reference- In call by reference method of parameter passing, the address of the actual parameters is passed to the function as the formal parameters. Both the actual and formal parameters refer to the same locations. Any changes made inside the function are actually reflected in the actual parameters of the caller. It requires a ref or out keyword to achieve a call by reference.

Function calling itself is called ***Recursion***. Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.

What are the advantages of Recursion in C# Language?

1. Function calling-related information will be maintained by recursion.
2. Stack evaluation will take place by using recursion.
3. Prefix, postfix, infix notation will be evaluated by using recursion

What are the disadvantages of Recursion in C# Language?

1. It is a very slow process due to stack overlapping.
2. The recursive program can create stack overflow.
3. The recursive program can create infinite loops.

Command Line Arguments

Main() method is the starting point from where the program execution starts. The most important point that you need to remember is that the main method doesn't accept any parameter from any method. **It only accepts parameters through the Command-Line.** If you notice the Main method signature, it has a string array type parameter that can accept n number of parameters at runtime. In Main(string[] args), args is a string type of array that can hold numerous parameters.

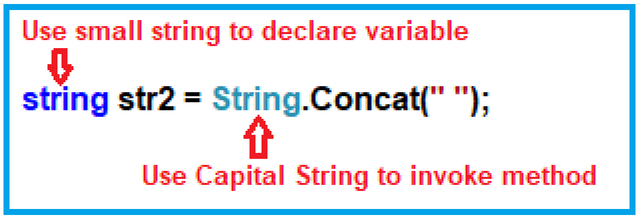
Strings- String are reference types. String are objects. strings are immutable.

CLR stores strings as an array of characters. So if we delete or add characters to strings, the original arrays of characters are fixed, hence a new array of characters is created to accommodate change. This is known as immutability of strings.

Differences between String(Capital) vs string(small)

small string is actually an alias of String (Capital string). If you right-click on the small string and if you go to the definition then you will see that the actual class name is a capital string.

as per the naming convention when you are creating a variable use the small string (i.e. string) and whenever you want to invoke methods on the string then use the capital string (i.e. String)

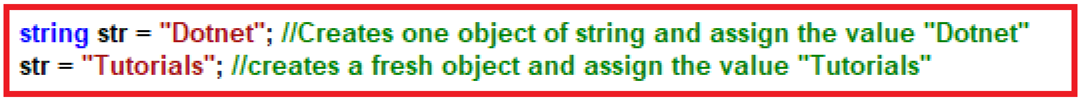


Use small string to declare variable

```
string str2 = String.Concat(" ");
```

Use Capital String to invoke method

Mutable means can be changed whereas Immutable means can not be changed
Please have a look at the below image. When the first statement is executed, it will create one object and assign the value DotNet. But when the second statement is executed, it will not override the first object, it lets the first object be there for garbage collection and creates a fresh object, and assign the value Tutorials.



```
string str = "Dotnet"; //Creates one object of string and assign the value "Dotnet"
str = "Tutorials"; //creates a fresh object and assign the value "Tutorials"
```

when the above two statements are executed, internally two memory locations are created. One with the value DotNet and the current one with the value Tutorials and the current one is going to be referred to in the program. So, each time, we assign a new value to the string variable, a new object is created and this is the reason why strings are immutable in C#. Strings Immutable for Thread Safety

String Intern is a process that uses the same memory location if the value is the same. Improves performance.

String Builder -- pending

static keyword is used in Factory Design Pattern, Singleton Design Pattern as well as used for data sharing. static class is also going to be used to share the common data.

Static is a modifier. Static can be method, variable, constructor and class.

Static variables can be initialized only once during the lifecycle of class.

If we declare a method using the static modifier then it is called a static method. We can directly consume the static members within a non-static method without any restriction.

‘This’ Keyword can not be used within a static method because static, variables/methods can be used in a static method. Static field of a non-static class is shared across all instances.

If we create the constructor explicitly by the static modifier, then we call it a static constructor. Static constructor is the fast block of code that gets executed under a class.

Class which is created by using the static modifier is called a static class. A static class can contain only static members. It is not possible to create an instance of a static class. Static class is a sealed class. Static class can't be inherited.

Static Variables- variable declared by using the static modifier or variable declared inside of any static block then those variables are considered as static variables whereas the rest of the others are considered as non-static variables. If you want a variable to have the same value throughout all instances of a class then you need to declare that variable as a static variable. So, the static variables are going to hold the application level data which is going to be the same for all the objects.

static variable gets initialized immediately once the execution of the class starts whereas the non-static variables are initialized only after creating the object of the class
A static variable gets initialized only once during the life cycle of a class whereas a non-static variable gets initialized either 0 or n number of times, depending on the number of objects created for that class.
If you want to access the static members of a class, then you need to access them using the class name whereas you need an instance of a class to access the non-static members.

Non Static variables are created when the object is created and are destroyed when the object is destroyed. The object is destroyed when its reference variable is destroyed or initialized with null. So we can say that the scope of the object is the scope of its referenced variables.

Static and Non-Static Methods in C# If we declare a method using the static modifier then it is called a static method else it is a non-static method. You cannot consume the non-static members directly within a static method. If you want to consume any non-static members with a static method then you need to create an object and then through the object, you can access the non-static members. On the other hand, you can directly consume the static members within a non-static method without any restriction.

Rules to follow while working with static and non-static members in c#:

1. Non-static to static: Can be consumed only by using the object of that class.
2. Static to static: Can be consumed directly or by using the class name.
3. Static to non-static: Can be consumed directly or by using the class name.
4. Non-static to non-static: Can be consumed directly or by using the “this” keyword.

Understanding Static and Non-Static Constructor in C#:

If we create the constructor explicitly by the static modifier, then we call it a static constructor and the rest of the others are the non-static constructors.

The most important point that you need to remember is the **static constructor is the fast block of code that gets executed under a class.** No matter how many numbers of objects you created for the class the static constructor is executed only once. On the other hand, a non-static constructor gets executed only when we created the object of the class and that is too for each and every object of the class.

It is not possible to create a static constructor with parameters. This is because the static constructor is the first block of code that is going to execute under a class. And this static constructor is called implicitly, even if parameterized there is no chance of sending the parameter values

Static class- The class which is created by using the static modifier is called a static class in C#. A static class can contain only static members. It is not possible to create an instance of a static class. This is because it contains only static members. And we know we can access the static members of a class by using the class name.

Constants are the immutable values that are known at the time of program compilation and do not change their values for the lifetime of the program. Constants are static by default.
The behavior of a constant variable is the same as the behavior of a static variable i.e. maintains only one copy in the life cycle of class and initialize immediately once the execution of the class start (object not required)

Read-only variables are also immutable values but these values are known at runtime and also do not change their values for the life of the program. variable which is created by using the **readonly** keyword is known as a read-only variable in C#. The read-only variable's value cannot be modified once after its initialization.
behavior of a read-only variable is similar to the behavior of a non-static variable. That is, it maintains a separate copy for each object. The only difference between these two is that the value of the non-static variable can be modified from outside the constructor while the value of the read-only variable cannot be modified from outside the constructor body.

Properties- Properties are used as a mechanism to set and get the values of a class outside of that class. It is used to encapsulate and protect the data members (i.e. fields).

A property in C# is a member of a class which is used to set and get the data from a data field of a class. A property is never used to store data, it just acts as an interface to transfer the data. We use the Properties as they are the public data members of a class, but they are actually special methods called accessors.

If a class contains any value in it and if we want to access those values outside of that class, then you can provide access to those values in 2 different ways

1. By storing the value under a public variable we can give access to the value outside of the class.
2. By storing that value in a private variable we can also give access to that value outside of the class by defining a property for that variable,

Assessors are nothing but special methods which are used to set and get the values from the underlying data member. Assessors are of two types such as

1- set accessor   2-get accessor

set accessor is used to set the data (i.e. value) into a data field. This set accessor contains a fixed variable named "value". Whenever we call the property to set the data, whatever data (value) we are supplying that will come and store in the variable "value" by default.

Syntax: `set { Data Field Name = value; }`

get accessor is used to get the data from the data field. Using this get accessor you cannot set the data.

Syntax: get { return Data Field Name; }

What are the different types of properties supported by C#.NET?

1. Read-only property
2. Write only property
3. Read Write property
4. Auto-implemented property

Read-only Property is used to read the data from the data field. Using this property you cannot set the data into the data field. This property will contain only one accessor i.e. “get” accessor.

Syntax: AccessModifier Datatype PropertyName { get { return DataFieldName; } }

Write only Property is used to write the data into the data field of a class. Using this property you cannot read the data from the data field. This property will contain only one accessor i.e. set accessor.

Syntax: AccessModifier Datatype PropertyName { set { DataFieldName = value; } }

Read Write Property is used for both reading the data from the data field as well as writing the data into the data field. This property will contain two accessors i.e. set and get.

What are the advantages of using Properties in C#?

1. Properties will provide the abstraction to the data fields.
2. They also provide security to the data fields.
3. Properties can also validate the data before storing into the data fields.

Equality Operator(==)- compares by reference. It is a reference type which means that if equality operator is used. It will return true only if both references point to the same object.

Equals method- is used to compare values carried by object.

Convert.ToString and ToString Method in C# Both these methods are used to convert a value to a string. The difference is Convert.ToString() method handles null whereas the ToString() doesn’t handle null.

checked keyword in C# is used to explicitly enable overflow checking for integral-type arithmetic operations and conversions.

unchecked keyword is used to suppress overflow-checking for integral-type arithmetic operations and conversions.

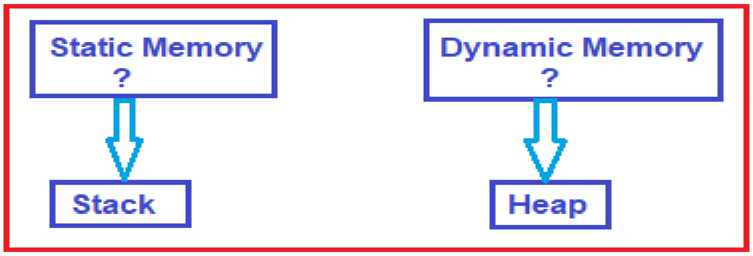
overflow checking means when the value of any integral-type exceeds its range, it does not raises any exception, instead it will give us unexpected or garbage results.

we declare a variable in a .NET application, it allocates some memory in the RAM. The memory which it allocates in RAM has three things are as follows:

1. **Name of the variable,**
2. **The data type of the variable, and**
3. **Value of the variable.**

Stack memory is responsible for keeping track of the running memory needed in your application. Variable,data, data location are at same place. Stack stores value type.

Heap memory location does not track running memory. Heap is used for dynamic memory allocation. heap needs to be de-allocated by the garbage collector. Heap stores reference type.



Comments-

XML Comments- ///

Single Line Comments- //

Multi Line Comments- /*..... */