Q1) What is C# ?

Ans:

* C# is a strongly typed object-oriented programming language.
* It is modern, object-oriented, and type-safe programming language.
* C# is developed and launched by Microsoft in year 2001.
* Microsoft's Anders Hejlsberg(a Danish software engineer) created C# language.
* C# ranked #4 on the PYPL(Popularity of Programming Language Index) in February 2019.
* We can create a number of different programs and applications using C# like mobile apps, desktop apps, cloud-based services, websites, enterprise software and games.

C# is a widely used professional language –

It is a modern, general-purpose programming language.
It is object & component oriented.
It is a structured language.
It produces efficient programs.
It can be compiled on a wide range of computer platforms.
It is a part of .Net Framework.

Q2) What is .NET and .NET Framework ?

Ans:

.NET is a developer platform made up of tools, programming languages and libraries for building different types of applications.

.NET framework is a complete development and execution environment.

Following are the components of the .Net framework –

1) Common Language Runtime (CLR).
2) .Net Framework Class Library(FCL).
3) Common Language Specification(CLS).
4) Common Type System(CTS).
5) Metadata and Assemblies.
.....ETC

.Net Framework run, and deploy the following application.

1) Console application.
2) Web application
3) Web service
4) WCF service
5) Windows application
6) Windows service
7) WPF, WWF etc.....

Q3) What is Common Language Runtime (CLR) ?

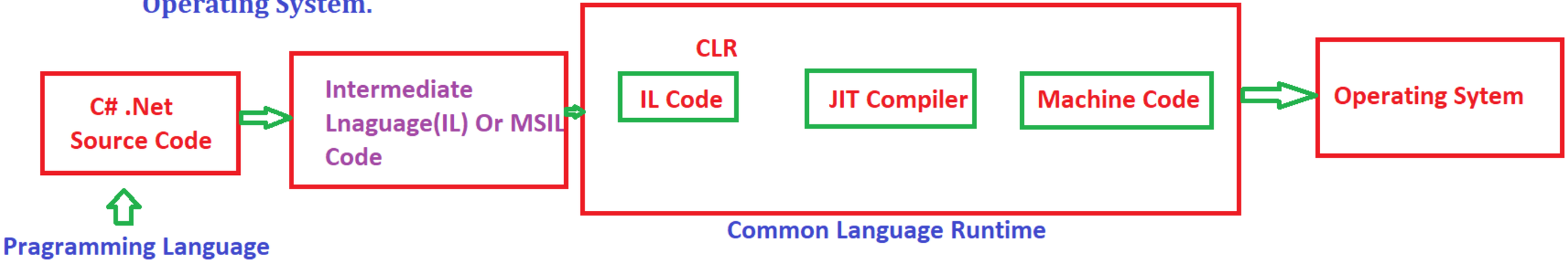
Ans:

CLRis the run-time environment of .NET Framework that runs the codes and provides various services.

CLR takes the responsibility to execute your Microsoft Intermediate language (MSIL) or Intermediate language (IL) Code.

CLR takes the IL (Intermediate Language) code and hand over to JIT (Just-in-Time) Compiler.

JIT compiler reads each and every line of IL code and converts it to machine-language (i.e in binary format) that is executed by the Operating System.



Q4) What are Functions, Components & Benefits of .NET CLR ?

Functions of CLR

1) Convert code into CLI
2) Provides better Security
3) Provides Type safety
4) Memory management
6) Provides Exception handling
7) Improved performance
8) Platform independency
9) Language independency

Components of CLR

1) Class Loader
2) Just In Time (JIT) compiler converts MSIL code into native code.
3) Code Manager manages the code at run time.
4) Garbage Collector collects all unused objects and deallocate them.
5) Thread Support - It supports multithreading of our application.
6) Exception Handler handles exceptions at run time.

Benefits of CLR

1) Improved Performance .
2) Easy to use components developed in other languages.
3) Garbage collection support.
4) Support for exception handling.
5) Support for custom attributes.

etc.....

Q5) What is Common Type System(CTS) in .NET Framework ?

Ans:

.NET Framework supports many languages i.e C#, VB.NET, F# etc and each & every languages has its own data type. But One programming language data type cannot be understood by other languages.

So there can be condition where we need to communicate between two different programming languages.

Common Type System (CTS) ensures that data types defined in two different languages get compiled to a common data type.

Q6) What is Managed and Unmanaged Code in C# ?

Ans:

Whenever we create any application in .NET Framework using visual studio then these applications are run completely under the control of CLR.

Managed Code:

1) The codes which run under the complete control of CLR are called Managed Code
2) It improves the security of the application.
3) It calls the garbage collection automatically when required.
4) It provides dynamic type checking.
5) Disadvantage: You are not allowed to allocate memory directly.

Unmanaged code:

1) The code which do not run under the control of CLR is called unmanaged code.
2) PowerPoint,Skype,Microsoft Excel does not require dot net runtime which is an example of unmannged code.
3) CLR will not provide any facilities and features to the unmanaged code.
4) It does not provide security to the application.
5) Error and exceptions are purely handled by the developer.

7. What is the difference between const and readonly in c# ?

Const

1. Const is a compile time constant.

2. Constant fields are created using const keyword.

3. Const values will evaluate at compile time only.

4. Const variables need to initialize while declaration

```
//Const variables need to initialize while declaration
        const int ID = 1294;
// Reassigning a const variable is not allowed.
ID = 15685;   // Will result compile time error.
```

5. Const value can't be changed and it will be same at all the time.

ReadOnly

1. ReadOnly is a runtime constant

2. ReadOnly fields can be created using readonly keyword

3. ReadOnly values will evaluate at runtime only.

4. You can initilize readonly varabiles while declaration or in constructor

```
readonly  string Address = "f-192";
    //readonly fields can be initlized in constructor
    public ReadOnlyDemo (string address)
    {
        Address = address;
    }
```

5. ReadOnly value can be changed.

8. Is "for" loop faster than "foreach" loop in c# ?

Ans:

1. for loops on List are 2 times cheaper than foreach loops on List.
2. looping on array using for is 5 times cheaper than looping on List using foreach loop.
3. for loop is faster than foreach loop if the array is accessed once per iteration

```
public static void Main(String[] args)
{
    var watch = new System.Diagnostics.Stopwatch();
    watch.Start();
    int[] arr = new int[1000];
    for (int i = 0; i < arr.Length; i++)
    {
        arr[i] = i;
    }

    for (int i = 0; i < arr.Length; i++)
    {
        Console.WriteLine(arr[i]);
    }
    watch.Stop();
    Console.WriteLine($"Execution Time: {watch.ElapsedMilliseconds} ms");
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
996
997
998
999
Execution Time: 804 ms
```

```
public static void Main(String[] args)
{
    var watch = new System.Diagnostics.Stopwatch();
    watch.Start();
    int[] arr = new int[1000];
    for (int i = 0; i < arr.Length; i++)
    {
        arr[i] = i;
    }
    foreach (var item in arr)
    {
        Console.WriteLine(item);
    }
    watch.Stop();
    Console.WriteLine($"Execution Time: {watch.ElapsedMilliseconds} ms");
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
996
997
998
999
Execution Time: 943 ms
```

Q9. How to Convert int to enum in C# ?

We can explicitly type cast an int to a particular enum type, as shown right.

We can also convert using Enum.ToObject() Method as shown below.

```
public enum DaysOfWeek
{
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}
0 references
public class program
{
    0 references
    public static void Main(String[] args)
    {
        int i = 1, j = 6, k = 4;
        DaysOfWeek day1, day2, day3;

        day1 = (DaysOfWeek)i; //Tuesday
        day2 = (DaysOfWeek)j; //Sunday
        day3 = (DaysOfWeek)k; //Friday
        Console.Write(day1 + " ," + day2 + " , " + day3);
        Console.Read();
    }
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Tuesday ,Sunday , Friday
```

```
public static void Main(String[] args)
{
    int i = 1, j = 6;
    DaysOfWeek day1, day2;

    day1 = (DaysOfWeek)Enum.ToObject(typeof(DaysOfWeek), i); //Tuesday
    day2 = (DaysOfWeek)Enum.ToObject(typeof(DaysOfWeek), j); //Sunday
    Console.Write(day1 + " ," + day2 );
    Console.Read();
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Tuesday ,Sunday
```

Q10) What is Constructor Chaining In C# ?

Ans:

Constructor chaining is the art of calling constructors from other constructors in the same class or from the base class.

:this() keyword, is a reference to another constructor. First, it call the constructor which is referenced with the :this() keyword, and if that also references another constructor, it will also call that constructor, climbing up the call chain.

```
class employee
{
    1 reference
    public employee() : this(30) // Constructor Chaining
    {
        Console.WriteLine("Parameterless constructor ");
    }
    //public employee()
    //{
    //    employee emp = new employee(30);
    //     Console.WriteLine("Parameterless constructor ");
    //}
    1 reference
    public employee(int id)
    {
        Console.WriteLine("Constructor with  parameter " + id);
    }
}
0 references
public class program
{
    0 references
    public static void Main(String[] args)
    {
        employee emp = new employee();
        Console.Read();
    }
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Constructor with  parameter 30
Parameterless constructor
```

s found

11.) Explain hash table in c# ?

Ans:

* The Hashtable is a non-generic collection that stores key-value pairs.
* Hashtable is the part of System.Collection namespace.
* It implements IDictionary interface.
* The values of hashtable can be null or duplicate.
* Keys in hashtable must be unique and cannot be null.
* Elements are stored in hashtable as DictionaryEntry objects.
* It's value can be accessed by passing key in the indexer e.g. myHashtable[key]

```
0 references
public static void Main(String[] args)
{
    Hashtable ht = new Hashtable();
    ht.Add(1, "Apple");
    ht.Add(2, "Orange");
    ht.Add(3, "Banana");

    foreach (DictionaryEntry item in ht)
    {
        Console.WriteLine(item.Key + ":" + item.Value);
    }

    Console.Read();
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1
3:Banana
2:Orange
1:Apple
```

12.) Explain Dictionary in c#

Ans:

* The Dictionary<TKey, TValue> is a generic collection
* Dictionary<TKey, TValue> stores data in key-value pairs.
* It is the part of System.Collections.Generic namespace.
* Keys in dictionary must be unique and cannot be null.
* Values in dictionary can be null or duplicate.
* It implements IDictionary<TKey, TValue> interface.
* Values can be accessed by passing associated key in the indexer e.g. myDict[key]
* Elements in dictionary are stored as KeyValuePair<TKey, TValue> objects.

```
0 references
public static void Main(String[] args)
{
    Dictionary<int,string> dictionary = new Dictionary<int, string>();
    dictionary.Add(1, "Apple");
    dictionary.Add(2, "Orange");
    dictionary.Add(3, "Banana");

    foreach (KeyValuePair<int,string> item in dictionary)
    {
        Console.WriteLine(item.Key + ":" + item.Value);
    }

    Console.Read();
}
```

```
D:\projects\programs\MyFirst\pp\bin\Debug\netcoreapp3.1\MyFirst\pp.exe
1:Apple
2:Orange
3:Banana
```

13.) What is the difference between Hashtable and Dictionary ?

Hashtable	Dictionary
1. Hashtable is included in the System.Collections namespace.	1. Dictionary is included in System.Collections.Generic namespace.
2. In Hashtable, we can store key/value pairs of the same type or different types.	2. In Dictionary, we can store key/value pairs of same type.
3. The data retrieval in hashtable is slower than Dictionary due to boxing/ unboxing.	3. The data retrieval is faster than Hashtable due to no boxing/ unboxing.
4. It is thread safe.	4. It is also thread safe but only for public static members.
5. Hashtable returns null if we try to find a key that does not exist.	5. Dictionary throws an exception if you try to find a key which does not exist.
6. It doesn't maintain the order of stored values.	6. It always maintain the order of stored values.

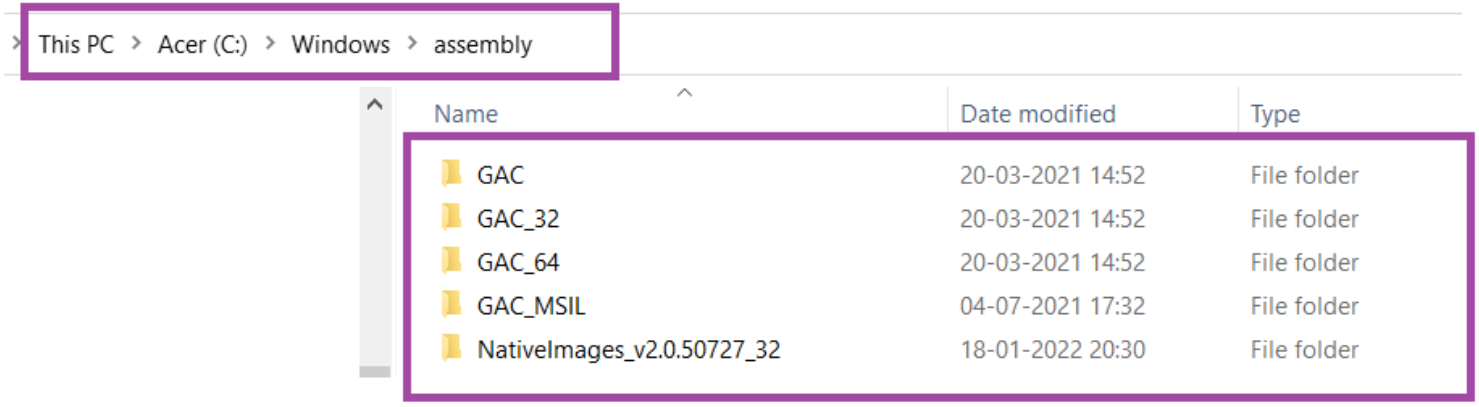
14.) What is GAC and where it is located?

Ans:

GAC stands for Global Assembly Cache. Shared assemblies get stored inside GAC.In order to install an assembly to GAC one needs to use the command :

gacutil.exe i "Assembly name"

It is located in %windir%\assembly (e.g, C:\WINDOWS\assembly) and it is a shared repository of libraries.



15.) What is the difference between Object pooling and connection Pooling ?

Ans:

Object Pooling: Object Pooling tries to keep a pool of objects in memory to be re-used later and hence it will reduce the load of object creation.

In Object pooling, you can control the number of connections.

Connection Pooling: It is the process of taking a connection from a pool of connections, once connection is created in the pool any application can re-use that connection.

In connection pooling, you can control the maximum number reached.

Q16.) Differences between Convert.ToString() and ToString() in c#

Ans:

1. Both these methods are used to convert a value to a string.
2. Convert.ToString() method handles null whereas the ToString() doesn't handle null in C#.
3. ToString() method will throw the null reference exception in case of null value.
3. Convert.ToString() method will not throw an exception.

ToString Throw Exception

```
0 references
public static void Main(String[] args)
{
    string name = null;
    Console.WriteLine(name.ToString());
    Console.Read();
}
```

Exception Thrown

System.NullReferenceException: 'Object reference not set to an instance of an object.'

name was null.

Convert.ToString not

```
0 references
public static void Main(String[] args)
{
    string name = null;
    Console.WriteLine(Convert.ToString(name));
    Console.Read();
}
```

D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe

Q17. What are the advantages of an Array in C# ?

Ans:

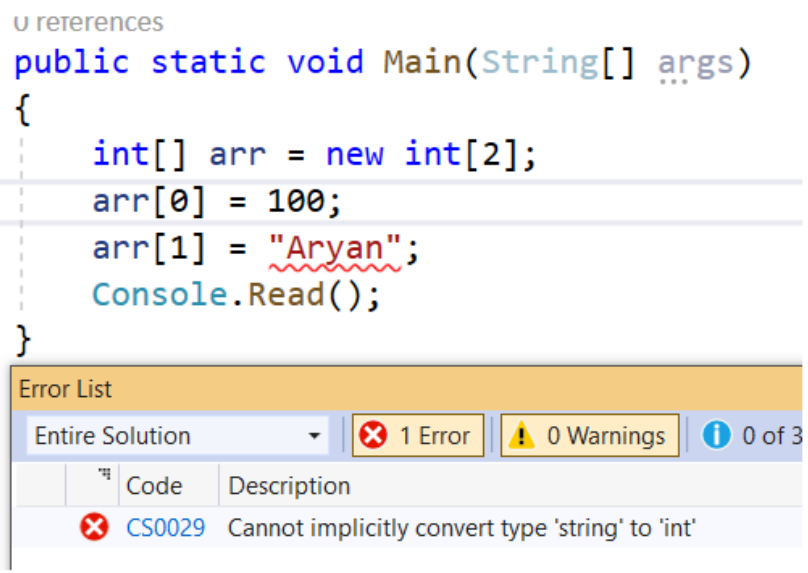
1.It is better way of storing the data of same datatype with same size.

2.It allocates memory in contiguous memory locations for its elements.
so there is no memory overflow in array.

3.It supports multidimensional array.

4.Iterating the arrays using it's index is faster as compared to any other ways like linked list.

5.Arrays in C# are strongly typed, due to this we are getting two advantages.
i) The performance of the application will be much better because of no boxing and unboxing.
ii) Runtime errors will be prevented because of a type mismatch.



Q18. What are the disadvantages of an Array in C# ?

Ans:

1. It allows us to enter only fixed number of elements into it, no way to alter the size of the array once it is declared.
2. We can not insert an element into the middle of an array.
3. Due to array fixed size, we should know in advance how many elements to be stored in the array.
4. Insertion and deletion would be costly since we add / delete the elements from the array, we need to manage memory space also.

```
public static void Main(String[] args)
{
    int[] arr = new int[3];
    arr[0] = 100;
    arr[1] = 6;
    arr[2] = 89;
    Console.Read();
}
```

Q19. What is var keyword in C# ?

Ans:

1.Var is used to declare implicitly typed local variable.

2.It tells the compiler to figure out the type of the variable at compile time

3.Var must be initialized at the time of it's declaration.

4.Var data type was introduced in C# 3.0.

5. var statements are compiled to IL, At compile time.

6. The compile-time type value of var variable cannot be null
but the runtime value can be null.

```
// invalid statements
var data; //need to initialize
var result = null; // can't be null at compile time
```

7. After var variable initialization its data type become fixed to the type of the first initialized data.

```
// invalid statements
var v1 = "testStr";
v1 = 123;
// int value can't be assign to implicitly type string variable v1
```

```
0 references
public static void Main(String[] args)
{
    string str = "test";
    var s2 = str;
    Console.WriteLine(s2); //will print "test"
    s2 = "abc";
    Console.WriteLine(s2); //will print "abc"
    s2 = null;
    Console.WriteLine(s2); //will print null value
    Console.Read();
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
test
abc
```

Q20.When to use a var keyword in C# ?

Ans:

* If local variable types are known to you, then don’t use var.

* Use var when you’re not sure what type of data will be stored in a variable.

Q21.What is Dynamic keyword in C# ?

Ans:

1. The dynamic keyword is used to declare dynamic types.
2. C# 4.0 introduced dynamic type.
3. It tells the compiler that the object is defined as dynamic and
4. It skip type-checking at compile time instead it resolves type at run time.
5. Dynamic types change data types at run-time based on the assigned value.
6. It does not support the intellisense in visual studio.
7. The compiler does not throw an error on dynamic type at compile time if there is no type checking for dynamic type.
8. Dynamic type variables are converted to other types implicitly.

```
dynamic dt = 100;
int num = dt;

dt = "test";
string str = dt;

Console.Read();
```

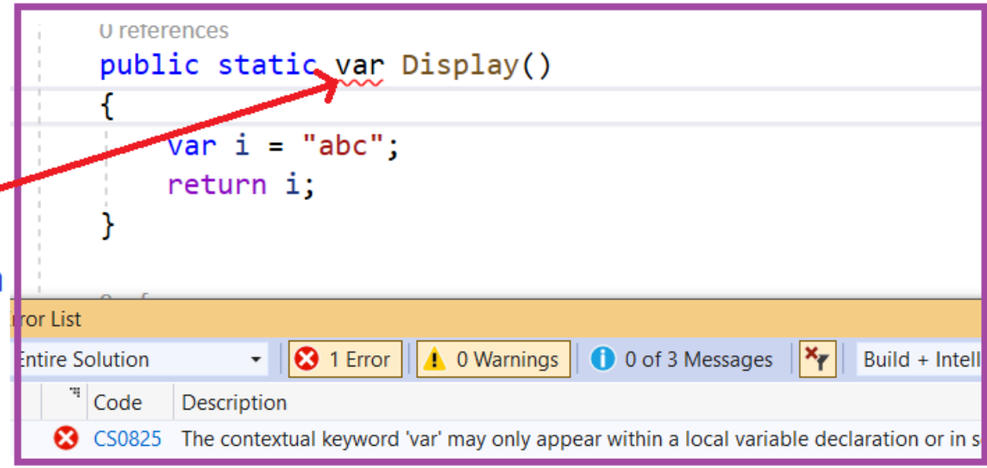
```
0 references
public static void Main(String[] args)
{
    dynamic dynamicVar = "test";
    Console.WriteLine(dynamicVar + ": " + dynamicVar.GetType());
     dynamicVar = 100;
    Console.WriteLine(dynamicVar + ": " + dynamicVar.GetType());
     dynamicVar = true;
    Console.WriteLine(dynamicVar + ": " + dynamicVar.GetType());
    Console.Read();
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
test: System.String
100: System.Int32
True: System.Boolean
```

Q22.) Differences between Var and Dynamic type in C# ?

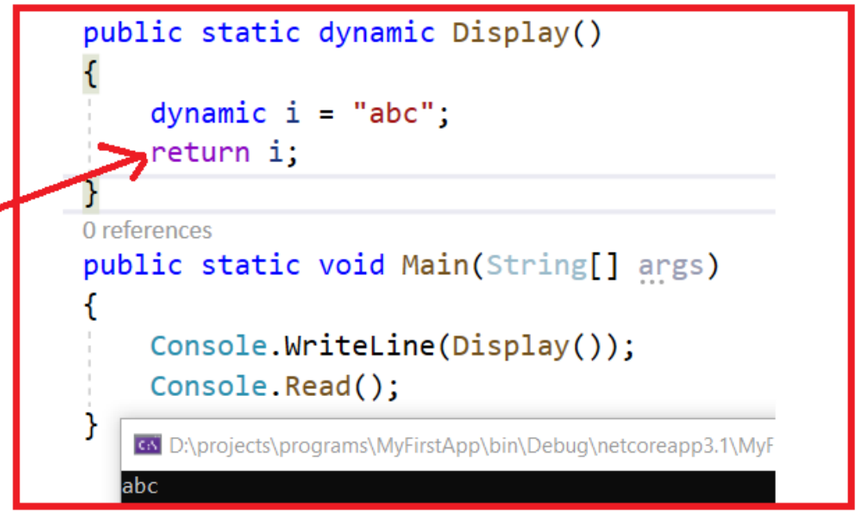
Var

1. Var was introduced with C# 3.0
2. It is type-safe as compiler has all information about the variable, so it doesn't cause any issue at run-time.
3. It can store any type of value.
4. Var type cannot be passed as a method argument and method cannot return var type. Var type work in the scope where it defined.
5. It is mandatory to initialize var types at the time of declaration.



Dynamic

1. Dynamic was introduced with C# 4.0
2. It is not type safe as compiler doesn't have any information about the type of variable.
3. It can store any type of the variable.
4. Dynamic type can be passed as a method argument and method also can return dynamic type.
5. Dynamic type variables need not be initialized when declared.



23.) What is Interface in C#?

* An Interface is a contract which is an agreement on what the class will provide to an application.
* An interface defines what operations a class can perform.
* To declare an interface, we use interface keyword.
* It contains are declarations of events, indexers, methods & properties.
* Multiple inheritance is possible with the help of Interfaces
* A class or struct can implement interfaces implicitly or explicitly.
* Implement interface explicitly using InterfaceName.MemberName
* An interface can inherit one or more interfaces.

```
interface IEmployee
{
    0 references
    public int Name { get; set; }
    2 references
    public void Display();
}
2 references
class employee : IEmployee
{
    2 references
    public void Display()
    {
        Console.WriteLine("IEmployee display" );
    }
```

*Interface cannot contain fields, and auto-implemented properties

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 3 Messages | Build + Int

Code	Description
CS0535	'employee' does not implement interface member 'IEmployee.Name'

* Interface can't contain instance fields but static

```
interface IEmployee
{
    static int item = 100;
     int item2 = 199;
    2 references
    void Display();
    2 references
    void Display2(int x, int y);
}
```

Error List

Entire Solution | 1 Error | 0 Warnings

Code	Description
CS0525	Interfaces cannot contain instance fields

* No other modifiers are allowed except public inside interface.

```
interface IEmployee
{
    2 references
    public void Display();
    2 references
    private void Display2(int x, int y);
}
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 4 Messages | Build + IntelliSense

Code	Description
CS0501	'IEmployee.Display2(int, int)' must declare a body because it is not marked abstract, extern, or partial

Q24.) What are the benefit of using interfaces in c# ?

Ans:

* Interfaces allow us to implement polymorphic behaviour.
* Complete Abstraction can be achieved by the implementation of Interface.
* We can achieve multiple class inheritance in C# using Interfaces.
*Interfaces are great for implementing Inverson of Control or Dependancy Injection.
*Interfaces allow to develop very loosely coupled systems.

Note: *Loose Coupling promotes single-responsibility and separation of concerns.
* Loose coupling reduces the inter-dependencies between the components.

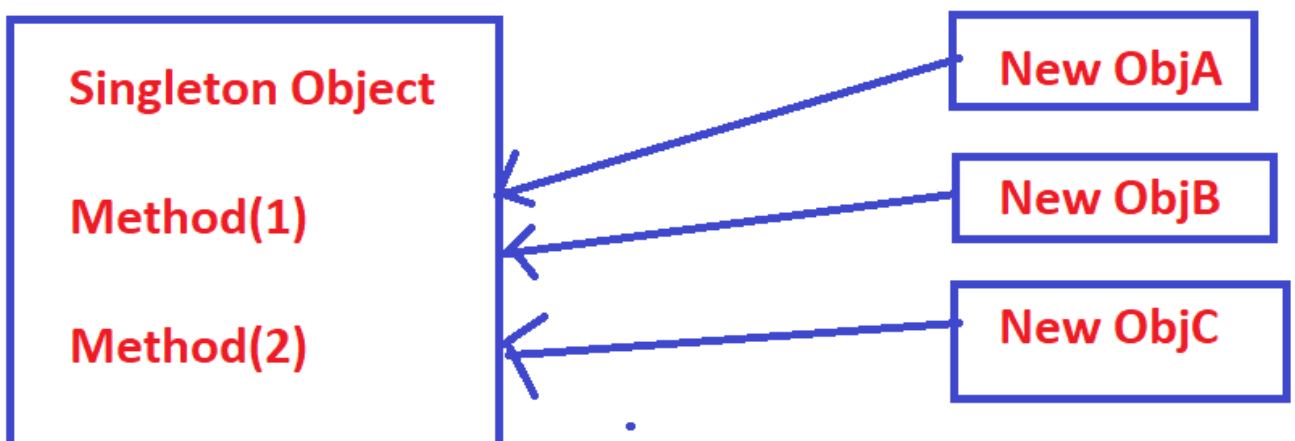
```
1 reference
interface IEmployee
{
    2 references
    public void Display();
}
2 references
class employee : IEmployee
{
    2 references
    public void Display()
    {
        Console.WriteLine("IEmployee display" );
    }
}
0 references
```

Q25.) What is Singleton Pattern in C#?

Ans:

1) Singleton Design Pattern is used when we need to ensures that only one instance of a particular class is going to be create.

2) It provides single global access to that instance for the entire application.



3) In the above diagram, different providers (ObjA, ObjB, ObjC ....n) trying to get the singleton instance.

Q26.) What is Singleton Pattern in C#? (Practical Implementation)

Class declared as sealed to prevent inheritance

```
8 references
public sealed class Singleton
{
    private static Singleton obj = null;
    private int counter = 0;
    2 references
    public static Singleton GetObject
    {
        get
        {
            if(obj == null)
            {
                obj = new Singleton();
            }
            return obj;
        }
    }
    1 reference
    private Singleton()
    {
        counter++;
        Console.WriteLine("Singleton constructor called: " + counter);
    }
    2 references
    public void display(string clientName)
    {
        Console.WriteLine(clientName);
    }
}
```

```
public class program
{
    0 references
    public static void Main(String[] args)
    {
        Singleton clientoBJA = Singleton.GetObject;
        clientoBJA.display("ClientA");
        Singleton clientoBJB = Singleton.GetObject;
        clientoBJA.display("ClientB");
        Console.Read();
    }
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Singleton constructor called: 1
ClientA
ClientB
```

Constructor called only one times, when first object is created. After that other client uses the same object.....

Q27.) What are the Benefits of using Singleton Design pattern?

Ans:

1) Control Instance Creation : Singleton ensures that all objects access the single instance. It prevents other objects to instantiate their own copies of the Singleton object.

2) Lazy loading: it is possible to implement Lazy loading using Singleton.

3) Saves Memory: When we create object of any class, it occupies space in memory, as Singleton class has only one instance, so it saves lot of memory.

4) Interfaces: Using the singleton approach Interfaces can be implemented.

Q28) Disadvantages of using the Singleton Design Pattern in C# ?

Ans:

1) Singleton introduces a global state into an application, so unit testing is more difficult.

2) Singleton reduces the potential for parallelism within a program, because to access the singleton in a multi-threaded system, an object must be serialized by using lock.

Q29.) What is Enum in C#?

Ans:

Enum is a value type with a set of related named constants. We cannot change the values associated with enums since Enums are treated as Constants.

1.The "enum" keyword is used to declare an enumeration.
3.Except Char, enum will be having an underlying type as Integral Type.
4.The default underlying type enum elements is int.
5.By default, the first enumerator has the value 0, after that each successive enumerator is increased by 1.
6.Enums make our code much readable and understandable.
7.We can declare Enum either in a class or out side of a class.
8.Enum cannot inherit from other Enum.
9. Enums are created on the stack, not on the heap as it is value types

```
enum DaysOfWeek
{
    Sunday = 1, Monday, Tuesday, Wednesday, Thursday , Friday, Saturday
}
0 references
public class program
{
    0 references
    public static void Main(String[] args)
    {
        Console.WriteLine("DaysOfWeek {0} {1}", (int)DaysOfWeek.Sunday, DaysOfWeek.Sunday);
        Console.WriteLine("DaysOfWeek {0} {1}", (int)DaysOfWeek.Monday, DaysOfWeek.Monday);
        Console.WriteLine("DaysOfWeek {0} {1}", (int)DaysOfWeek.Thursday, DaysOfWeek.Thursday);
        Console.WriteLine("DaysOfWeek {0} {1}", (int)DaysOfWeek.Saturday, DaysOfWeek.Saturday);
        Console.ReadLine();
    }
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
DaysOfWeek 1 Sunday
DaysOfWeek 2 Monday
DaysOfWeek 5 Thursday
```

Q30.) Difference between Struct and Enum in C# ?

Struct	Enum
1) To declare a structure, we use “struct” keyword.	1) To declare enum, we use “enum” keyword .
2) Structure is a value type and a collection of variables of different data types in a single unit.	2) Enum is a value type with a set of related named constants.
3) Struct can contain both data variables and methods.	3) Enum can contain data types only.
4) Struct supports private but not protected access specifier.	4) Enum does not have private and protected access specifier.
5) Struct supports encapsulation.	5) Enum doesn’t.
6) When the structure is declared, the values of its objects can be modified.	6) Enums value cannot be changed, Once it is declared, otherwise, the compiler will throw an error.

Q31.) Difference between String and string in C# ?

String

1) String is a class in .NET framework.

2) String can be used as a variable name.

var String = DateTime.Now; // Compiles without exception

3) String, can be used as a parameter for the nameof expression.

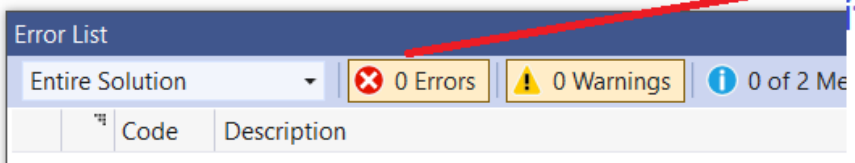
var str2 = nameof(String); // Compiles with no syntax error.

4) We can use "String" as a generic type parameter.

```
public class GenericList<String> { };
```

Zero error

its eyes



string

1) "string" is an alias of "String". It is a keyword to create string variable.

2) string cannot be used as a variable name.

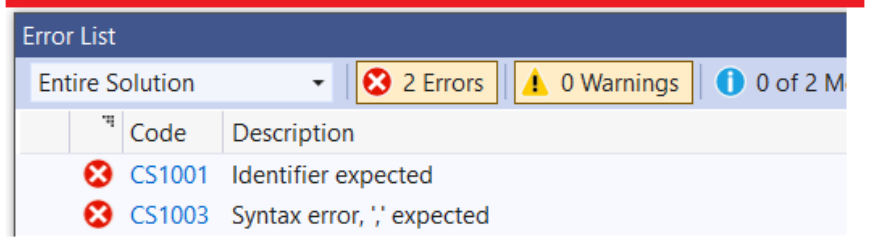
var string = DateTime.Now; // throw compile time exception

3) string, can't be used as a parameter of the "nameof" expression.

var str1 = nameof(string); // Syntax error occured.

4) We can't use a "string" keyword as a generic type parameter.

```
public class GenericList<string> { };
```



Q32) How to check whether dictionary contains any value or not in C# ?

Ans: Dictionary.ContainsValue() :

This method is used to check whether the Dictionary contains any specific value or not.

```
Dictionary<int, string> dictionary = new Dictionary<int, string>();
// Adding the items
dictionary.Add(100, "Aryan");
dictionary.Add(2000, "Sujit");
// Checking the value "India" but
// here "India" is the key
if (dictionary.ContainsValue("Aryan"))
    Console.WriteLine("Aryan");
else
    Console.WriteLine("Aryan is not present.");

Console.ReadLine();
```

Q33) How to check whether dictionary contains any Keys or not in C# ?

Ans: Dictionary.ContainsKey() :

This method is used to check whether the Dictionary contains any specific Key or not.

```
Dictionary<int, string> dictionary = new Dictionary<int, string>()
// Adding the items
dictionary.Add(100, "Aryan");
dictionary.Add(2000, "Sujit");
// Checking the value "India" but
// here "India" is the key
if (dictionary.ContainsKey(2000))

    Console.WriteLine(dictionary[2000]);
else
    Console.WriteLine("2000 key does not exist.");

Console.ReadLine();
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Sujit
```

Q34) What are Async and Await in C# ?

Method1 and method2 are independent of each other...it will execute asychronously

Ans:

1) Aync and Await are simply markers, which mark code positions from where the control should resume after a task or a thread completes.

2) It introduced in .NET 4.5 Framework

3) An async method runs synchronously until it reaches its first await operator, at which point the method is suspended while the awaited task is completed. In the meantime, the control returns to the caller of the method.

4) The await operator suspends the evaluation of the enclosing async method until the asynchronous operation completes.

5)

```
public static void Main(String[] args)
{
    Method1();
    Method2();
    Consol
}
```

void program.Method2()

```
1 reference
public static async Task Method1()
{
    await Task.Run(() =>
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine("method 1");
            Task.Delay(100).Wait();
        }
    });
}
```

```
1 reference
public static void Method2()
{
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine("Method2");
        Task.Delay(100).Wait();
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Method2
method 1
method 1
Method2
Method2
```

Q34) What are Async and Await in C# ?

N.B: From c# onwards we can use async in Main method also.

Method3's input is depedent on method1's output(i.e count), so we put await on method1, which first execute Method1 completely, then only move to Method3....

```
static async Task Main(string[] args)
{
    await callMethod();
    Console.ReadKey();
}
```

1

```
1 reference
public static async Task callMethod()
{
    Method2();
    var count = await Method1();
    Method3(count);
}
```

2

```
1 reference
public static async Task<int> Method1()
{
    int count = 0;
    await Task.Run(() =>
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine(" Method 1");
            count += 1;
        }
    });
    return count;
}
```

4

```
1 reference
public static void Method2()
{
    for (int i = 0; i < 5; i++)
    {
        Console.WriteLine(" Method 2");
    }
}
```

3

```
1 reference
public static void Method3(int count)
{
    Console.WriteLine("Total count is " + count);
}
```

5

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\My
Method 2
Method 2
Method 2
Method 2
Method 2
Method 1
Method 1
Method 1
Method 1
Method 1
Method 1
Method 1
Method 1
Method 1
Method 1
Total count is 10
```

Q35) What is AutoMapper in C# ?

Ans:

The AutoMapper in C# is a mapper between two objects. It is an object-object mapper. It maps the properties of two different objects by transforming the input object of one type to the output object of another type.

```
3 references
public class Employee
{
    1 reference
    public string EmpName { get; set; }
    1 reference
    public int EmpSalary { get; set; }
    1 reference
    public string EmpAddress { get; set; }
}
2 references
public class EmployeeDTO
{
    1 reference
    public string EmpName { get; set; }
    1 reference
    public int EmpSalary { get; set; }
    1 reference
    public string EmpAddress { get; set; }
}
}
```

1. Install AutoMapper : PM> Install-Package AutoMapper

```
using AutoMapper;

namespace MyFirstApp
{
    0 references
    public class program
    {
        0 references
        static void Main(string[] args)
        {
            var config = new MapperConfiguration(cnf =>
            cnf.CreateMap<Employee, EmployeeDTO>());

            // Creating employee object
            Employee emp = new Employee();
            emp.EmpName = "Amit";
            emp.EmpAddress = "f-90";
            emp.EmpSalary = 10000;

            // Use AUTO MAPPER
            var mapper = new Mapper(config);
            var employeeDTO = mapper.Map<EmployeeDTO>(emp);

            Console.WriteLine("EmployeeDTO: " + employeeDTO.EmpName + ", "
                + employeeDTO.EmpSalary + ", " + employeeDTO.EmpAddress);
            Console.ReadKey();
        }
```

Q37) What are the 4 pillars for OOPs ?

Ans:

The four pillars for OOP are:

1) Abstraction.
2) Encapsulation.
3) Inheritance.
4) Polymorphism.

Q38) What is Abstraction in C# ?

Ans:

* The process of showing the essential features without including the background details is called Abstraction.
* In simple words, expose what is necessary and hide what is unnecessary from the outside world.

Example: To ride the bike, we need to know the clutch, break, accilator,etc.. but not Engine, wheel.

* So the necessary methods and properties are exposed by using “public” access modifier whereas the unnecessary methods and properties hidden by using the “private” access modifier.

* We can access the public members of the Bike class using the Bike instance, but we get Compiler Error while processing the private members of the Bike.

```
public class Bike
{
    private string _bikeName = "HondaCity";
    1 reference
    public string BikeName
    {
        get { return _bikeName; }
        set {  _bikeName = value;}
    }
    1 reference
    public void Clutch()
    {
        Console.WriteLine("Clutch");
    }
    1 reference
    public void Brake()
    {
        Console.WriteLine("Clutch");
    }
    1 reference
    private void Engine()
    {
        Console.WriteLine("Clutch");
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Bike objBike = new Bike();
        objBike.Brake();
        objBike.Clutch();
        string bikeNAME = objBike.BikeName;
        objBike.Engine();
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 4 Messag

Code	Description
CS0122	'Bike.Engine()' is inaccessible due to its protection level

Q39) What is Encapsulation in C#?

Ans:

*The process of binding the data and functions together into a single unit is called encapsulation.

* Class is the better example of encapsulation.

* In other word, it define a class by hiding its internal data members from outside the class.

*Encapsulation is also called data hiding.

* Encapsulation is implemented

1) By declaring the variables as private.

2) By public setter and getter methods or properties to access private variables.

*We declare variables as private to prevent them directly from outside the class.

* So by using Encapsulation, we can validate & protect the user-given data according to business requirement

```
public class Employee
{
    private string employeeName = "Amit Jha";
    private int employeeNo  = 12706;
    1 reference
    public void AddEmployee()
    {
        if(ValidateEmployee())
        {
            DbCooectionObject();
        }
        else
        {
            Console.WriteLine("validation failed");
        }
    }
    1 reference
    private bool ValidateEmployee()
    {
        if(string.IsNullOrEmpty(employeeName) || employeeNo < 1)
        {
            return false;
        }
        return true;
    }
    1 reference
    private bool DbCooectionObject()
    {
        // coding.....
                                                    ee Added Successfully.");
    static void Main(string[] args)
    {
        Employee emp  = new Employee();
        emp.AddEmployee();
        emp.ValidateEmployee();
        emp.DbCooectionObject();
    }
```

```
r List
tire Solution    2 Errors   0 Warnings   0 of 4 Messages
Code    Description
CS0122  'Employee.ValidateEmployee()' is inaccessible due to its protection level
CS0122  'Employee.DbCooectionObject()' is inaccessible due to its protection level
```

Q40) What is the difference between Abstraction and Encapsulation in C#?

Encapsulation:

* It hides irrelevant data from the user or we can say that it protects the data.
*For example, when we purchage a phone, we can never see how the internal circuit board works, it is irrelevant pieces of information, and due to this they are encapsulated inside a cabinet.
* Encapsulation solves the problem at the implementation level.
* Encapsulation is inner layout in terms of implementation.

Abstraction:

* It is a mechanism that will show only the relevant information to the user.
* For example, when we purchage a phone. we can use many different types of functionalities such as a camera, calling , recording , mp3 , multimedia etc. These are the outer layers and it is nothing but an example of abstraction, because we are only seeing the relevant information instead of their internal work.
* Abstraction solves the problem at the design level.
* Abstraction is outer layout in terms of design.

Q41) What is Gang Of Four(GOF) ?

Ans:

* The "Gang Of Four" are the authors of the Book:
“Design Patterns: Elements of Reusable Object-Oriented Software”.

* This book describe 23 Object-Oriented Design Patterns and various development techniques.

The four authors are:

1) Erich Gamma 2) Ralph Johnson
3) John Vlissides 4) Richard Helm

They categorized all design patterns into 3 categories:

1) Creational Patterns
2) Structural Patterns
3) Behavioral Patterns

Q42) What is Shallow Copy in C# ?

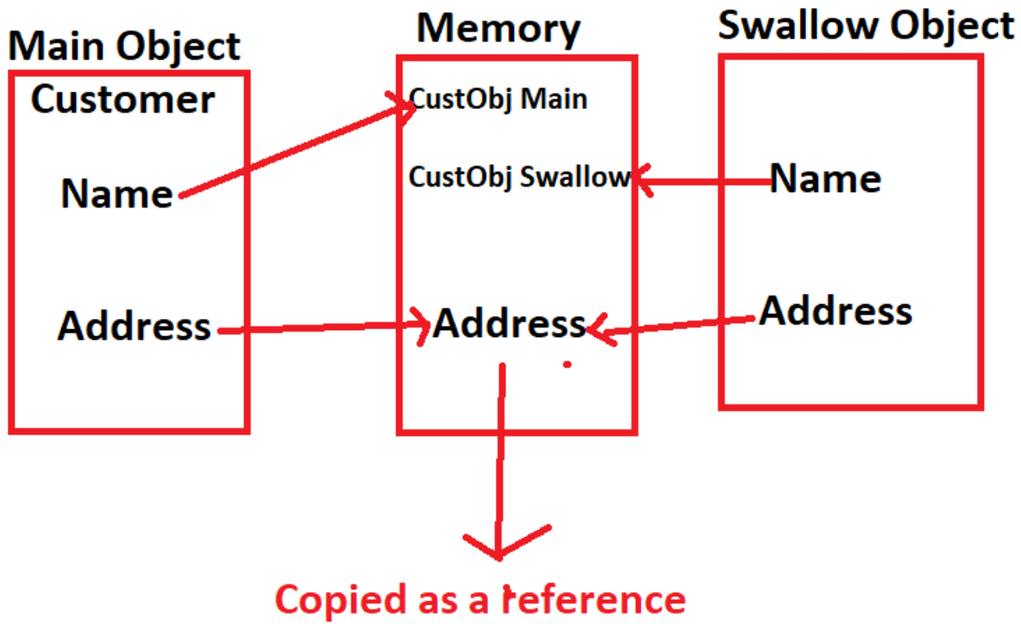
Ans:

Gang of Four Says: "Prototype Design Pattern Specify the kind of objects to create using a prototype instance and create new objects by copying this prototype"

In Simple Word, instead of creating object every time from scratch, we can make copies of an original instance.

Shallow Copy:

A Shallow Copy is about copying the object's value type fields into the target object and the object's reference types are copied as references into the target object.



Q42) What is Shallow Copy in C# ?

Ans:

Gang of Four Says: "Prototype Design Pattern Specify the kind of objects to create using a prototype instance and create new objects by copying this prototype"

In Simple Word, instead of creating object every time from scratch, we can make copies of an original instance.

step-01

```
class Customer
{
    public int Id;
    public Address addr;
    1 reference
    public Customer(int id, Address address)
    {
        this.Id = id;
        addr = new Address(address.address1);
    }

    // method for cloning object
    1 reference
    public object Shallowcopy()
    {
        return this.MemberwiseClone();
    }
}
6 references
class Address
{
    public string address1;
    2 references
    public Address(string addr1)
    {
        this.address1 = addr1;
    }
}
```

step-02

```
Address addr = new Address("f-192");

Customer c1 = new Customer(548, addr);

// Performing Shallow copy
Customer c2 = (Customer)c1.Shallowcopy();

Console.WriteLine("Before Changing: ");

// Before changing the value of c2 id and address1
Console.WriteLine(c1.Id);
Console.WriteLine(c2.Id);
Console.WriteLine(c2.addr.address1);
Console.WriteLine(c1.addr.address1);
```

```
Select Microsoft
Before Changing:
548
548
f-192
f-192
```

step-03

```
// changing the value of c2 id and address1

c2.Id = 500;
c2.addr.address1 = "t-900";

Console.WriteLine("\nAfter Changing:

// After changing the value of
// c2 id and address1
Console.WriteLine(c1.Id);
Console.WriteLine(c2.Id);
Console.WriteLine(c2.addr.address1);
Console.WriteLine(c1.addr.address1);
```

```
Select Microsoft
Before Changing:
548
548
f-192
f-192

After Changing:
548
500
t-900
t-900
```

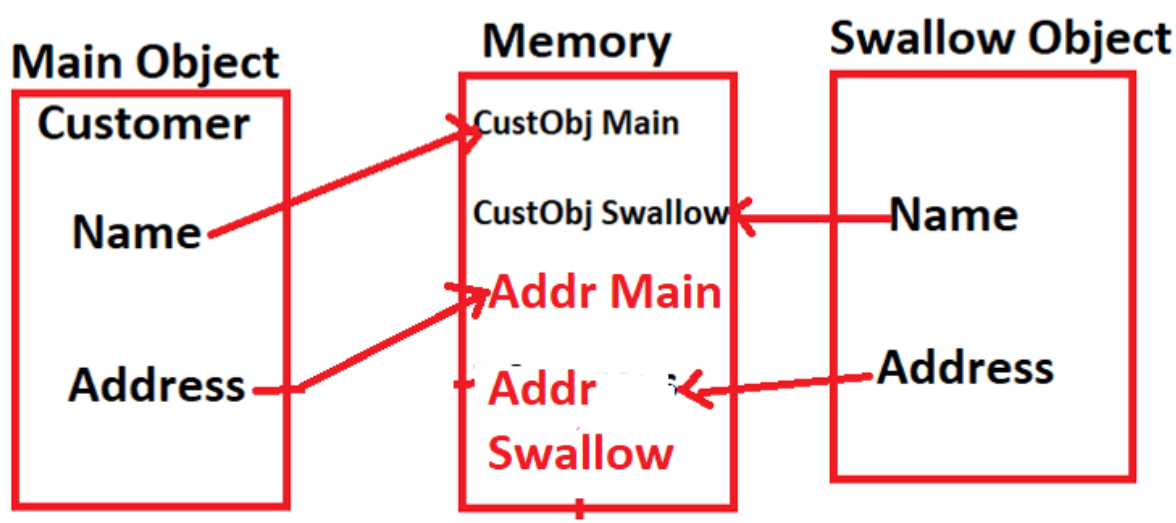
Q43) What is Deep Copy in C# ?

Ans:

Gang of Four Says: "Prototype Design Pattern Specify the kind of objects to create using a prototype instance and create new objects by copying this prototype"

In Simple Word, instead of creating object every time from scratch, we can make copies of an original instance.

Deep Copy : Deep Copy copies an object's value types and reference types into a new copy of the target objects.



Q43) What is Deep Copy in C# ?

In Deep Copy, it does not matter, the field type is value type or reference type. Deep copy always makes a copy of whole data and store it in a different memory location so if we change c2, c1 will not affected vice versa.

step-02

```
0 references
static void Main(string[] args)
{
    Address addr = new Address("f-192");

    Customer c1 = new Customer(548, addr);

    // Performing Shallow copy
    Customer c2 = (Customer)c1.DeepCopy();

    Console.WriteLine("Before Changing: ");

    // Before changing the value of c2 id and address1
    Console.WriteLine(c1.Id);
    Console.WriteLine(c2.Id);
    Console.WriteLine(c2.addr.address1);
    Console.WriteLine(c1.addr.address1);

    // changing the value of c2 id and address1

    c2.Id = 500;
    c2.addr.address1 = "t-900";

    Console.WriteLine("\nAfter Changing: ");

    // After changing the value of
    // c2 id and address1
    Console.WriteLine(c1.Id);
    Console.WriteLine(c2.Id);
    Console.WriteLine(c2.addr.address1);
    Console.WriteLine(c1.addr.address1);
```

```
Select Microsoft Visual Studio Debug Cons
Before Changing:
548
548
f-192
f-192

After Changing:
548
500
t-900
f-192

D:\projects\programs\MyFirstApp\bin
To automatically close the console
le when debugging stops.
Press any key to close this window
```

step-01

```
class Customer
{
    public int Id;
    public Address addr;
    2 references
    public Customer(int id, Address address)
    {
        this.Id = id;
        addr = new Address(address.address1);
    }

    // method for cloning object
    1 reference
    public Customer DeepCopy()
    {
        Customer deepcopy = new Customer(this.Id, addr);
        return deepcopy;
    }
}
6 references
class Address
{
    public string address1;
    2 references
    public Address(string addr1)
    {
        this.address1 = addr1;
    }
}
```

Q44) Can "var" hold null value in c#?

* Cannot assign <null> to an implicitly-typed variable.

```
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        var abc = null;

    }
}
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 2 Messag

	Code	Description
⊗	CS0815	Cannot assign <null> to an implicitly-typed variable

When we say, var abc = null;
then the compiler cannot resolve this because there's no type bound to null.

But we can make it like below.

```
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        string result = null;
        var finalResult = result;
    }
}
```

It will work bcz now finalResult can know its type at compile time.

Q45) How do you sort a dictionary by value in c#?

```
0 references
static void Main(string[] args)
{
    Dictionary<string, string> myDict = new Dictionary<string, string>();
    myDict.Add("one", "ary");
    myDict.Add("four", "jitu");
    myDict.Add("two", "zart");
    myDict.Add("three", "blub");

    List<KeyValuePair<string, string>> myList = myDict.ToList();

    myList.Sort((x, y) => x.Value.CompareTo(y.Value));
    foreach (var item in myList)
    {
        System.Console.WriteLine(item.Key + ":" + item.Value);
    }
}
}
```

Convert dictionary into list, and use Linq query to sort.

```
Microsoft Visual Studio Debug Console
one:ary
three:blub
four:Tapan
two:zart

D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe (process 8796) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close
le when debugging stops.
Press any key to close this window . . .
```

Q46) Differences between a float, double and a decimal in C#?

Float

1) Float is a 32-bit single-precision floating point type with range 3.4 x 1038 to + 3.4 x 1038

Memory Size is 4 bytes.

float a = 3.5f;

2) Float is less accurate than Double and Decimal.

Double

1) Double is a 64-bit double-precision floating point type with range (+/-)5.0 x 10-324 to (+/-)1.7 x 10308

Memory Size = 8 bytes.

double dValue = 5.78788

2) Double is more accurate than Float but less accurate than Decimal.

Float < Double < Decimal

Decimal

1) Decimal is a 128-bit precise decimal values with 28-29 significant digits with range (-7.9 x 1028 to 7.9 x 1028) / 100 to 28

Memory Size = 16 bytes.

decimal decValue = 1.0M;

2) Decimal is more accurate than Float and Double.

Q47) How to calculate someone's age based on a DateTime type birthday ?

Approach 1

```
static class Age
{
    1 reference
    public static Int32 GetAge(this DateTime dob)
    {
        var today = DateTime.Today;
        var a = (today.Year * 100 + today.Month) * 100 + today.Day;
        var b = (dob.Year * 100 + dob.Month) * 100 + dob.Day;
        return (a - b) / 10000;
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        DateTime yourDateOfBirth = Convert.ToDateTime("1987-08-01");
        //Approach 2
        int age = Age.GetAge(yourDateOfBirth);
        Console.WriteLine("your Age Is :" + age);
    }
}
```

```
Microsoft Visual Studio Debug Console
your Age Is :34
```

Approach-2

```
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        DateTime yourDateOfBirth = Convert.ToDateTime("1987-08-01");
        ////Approach 1
        int today = int.Parse(DateTime.Now.ToString("yyyyMMdd"));
        int dob = int.Parse(yourDateOfBirth.ToString("yyyyMMdd"));
        int yourAge = (today - dob) / 10000;

        Console.WriteLine("your Age Is :" + yourAge);
    }
}
```

```
Microsoft Visual Studio Debug Console
your Age Is :34
```

Q48) Difference Between Int and Int32 in C# ?

Ans:

Int32 vs int

* Int32 is a type provided by .NET.
* int is an alias for Int32 in C#.
* They compile to the same code. so there is no difference at execution time.
*" Int32" can be only used with System namespace otherwise it will generate an error but it is not true with "int".

Int32

```
0 references
static void Main(string[] args)
{
    Int32 data = 456;
    Console.WriteLine(data);
}
```

```
Microsoft Visual Studio Debug Console
456

D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3
To automatically close the console when debugging sto
le when debugging stops.
Press any key to close this window . . .
```

int

```
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        int data = 456;
        Console.WriteLine(data);
    }
}
```

```
Microsoft Visual Studio Debug Console
456

D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyF
To automatically close the console when debugging stops, ena
le when debugging stops.
Press any key to close this window
```

Q49) Difference between Int16, Int32 and Int64 in C# ?

Int16

1 Int16 is a FCL(framework class library) type.

2.short is mapped to Int16.

3. It is signed and takes 16 bits.

4. short is a value type.

5. Represent System.Int16 struct.

6. Capacity: minimum -32768 and maximum +32767 .

Int32

1. Int32 is a FCL type.

2. int is mapped to Int32.

3. Int32 is signed and takes 32 bits.

4. Int32 is a value type

5. Represent System.Int32 struct.

6. Capacity minimum -2147483648 and maximum +2147483647.

Int64

1. It is also a FCL type.

2 long is mapped to Int64.

3. Int64 is signed and it takes 64 bits.

4. Int64 is a value type.

5. Represent System.Int64 struct.

6. Capacity:
minimum : 9,223,372,036,854,775,808
maximum : 9,223,372,036,854,775,807

Q50) What is type-safe in C# ?

Ans:

Type safety in c# has introduced to prevent the objects of one type from peeking into the memory assigned for the other object.

Writing safe code means to prevent data loss during conversion of one type to another type.

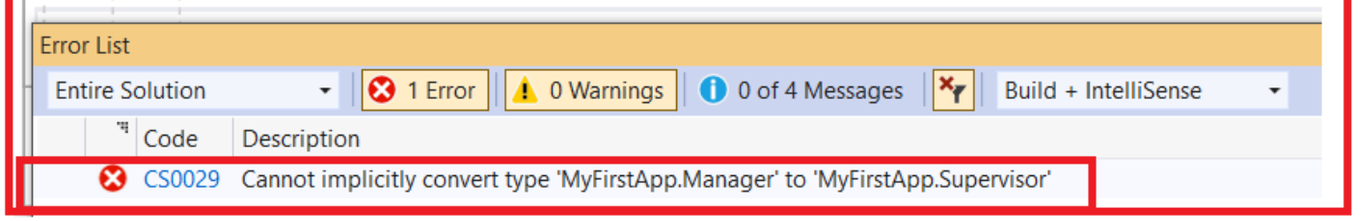
In simple word, Type-safe code accesses only the memory locations it is authorized to access.

1

```
3 references
public class Manager
{
    1 reference
    public string ManagerName { get; set; }
}
1 reference
public class Supervisor
{
    0 references
    public int Id { get; set; }
    0 references
    public int Name { get; set; }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
```

2

```
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Manager mgr = new Manager();
        mgr.ManagerName = "sujit";
        Supervisor Suvrobj = (Manager)mgr; //will get compile time error
```



Q51) Difference between OOPS and Procedural Oriented Programming in c#?

OOPS	Procedural Programming
1) It is object-oriented .	1) It is structure/procedure-oriented.
2) It follows a bottom-up approach.	2) It follows a top-down approach.
3) Objects communicate with each other via member functions.	3) Data moves freely within the system from one function to another.
4) Due to abstractionData, hiding is possible in OOPS . So, it is more secure than procedural programming.	4) Less secure than OOPs.
5) Possible through function overloading and operator overloading.	5) Overloading is not possible in PP.

Q52) What are the advantages of OOPS ?

Ans:

1) Modularity: Objects are self-contained and each bit of functionality does its own work and leave the other bits alone. Suppose if any object brokes then we can identify the call where the problem occurs we don't have to go line-by-line through all your code. That's the beauty of encapsulation.

2) Reusability through inheritance: Create one generic class (Phone), and then define the subclasses (AndroidPhone & WindowPhone) that will inherit the Phone class common functionality.

3) Flexibility through polymorphism: Instead of creating class for similar functionality we can overload/override the method by changing only signature. e.g Area() method can be area of square/triangle/rectangle.....etc. so we will overload it.

4) Provide better security through data abstraction: we can only show what is required and will hide the extra details from end user.

Q53) What are the limitations of OOPS?

Ans:

1)The size of the programs created using this approach may become larger than the programs written using procedure programming(PP) approach.

2)Programmers need to have good designing and programming skill. It requires a substantial amount of pre-work and planning.

3) Sometimes OOP code become difficult to understand if documentation not provided.

4) In certain scenarios, these programs can consume a large amount of memory.

Q54) How to prevent a class from being inherited in C#.NET?

Ans:

Option1

* The sealed modifier is used to prevent a class from being inherited.

* You will get an error if a sealed class is specified as the base class of another class.

* Sealed class cannot be an abstract class.

* By sealing a class , ensure that no class can be derived from that class.

Option2

In order to avoid inheritance, we can use private constructor of that class and implement singleton pattern in C#.

Q55) Explain Polymorphism and its type in c# ?

Ans:

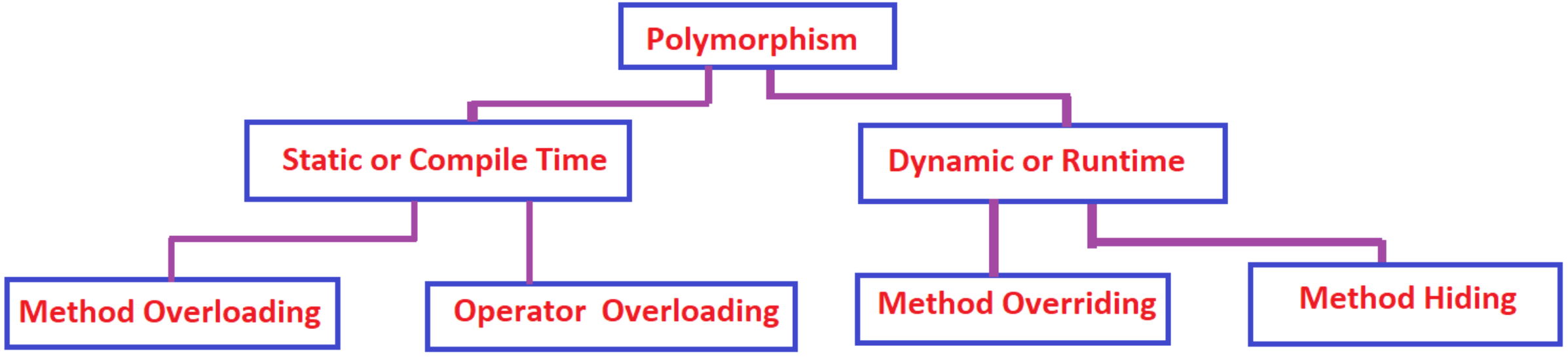
The word polymorphism means having in many forms. It is often expressed as through one interface, multiple functions .

Polymorphism provides the ability to a class to have multiple implementations of a method with the same method name.

There are two types of polymorphism:

1) Static / Compile Time Polymorphism.

2) Dynamic / Runtime Polymorphism.



Q56) Can we allow a class to be inherited, but prevent method from being overridden ?

Ans:

Yes, put sealed keyword before the method.

AVP class, can inherit other method of Supervisor class but not the sealed method (i.e Dispay())

```
1 reference
public class Manager
{
    3 references
    public virtual void Display()
    {
        Console.WriteLine("Manager");
    }
}
3 references
public class Supervisor: Manager
{
    2 references
    public sealed override void Display()
    {
        Console.WriteLine("Supervisor");
    }
}
0 references
public class AVP : Supervisor
{
    3 references
    public override void Display()
    {
        Console.WriteLine("Supervisor");
    }
}
0 references
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 2 Messages | Build + IntelliSense

Code	Description
CS0239	'AVP.Display()': cannot override inherited member 'Supervisor.Display()' because it is sealed

Q57) Why static constructor is parameterless in c# ?

Ans:

The user can't call static constructor so it is not possible to pass parameter.

It is invoked when you access a static member or create an instance of the class, it is called by the runtime. So we don't have any control to call it.

```
public static class Employee
{
    2 references
    public static string Name { get; set; }
    0 references
    static Employee(string name)
    {
        Name = name;
    }
}
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 2 Messages

Code	Description
CS0132	'Employee.Employee(string)': a static constructor must be parameterless

```
namespace MyFirstApp
{
    2 references
    public static class Employee
    {
        2 references
        public static string Name { get; set; }
        0 references
        static Employee()
        {
            Name = "Mohan";
        }
    }

    0 references
    public class program
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine(Employee.Name);
            Console.ReadLine();
        }
    }
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Mohan
```

Q58) What is Static in c#?

Ans:

* The static keyword is used to specify a static member and function.
* static members are common to all the objects and it is not get tied to a specific object.
* static keyword can be used with classes, fields, methods, properties, operators, events, and constructors
* static can not be used with indexers, destructors, or types other than classes.

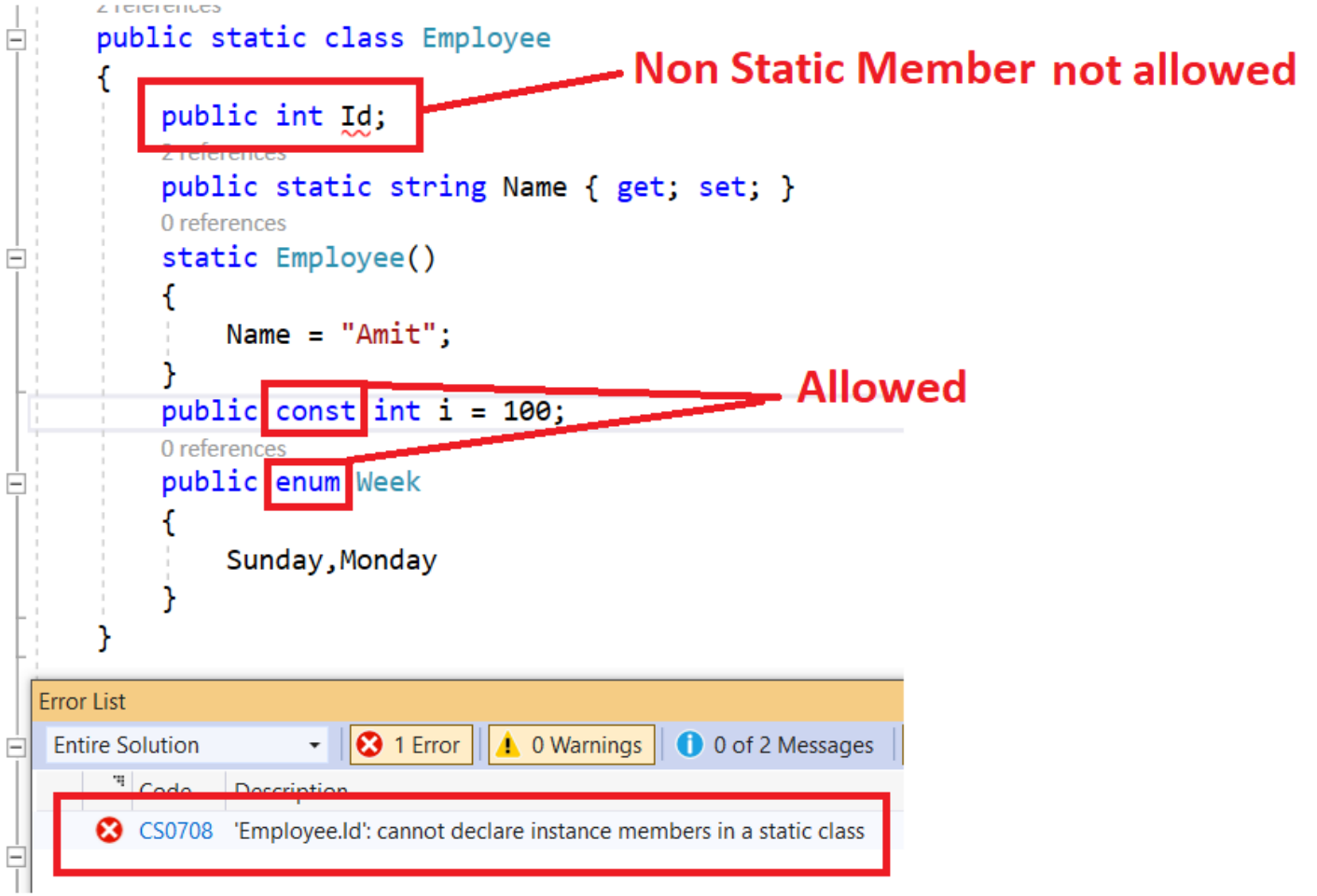
```
2 references
public static class Employee
{
    public static int Id;
    2 references
    public static string Name { get; set; }
    0 references
    static Employee()
    {
        Name = "Amit";
    }
}

0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Console.WriteLine(Employee.Name);
        Console.ReadLine();
```

Q59) Can a static class contains non-static members ?

Ans:

No, a static class cannot contain non-static members except const and enum.



Q60) Can a static class implement the interfaces ?

Ans:

No, a static classes cannot implement the interface.

```
interface IEmployee
{
    0 references
    void Display();
}
2 references
public static class Employee : IEmployee
{
    2 references
    public static string Name { get; set; }
    0 references
    static Employee()
    {
        Name = "Amit";
    }
```

Error List

Entire Solution | 2 Errors | 0 Warnings | 0 of 2 Messages

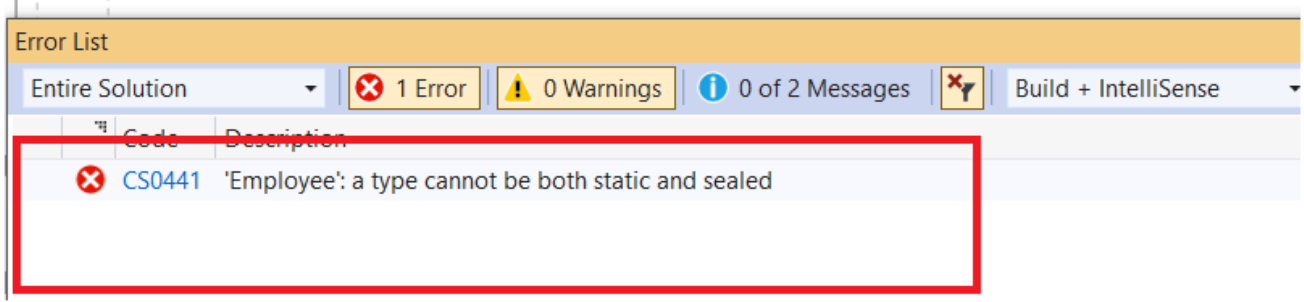
Code	Description
CS0714	'Employee': static classes cannot implement interfaces
CS0535	'Employee' does not implement interface member 'IEmployee.Display()'

Q61) Can we marked a static class as sealed in c# ?

Ans: No

a static class cannot be marked as sealed explicitly , because by default static classes are sealed.

```
public sealed static class Employee
{
    2 references
    public static string Name { get; set; }
    0 references
    static Employee()
    {
        Name = "Amit";
    }
```



Q62) Can we have destructor in static class ?

Ans:

No, a static class can't contain any destructor.

```
3 references
public static class Employee
{
    2 references
    public static string Name { get; set; }
    0 references
    static Employee()
    {
        Name = "";
    }
    0 references
    ~Employee()
    {
        Console.WriteLine("Destructor Invoked");
    }
}
```

Not Allowed

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 2 Messages | Build + IntelliSe

Code	Description
CS0711	Static classes cannot contain destructors

Q63) Can we pass ref or out parameter to a static method ?

Ans:

Yes, you can pass both ref and out type parameters in a static method call.

```
public class program
{
    0 references
    static void Main(string[] args)
    {
        // Assign string value
        string str = "Sohan";

        // Pass as a reference parameter
        SetName(ref str);

        // Display the given string
        Console.WriteLine(str);

        Console.ReadLine();
    }

    1 reference
    static void SetName(ref string str)
    {
        if (str == "Sohan")
        {
            Console.WriteLine("Hello Sohan");
        }
        str = "Sohan Mishra";
    }
```

Q64) Can a non-static class contain a static constructor ?

Ans:

Yes, a non-static class can also contain a static constructor.

```
1 reference
public class program
{
    0 references
    static program()
    {
        Console.WriteLine("Static Constructor in non static class");
    }
    0 references
    static void Main(string[] args)
    {

        Console.ReadLine();
    }

}
}
```

Q65) Can a static class contains instance constructors?

Ans:

No, a static class cannot contain any instance constructors.

```
1 reference
public static class Employee
{
    1 reference
    public static string Name { get; set; }
    0 references
    Employee()
    {
        Name = "";
    }
}
0 references
public class program
{

    0 references
    static void Main(string[] args)
    {
        Console.ReadLine();
    }
```

Instance Constructor Not Allowed

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 2 Messages | Build + IntelliSense

Code	Description
CS0710	Static classes cannot have instance constructors

Q66) Can a Method return multiple values at a time in c# ?

Ans:

Yes, a method can return multiple values at a time in C# by using below options:

1) Class or Struct.
2) Tuple.
3) KeyValuePair.
4) Ref or Out parameters.

Q67) What is Operator Overloading in C#?

Ans:

Operator overloading provides the ability to use the same operator for various operations.

It provides additional capabilities to C# operators when they are applied to user-defined data types.

Similar to any other function, an overloaded operator has a return type and a parameter list.

+, -, *, /, %, &, |, <<, >> (binary operators can be overloaded.)
+, -, !, ~, ++, --, true, false (Unary operators can be overloaded.)
==, !=, <, >, <= , >= (All relational operators can be overloaded)

&&, ||, [] (Array index operator), () (Conversion operator) -- (can't be overloaded)

=, . , ?:, ->, new, is, as, sizeof -- (can't be overloaded)

Q68) Is it possible to restrict object creation in C# ?

Ans:

Yes, we can restrict object creation in C# by using below options,

1) Static Class
2) Abstract Class
3) Private or Protected Constructor

```
2 references
public static class Employee
{
    0 references
    public static string Name { get; set; }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee();
        Console.ReadLine();
    }
```

Error List

Entire Solution | 2 Errors | 0 Warnings | 0 of 3 Message

Code	Description
CS0723	Cannot declare a variable of static type 'Employee'
CS0712	Cannot create an instance of the static class 'Employee'

```
2 references
public abstract class Employee
{
    0 references
    public static string Name { get; set; }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee();
        Console.ReadLine();
    }
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 3 Messages

Code	Description
CS0144	Cannot create an instance of the abstract type or interface 'Employee'

```
3 references
public class Employee
{
    0 references
    public static string Name { get; set; }
    1 reference
    private Employee()
    {
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee();
        Console.ReadLine();
    }
```

Error List

Entire Solution | 1 Error | 0 Warnings | 0 of 3 Messages

Code	Description
CS0122	'Employee.Employee()' is inaccessible due to its protection level

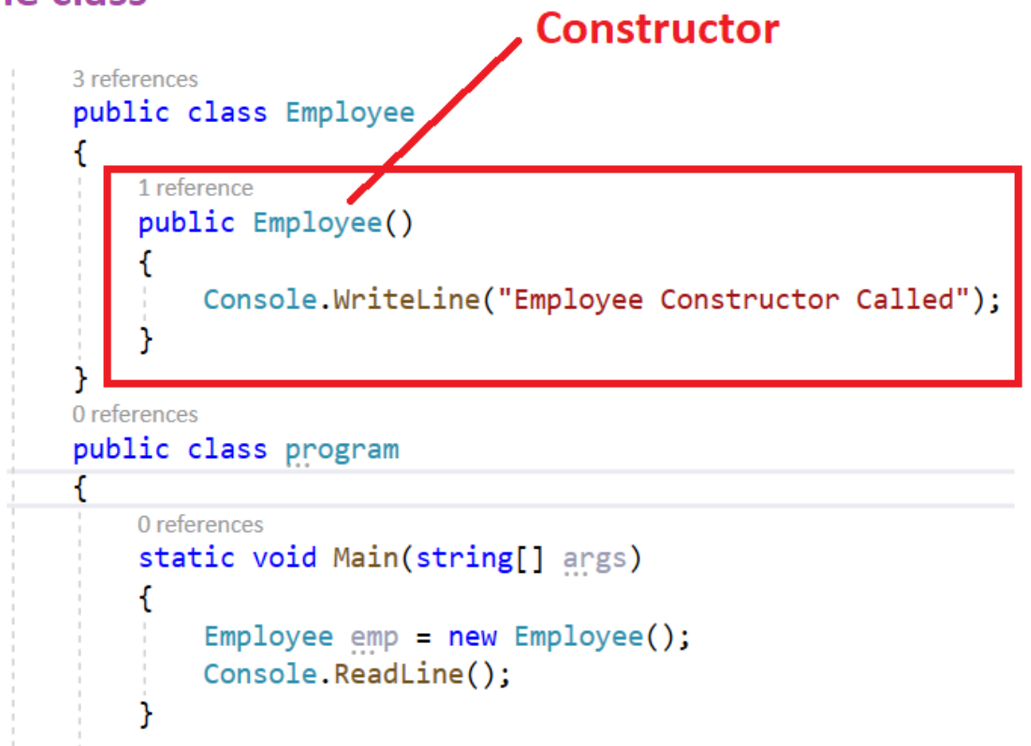
Q69) What is a constructor in C#?

Ans:

Constructor is the special method of the class that is automatically invoked during instance creation.

The main purpose of constructors is to initialize the private fields of the class during instance creation of the class.

1) The constructor name should be the same as the class name.
2) A class can have any number of constructors.
3) A constructor doesn't have any return type, not even void.
4) The constructor should not contain modifiers.
5) A static constructor can not be a parametrized constructor.
6) We can create only one static constructor of a class.
7) We can throw an exception from the constructor.
8) Constructor can have all five access modifiers.



Q70) What are the different types of constructors in C# ?

Ans:

There are five types of constructors available in C#:

1) Default Constructor.
2) Parameterized Constructor.
3) Copy Constructor.
4) Private Constructor.
5) Static Constructor.

```
public class Employee
{
    1 reference
    public Employee()
    {
        Console.WriteLine("Employee Constructor Called");
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee();
        Console.ReadLine();
    }

}
```

Q71) What is Default Constructor in C# ?

Ans:

* A constructor without any parameters is called a default constructor.
* Default constructor is divided into two types.

1) System-defined: If you not defined any constructor explicitly in your program, then system will provide for you at compilation time.

2) User-defined: This constructor is defined by the user.

Notes:

The default constructor initializes:

1) All numeric fields in the class to zero.
2) All string and object fields to null.

System-defined

```
2 references
class Employee
{
    int id;
    String name;
    1 reference
    public void Display()
    {
        Console.WriteLine("Employee ID:   " + id);
        Console.WriteLine("Employee Name:   " + name);
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee();
        emp.Display();
        Console.ReadLine();
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcorea
Employee ID:  0
Employee Name:
```

User Defined

```
3 references
class Employee
{
    int id;
    String name;

    1 reference
    public Employee()
    {
        id = 100;
        name = "Sujit";
    }
    1 reference
    public void Display()
    {
        Console.WriteLine("Employee ID:   " + id);
        Console.WriteLine("Employee Name:   " + name);
    }
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Employee ID:  100
Employee Name:  Sujit
```

Q72) What is Parameterized Constructor in C# ?

Ans:

* A constructor with at least one parameter is called a parameterized constructor.
* We can initialize each instance of the class with a different value.
* Within a class, we can have any number of constructors.
* Each and every constructor must have a different signature.
* The number, type, and parameter order should be different.
* We can define one default and any number of parameterized constructors.

```
class Employee
{
    int id;
    String name;

    2 references
    public Employee(int _id , string _name)
    {
        id = _id;
        name = _name;
    }
    2 references
    public void Display()
    {
        Console.WriteLine("Employee ID:   " + id);
        Console.WriteLine("Employee Name:   " + name)
    }
}
0 references
```

```
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee(200,"Amit");
        Employee emp2 = new Employee(300, "Sujit");
        emp.Display();
        emp2.Display();
        Console.ReadLine();
    }
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Employee ID:  200
Employee Name:  Amit
Employee ID:  300
Employee Name:  Sujit
```

Q73) What is Copy Constructor in C# ?

Ans:

*** In copy constructor, creates an object by copying variables from another object.**

*** In simple word, it is used to copy one object's data into another object.**

*** The main advantage of the copy constructor is to initialize a new object with the values of an existing object.**

```
7 references
class Employee
{
    int id;
    String name;
    1 reference
    public Employee(Employee emp)   // Copy constructor.
    {
        id = emp.id;
        name = emp.name;
    }
    1 reference
    public Employee(int _id , string _name) // Instance  constructor.
    {
        id = _id;
        name = _name;
    }
    2 references
    public void Display()
    {
        Console.WriteLine("Employee ID:  " + id);
        Console.WriteLine("Employee Name:  " + name);
    }
    2 references
    public void Display()
    {
        Console.WriteLine("Employee ID:  " + id);
        Console.WriteLine("Employee Name:  " + name);
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee(200,"Amit");
        Employee emp2 = new Employee(emp);
        emp.Display();
        emp2.Display();
        Console.ReadLine();
    }
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Employee ID:  200
Employee Name:  Amit
Employee ID:  200
Employee Name:  Amit
```

Q74) What is Static Constructor in C# ?

Ans:

* Static constructor is created using a static keyword.
*It will be invoked only once(during the first instance creation) or the first reference to a static member in the class.
* Static constructor is used to initialize static fields of the class and to write the code that required to be executed only once.

Keys Points:

1) There can be only one static constructor in a class.
2) The static constructor is parameterless.
3) Static constructor can only access the static members of the class.
4) No access modifier required in the static constructor definition.
5) We cannot create the object of the static class.

```
3 references
public class Employee
{
    static int id;
    0 references
    static Employee()
    {
        id = 500;
        Console.WriteLine("Static Constructor Called " + id);
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee();
        Console.ReadLine();
    }
}
```

When we create object of class, its static constructor invoked first

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Static Constructor Called 500
```

Q75) What is Private Constructor in C# ?

Ans:

***Private constructor is created using private specifier.**

*** If a class contains private constructor then we cannot create an object of the class outside of the class.**

*** We can create object within the same class which have private constructor.**

*** Private constructor is used in Singleton design pattern.**

```
public class Employee
{
    private static Employee emp = null;
    1 reference
    public static Employee GetInstance()
    {
        if(emp == null)
        {
            emp = new Employee();
        }
        return emp;
    }

    1 reference
    public void Display(string name)
    {
        Console.WriteLine(name);
    }
    1 reference
    private Employee()
    {
    }
```

```
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = Employee.GetInstance();
        emp.Display("amit");
        Console.ReadLine();
    }

}
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
amit
```

Private constructor prevent object creation outside of it.

Q76) What is constructor overloading? ?

Ans:

Constructor overoading is quite similar to the Method Overloading.
It is the ability to redefine a Constructor in more than one form.

We can overload constructors in following ways:

1) Using different type of arguments
2) Using different number of arguments
3) Using different order of arguments

```
        public Employee()
        {
            Console.WriteLine("Default constructor");
        }
        1 reference
        public Employee(int x)
        {
            Console.WriteLine("Constructor with one parameter");
        }
        1 reference
        public Employee(int x, int y)
        {
            Console.WriteLine("Constructor with two parameters");
        }
    }
    0 references
    public class program
    {
        0 references
        static void Main(string[] args)
        {
            Employee emp = new Employee();
            Employee emp2 = new Employee(200);
            Employee emp3 = new Employee(400,500);
            Console.ReadLine();
        }
    }
```

Invocation.....

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Default constructor
Constructor with one parameter
Constructor with two parameters
```

Q77) How many constructors can be defined in a class ?

Ans:

We can define any number of constructors. But each and every constructor must have a different signature.

Signature means the number, type, and parameter order should be different.

We can overload constructors in following ways:

1) Using different type of arguments
2) Using different number of arguments
3) Using different order of arguments

```
        public Employee()
        {
            Console.WriteLine("Default constructor");
        }
        1 reference
        public Employee(int x)
        {
            Console.WriteLine("Constructor with one parameter");
        }
        1 reference
        public Employee(int x, int y)
        {
            Console.WriteLine("Constructor with two parameters");
        }
    }
    0 references
    public class program
    {
        0 references
        static void Main(string[] args)
        {
            Employee emp = new Employee();
            Employee emp2 = new Employee(200);
            Employee emp3 = new Employee(400,500);
            Console.ReadLine();
        }
    }
```

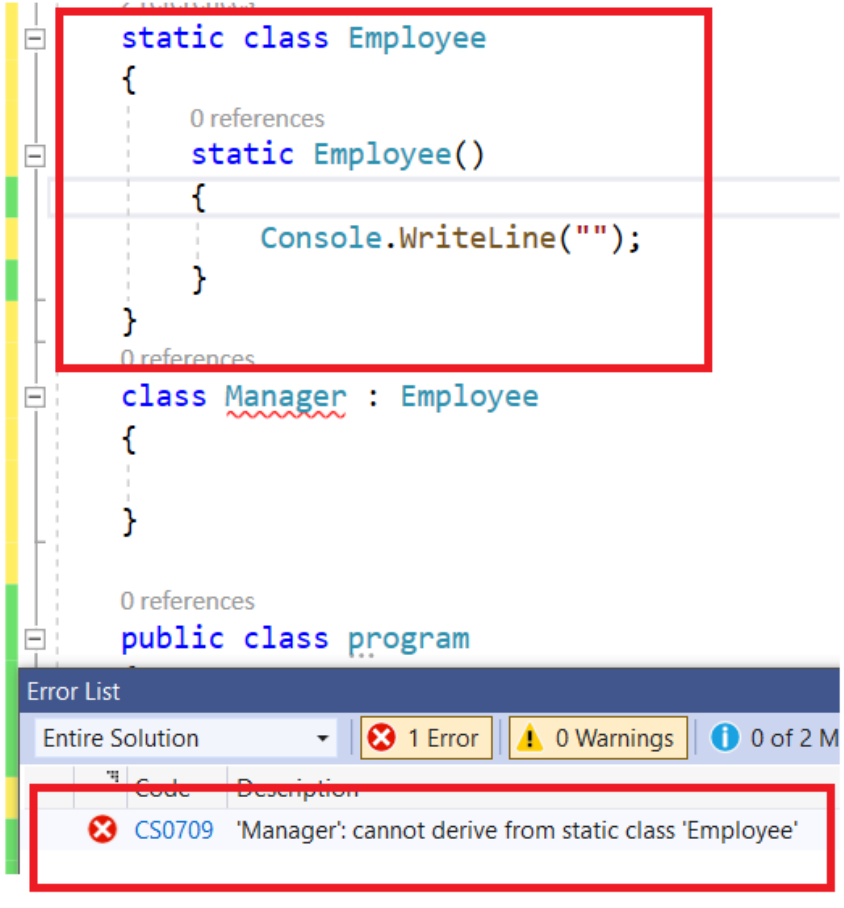
Invocation.....

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Default constructor
Constructor with one parameter
Constructor with two parameters
```

Q78) Can a static class inherit from another class ?

Ans:

No, a static class cannot inherit from any other classes except Object class.



Q79) Can a static class contains any private constructor ?

Ans:

No, we cannot define any explicit private constructor in a static class. we can only define a static constructor without any access modifier and parameter.

```
1 reference
static class Employee
{
    0 references
    private Employee()
    {
        Console.WriteLine("");
    }
}

0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
```

Error List

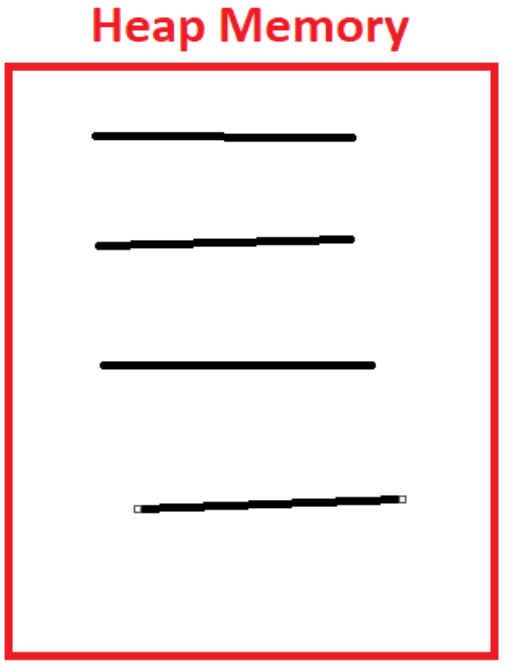
Entire Solution | 1 Error | 0 Warnings | 0 of

Code	Description
CS0710	Static classes cannot have instance constructors

Q80) Where static classes and members are stored in memory ?

Ans:

The static classes and members are stored in a special memory area called High Frequency Heap.



Q81) Static methods overloading is allowed or not in c# ?

Ans:

Yes, a static method can be overloaded but cannot be overridden.

```
public class Employee
{
    1 reference
    public static void Display()
    {
        Console.WriteLine("default static method");
    }
    1 reference
    public  static void Display(string name)
    {
        Console.WriteLine("Static method with parameter");
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Employee.Display();
        Employee.Display("name");
        Console.ReadLine();
    }

}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
default static method
Static method with parameter
```

Q82) Can a static constructor be private ?

Ans:

No, static constructors are private implicitly, so we cannot be mark it as private explicitly.

```
public class Employee
{
    0 references
    private static Employee()
    {

    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
```

Error List

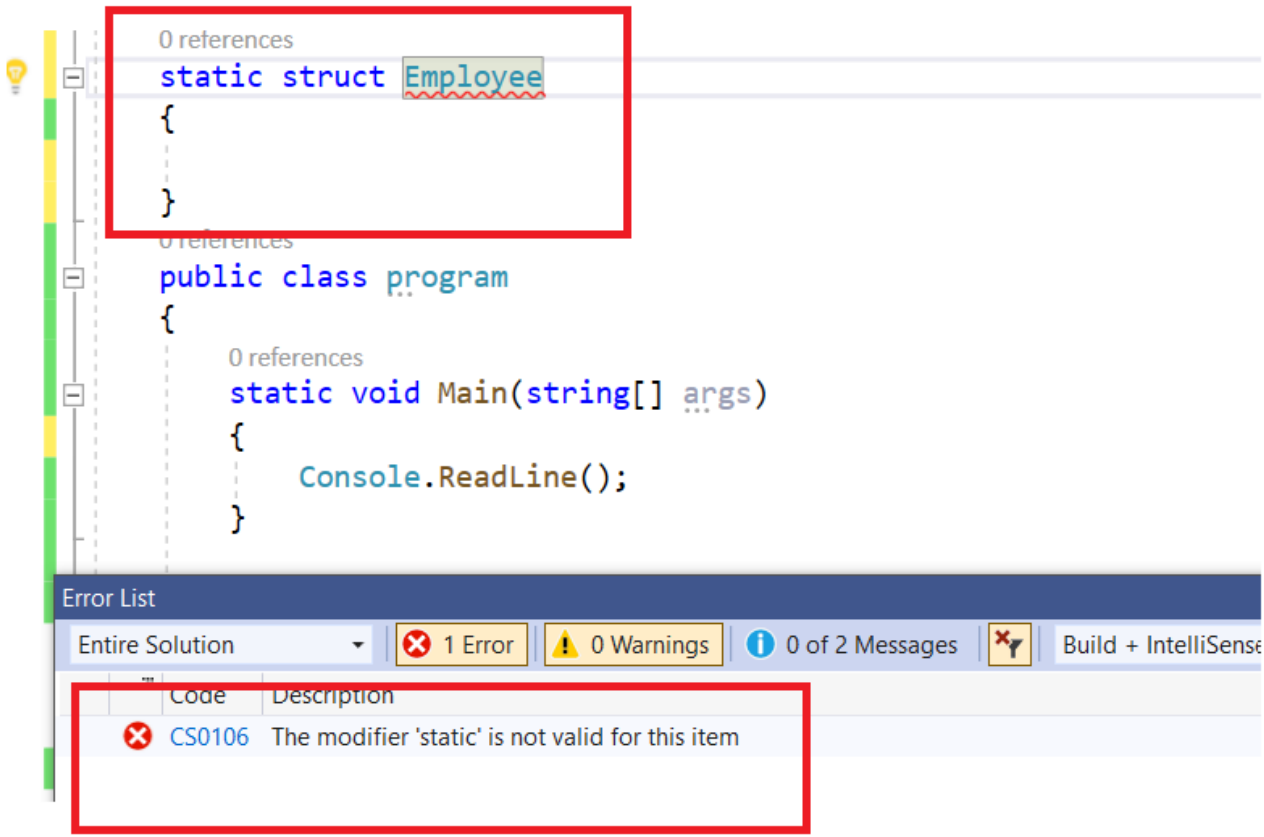
Entire Solution | 1 Error | 0 Warnings | 0 of 2 Messages | Build + Intelli

	Code	Description
⊗	CS0515	'Employee.Employee()': access modifiers are not allowed on static constructors

Q83) Can we defined a structure as static ?

Ans:

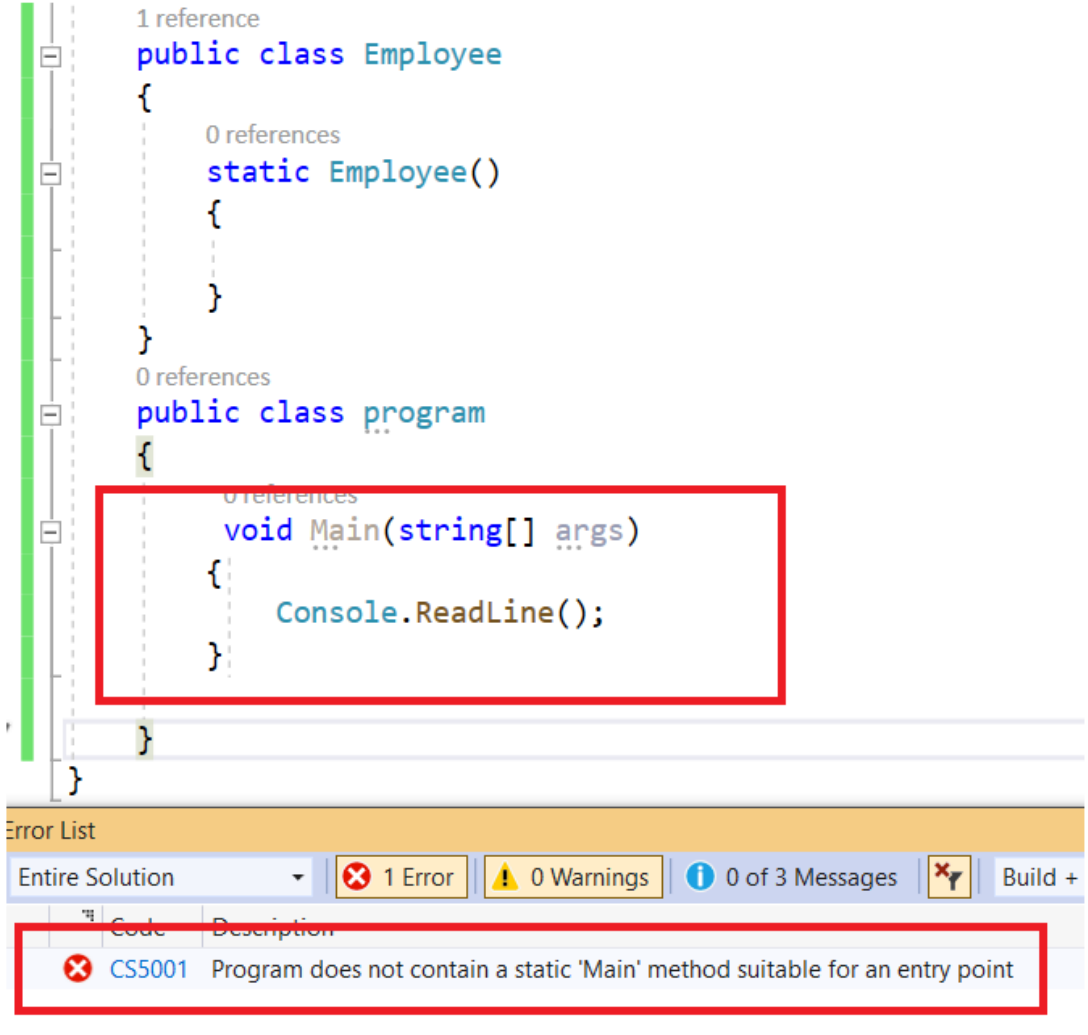
No, a struct cannot be defined as static.



Q84) Main() method can be non-static in C # ?

Ans:

No, the static Main() method cannot be non-static. Because the static main method is used as the entry point of the application.



Q85) Can a static method declare local static variables ?

Ans:

No, a static method cannot declare local static variables.

```
public class Employee
{
    static int a;
    0 references
    static void Display()
    {
        a = 500;
       static  int b;
        Console.WriteLine("Static method");
    }
}
0 references
public class program
{
    0 references
    static void Main(string[] args)
    {
        Console.ReadLine();
```

Error List

Entire Solution | 1 Error | 2 Warnings | 0 of 2 Messages | Build + IntelliS

	Code	Description
⊗	CS0106	The modifier 'static' is not valid for this item
⚠	CS0169	The field 'Employee.a' is never used
⚠	CS0168	The variable 'b' is declared but never used

Q86) What is the difference between is and as operators in c#?

IS

1. The "is" operator is used to check whether the run-time type of an object is compatible with a given type or not.

```
if(obj is classA)
{
   Console.WriteLine("obj is ClassA");
}
else if(obj is classA)
{
   Console.WriteLine("obj is ClassB");
}
```

2. The is operator returns false if the given object is not of the same type.

3. "is" operator returns true if the given object is of the same type

AS

1. The "as" operator is used to perform conversions between compatible types.

```
object s = "test";
string str=string.Empty;
if( s is string)
   str = s as string;
```

2. "as" operator return null if the conversion is not possible

3. "as" operator returns the object when they are compatible with the given type.

Q87) What is the difference between a struct and a class in c# ?

CLASS	STRUCT
1. Classes are reference types, and allocated on the heap and garbage-collected.	1. Structs are value types and allocated on the stack.
2. Class can contain constructor or destructor.	2. Structure does not contain parameter less constructor or destructor but can contain Parameterized or static constructor.
3. A Class can inherit from another class.	3. A Struct is not allowed to inherit from another struct or class.
4. Member function of the class can be virtual or abstract.	4. Member function of the struct cannot be virtual or abstract.
5. The data member of a class can be protected.	5. The data member of struct can't be protected.
6. Classes used new keyword for creating instances.	6. Struct can create an instance, without new keyword.

Q88) What are the nullable types in c# ?

1. The Nullable type allows you to assign a null value to a variable.
2. Nullable types introduced in C#2.0
3. Nullable types only work with Value Type, not with Reference Type.
4. The nullable types for Reference Type is introduced in C# 8.0.
5. Syntax : datatype? variable name = null;
6. Nullable type concept is not compatible with "var".
   So we will have Compile Time error if we declare like: var? i = null;
7. Null coalescing operator in c# is represented as "??"

```
int? x = null;   int y = 100;
int? result;
result = x ?? y;
```

Null coalescing operator (??) assigned the value of "x" to" result " if "x" is null else it will assign the value of "y" to the "result".

Advantage of Nullable Types:

**The main use of nullable type is in database applications.

**Nullable type is also useful to represent undefined value.

**We can use Nullable type instead of a reference type to store a null value.

Q89) Can multiple catch blocks be executed in a C# ?

Ans: No, we can use multiple catch blocks but at time only one will execute.

```
try
{
}
catch (Exception e1)
{
   // Block 1
}
catch (IOException e2)
{
   // Block 2
}
```

Q90) Can we have try block without a catch block in c# ?

Ans: Yes, we can have try block without a catch block.

```
try
{
Console.Write("Try block executed");
}
finally
{
 Console.Write("Finally block executed");
}
```

Q91) Difference between value type and reference type in c# ?

Value Type

1. When we passed as value type then new copy is created and passed, so changes to variable does not reflected back.

2. Value types are derived from System.ValueType.

3. Value types are faster in access.

4. These are Value types:
bool, byte, char, decimal, double, enum, float, int long, sbyte, short, struct, uint, ulong, ushort struct, enum

5. Data stored in stack memory

Reference Type

1. When we passed as Reference type then reference of that variable is passed, so changes to variable reflected back to caller.

2. Reference types are derived from System.Object

3. Reference types are slower in access compared to value types.

4. These are reference types:
String, Arrays , Class, Delegate, interface,

5. Data stored in heap memory

Q92) What is Garbage Collection in C# ?

Ans:

Garbage collector manages the allocation and reclaiming of memory.

GC (Garbage collector) visits the heap time to time and collects objects that are no longer used by the application and then makes them free from memory.

GC (Garbage collector) works on managed heap, which is a block of memory for storing objects.

Heap is managed by three Generations:

1) 0 Generation : It holds short-lived objects, i.e., Temporary objects. GC initiates garbage collection process frequently in this generation.

2) 1 Generation : It is the buffer between short lived and long lived objects.

3) 2 Generation : It holds long lived objects i.e static and global variable that needs to be store for a longer period of time.

When GC(Garbage collector) gets triggered?

GC automatically start operation so there is no specific time for GC to get triggered.

1) When virtual memory is running out of space.

2) When you explicitly call GC.Collect() method.

3) When allocated memory is suppressed acceptable threshold.

Q93) What is Boxing and Unboxing in C#?

Ans:

Boxing

Boxing is the process of Converting a Value Type (int,char,bool etc.) to a Reference Type(object).
Boxing is implicit conversion process.
Value type is stored in Stack and Reference Type is stored in Heap.

```
static void Main(string[] args)
{
    // assigned int value
    int result = 9000;

    // boxing
    object obj = result;

    // value of result changing
    result = 2000;

    System.Console.WriteLine("Value type value " + result);
    System.Console.WriteLine("Reference type " + obj);

    Console.ReadLine();
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Value type value 2000
Reference type 9000
```

Unboxing

Unboxing is the process of converting reference type to the value type.
It is explicit conversion process.

```
static void Main(string[] args)
{
    // assigned int value
    int result = 9000;

    // boxing
    object obj = result;

    // unboxing result value from obj
    int finalResult = (int)obj;

    System.Console.WriteLine("Final Result value " + finalResult);
    System.Console.WriteLine("Obj value " + obj);

    Console.ReadLine();
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
Final Result value 9000
Obj value 9000
```

Q94) What is the use of Params keyword in C# ?

Ans:

* The "params" keyword in C# allows a method to accept a variable number of arguments.
* C# params works as an array of objects.
* We can pass 'n' number of arguments by using params keyword.
* Params parameter will always be last parameter if we pass other parameters also.

E.G: //Params should be the last parameter

public int Display(params int[] nums, string name) // Invalid

public int Display(string name,params int[] nums) // Valid

* "params" parameter should be one-dimensional array otherwise compile time error will show.

When to use params:

1) "Params" parameter keyword is useful when we don't know in advance how many parameters we are expecting
2) "Params" are also useful to write "clean code".

params Example

```
1 reference
static void ParamsExample(int x, params object[] items)
{
    Console.WriteLine(x);
    if (items != null)
    {
        foreach (var i in items)
        {
            Console.WriteLine(i);
        }
    }
}

0 references
static void Main(string[] args)
{
    ParamsExample(500,16,38, "Provider");
    Console.ReadLine();
}
```

```
D:\projects\programs\MyFirstApp\bin\Debug\netcoreapp3.1\MyFirstApp.exe
500
16
38
Provider
```

Q95) Differences between “continue” and “break” statements in C# ?

Break

1) It stops the execution of remaining iteration of the loop.

2) It resumes the control of the program from the end of loop enclosing the 'break' statement.

3) 'break' statement can be used in switch.

4) It causes early termination of loop.

```
0 references
static void Main(string[] args)
{

    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine(i);
        if (i == 5)
            break;
    }

    Console.ReadLine();
}
```

```
0
1
2
3
4
5
```

Control jump out of the "for loop" after break statement

Continue

1) It terminates only the current iteration of the loop, from next iteration program execution will continue.

2) It resumes the control of the program to the next iteration of the loop enclosing 'continue'.

3) 'continue' can not be executed with switch.

4) It causes early execution of the next iteration.

```
static void Main(string[] args)
{

    for (int i = 0; i < 10; i++)
    {

        if (i == 5)
            continue;
        Console.WriteLine(i);
    }

    Console.ReadLine();
}
```

```
0
1
2
3
4
6
7
8
9
```

Control only skip i=5 value and continue from i=6.....

Q96) What are the Literals in c# ?

Ans:
Literals in C# are the fixed value used by a variable. Literals are predefined and it cannot be modified during the code execution.

Literal is a value that is used by the variables.

```
// below 50 is a constant/literal.
int result = 50;
```

Literals can be of the following types:
1) Integer Literals
2) Character Literals
3) String Literals
4) Floating-point Literals
5) Boolean Literals
6) Null Literals

5) Floating-point
Literals:Double dVal = 7.165

1) Integer Literals:

```
int a = 200;  // decimal type
int b = 061;  // octal type
int c = 0x253f;   // hexadecimal
type
```

2. String Literals

```
string str =  “Literals”;   // string literals
```

3) Character Literals
For character data types we can specify character literals in three ways:
i)Single quote: char ch1 = 'p';
ii)Unicode Representation:

```
char ch2 = '\u0061'; // /u0061 represent a.
```

iii) Escape Sequence: char ch3 = '\n';

4) Boolean Literals: bool isTrue = true;

```
bool isTrue  = false;
```

Q97) What are the different Data Types in C# ?

*Data types specify the type of data that a variable can hold.

Data types in are divided into three categories:

1) Value Data Types:

* It will directly store the variable value in memory
* It can accept both signed and unsigned literals.

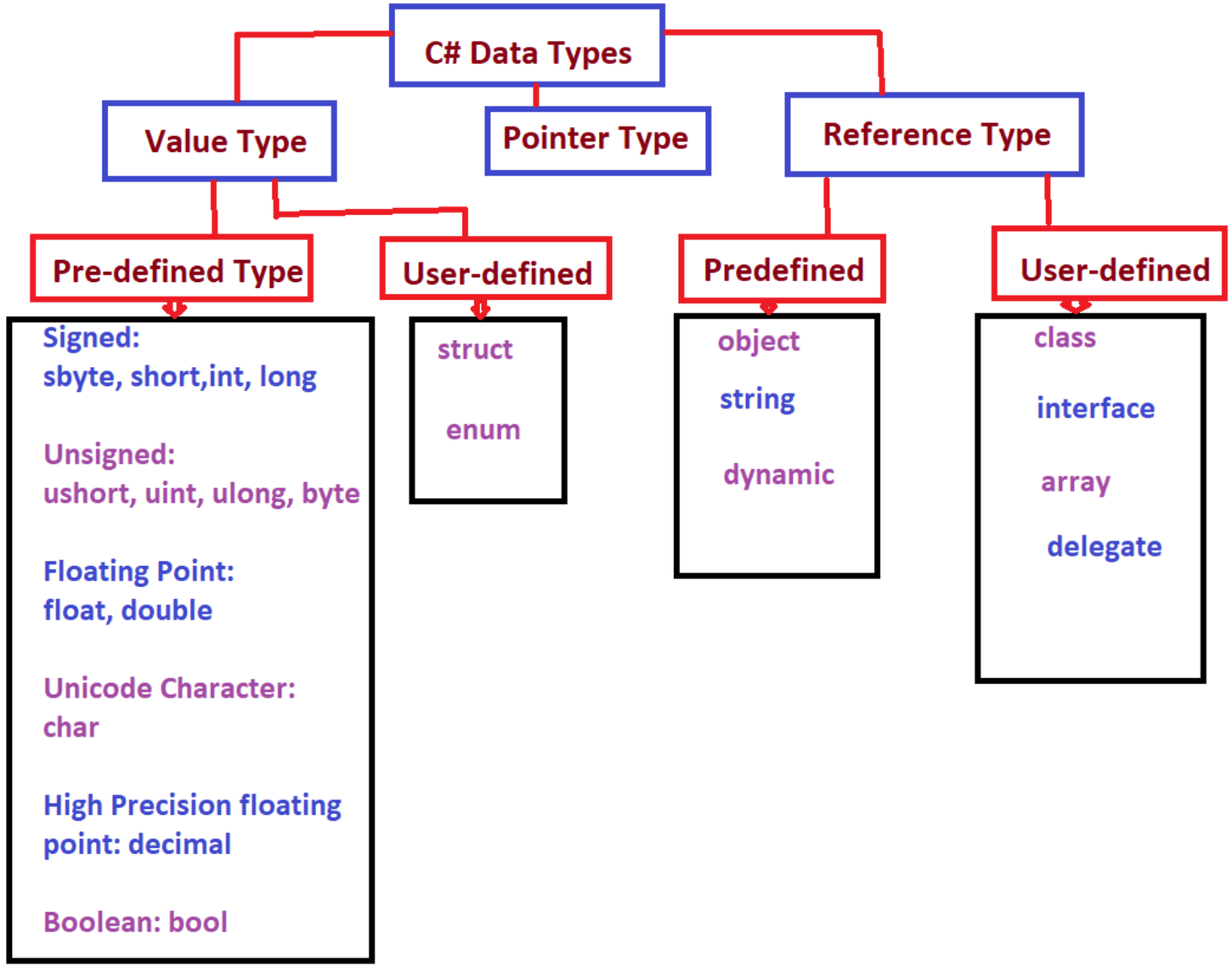
2) Reference Data Types:

* It will contain a memory address of variable value but won't store the variable value directly in memory.
* The built-in reference types are string, object.

3) Pointer Data Type

* It will contain a memory address of the variable value.

*We have a two symbols ampersand (&) and asterisk (*) to get the pointer details.



Q98) What are the different Variables in C# ?

Ans:

* Variable is a name that you give to the memory location.
* Variables are name given to data value.
* Variables can hold the value of particular data types, e.g int, string, float etc.....

Variable naming conventions:

It must be unique.

It can contain letters, digits and the underscore only.

It must start with a letter.

It is are case-sensitive.

It cannot contain reserved keywords.

Types of Variables in C#:

1) Local Variables
2) Instance Variables(Non Static)
3) Static Variables
4) Read-only Variables
5) Constant Variables

Q99) What are the keywords in C# ?

* The reserved words that have special meaning for the compiler are called "keywords".
* These keywords are reserved for the compiler.

Types of keywords

Modifier Keywords
- event
- abstract
- new
- async
- const
- extern
- override
- etc.....

Access Modifier Keywords
- public
- private
- internal
- protected

Method Parameter Keywords
- ref
- out
- params

Namespace Keywords
- using
- :: operator
- . operator
- extern

Statement Keywords
- if
- else
- for
- foreach
- do
- in
- while
- switch
- case
- break etc.....

Operator Keywords
- sizeof
- typeof
- as
- is
- new
- await

Type Keywords
- byte
- decimal
- double
- char
- class
- enum
- float etc.....

Contextual Keywords
- var
- dynamic
- global
- value
- etc.....

Query Keywords
- select
- group
- from
- where
- into
- orderby

Q100) What is Namespaces in C# ?

Ans:

* Namespaces are used to organize the classes.
* It also helps to control the scope of methods and classes.
* The beauty of namespace is that, the class declared in one namespace will not conflict with the same class names declared in another namespace.
* Inside one namespace, no two classes can have the same name.
* We can define a namespace into another namespace, called nested namespace.

* To reduce code redundancy we use Namespace in c#.

* Namespaces are used to arrange code into logical groups