

1)Write a program to demonstrate different type of constructors in C# language.

### Introduction:

A constructor in C# is a member of a class. It is a method in the class which gets executed when a class object is created. Usually, we put the initialization code in the constructor. The name of the constructor is always is the same name as the class. A C# constructor can be public or private. A class can have multiple overloaded constructors. C# supports overloading of constructors, that means we can have constructors with different set of parameters.

Constructor of a class must have the same name as the class name in which it resides. A constructor can not be abstract, final, and Synchronized. Within a class, you can create only one static constructor. A constructor doesn't have any return type, not even void. A static constructor cannot be a parameterized constructor. A class can have any number of constructors. Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

Types of Constructor are: Default Constructor, Parameterized Constructor, Copy Constructor, Private Constructor, Static Constructor

### Syntax:

```
Class ClassName{  
  
ClassName() { } //default constructor  
  
}
```

### Code:

```
using System;  
public class Employee  
{  
    public Employee() //Default Constructor  
    {  
        Console.WriteLine("Default Constructor Invoked");  
    }  
  
    private int sum;  
    public Employee(int a, int b) //paramaterized constructor  
    {  
        sum = a + b;  
    }  
}
```

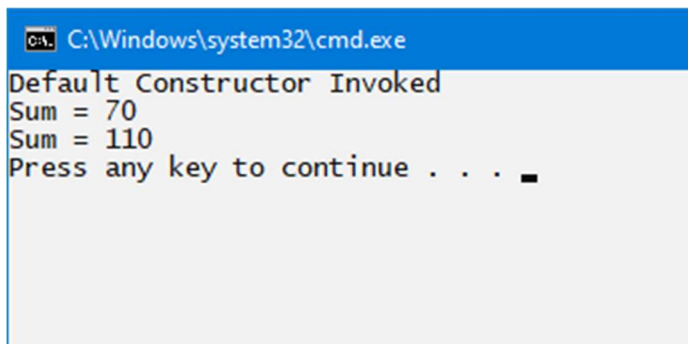
```

        Console.WriteLine("Sum = {0}", sum);
    }

    public Employee(int a, int b, int c) //overloaded constructor
    {
        sum = a + b;
        Console.WriteLine("Sum = {0}", sum);
    }
    static Employee()
    {
        Console.WriteLine("Inside static constructor.");
    }
}
class Program
{
    static void Main(string [] args)
    {
        Employee e1 = new Employee(); //both_constructor_called
        new Employee(30, 40); //constructor is called
        new Employee(50, 60, 70);
    }
}

```

Output:



```

C:\Windows\system32\cmd.exe
Default Constructor Invoked
Sum = 70
Sum = 110
Press any key to continue . . . 

```

2)Write a C# program to demonstrate multilevel and multiple inheritance.

(a) Multilevel inheritance:

Introduction:

In the Multilevel inheritance, a derived class will inherit a base class and as well as the derived class also act as the base class to other class. For example, three classes called A, B, and C, as shown in the below image, where class C is derived from class B and class B, is derived from class A. In this situation, each derived class inherit all the characteristics of its base classes. So class C inherits all the features of class A and B.

Syntax:

```
Class A{ }
```

```
Class B:A{ }
```

```
Class C:B{ }
```

Code:

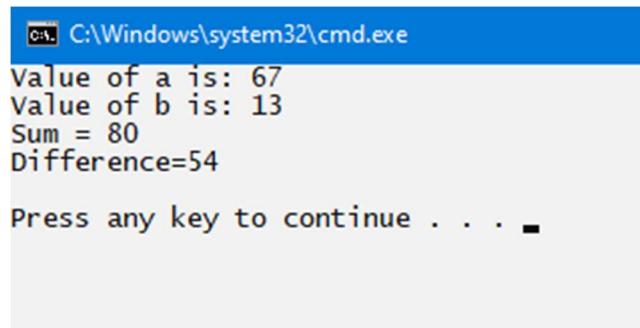
```
using System;
namespace ConsoleApp2
{
    class A{
        public int a, b, c;
        public void ReadData(int a, int b){
            this.a = a;
            this.b = b;
        }
        public void Display(){
            Console.WriteLine("Value of a is: " + a);
            Console.WriteLine("Value of b is: " + b);
        }
    }
    class B : A{
```

```

        public void Add(){
            base.c = base.a + base.b;
            Console.WriteLine("Sum = " + base.c);
        }
    }
    class C : B{
        public void Sub(){
            base.c = base.a - base.b;
            Console.WriteLine("Difference=" + base.c);
        }
    }
    class Level
    {
        static void Main()
        {
            C obj = new C();
            obj.ReadData(67, 13);
            obj.Display();
            obj.Add();
            obj.Sub();
            Console.ReadLine();
        }
    }
}

```

Output:



```

C:\Windows\system32\cmd.exe
Value of a is: 67
Value of b is: 13
Sum = 80
Difference=54
Press any key to continue . . . 

```

### (b) Multiple Inheritance by interface:

#### Introduction:

Like a class, Interface can have methods, properties, events, and indexers as its members. But interfaces will contain only the declaration of the members. The implementation of the interface's members will be given by class who implements the interface implicitly or explicitly. Multiple inheritance is possible with the help of Interfaces but not with classes.

#### Syntax:

```
interface <interface_name >
{
    // declare Events
    // declare indexers
    // declare methods
    // declare properties
}
class class_name : interface_name
```

#### Code:

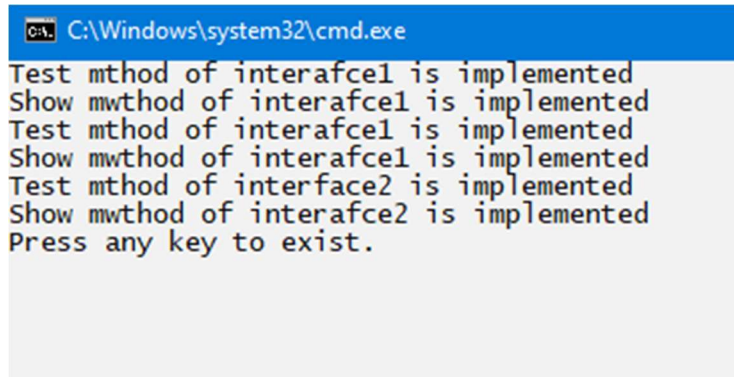
```
using System;
namespace ConsoleApp2
{
    public interface Interface1
    {
        void Test();
        void Show();
    }
    public interface Interface2
    {
        void Test();
        void Show();
    }
    class ImplementInterafce : Interface1, Interface2
    {
        //public modifier is not allowed
        void Interface1.Test()
        {
            Console.WriteLine("Test mthod of interafce1 is implemented");
        }
        void Interface1.Show()
        {
            Console.WriteLine("Show mwthod of interafce1 is implemented");
        }
    }
}
```

```

    }
    void Interface2.Test()
    {
        Console.WriteLine("Test mthod of interface2 is implemented");
    }
    void Interface2.Show()
    {
        Console.WriteLine("Show mwthod of interafce2 is implemented");
    }
}
class Program
{
    static void Main(string[] args)
    {
        ImplementInterafce obj = new ImplementInterafce();
        //obj.Test(); //not possible
        //obj.Show(); //not possible
        ((Interface1)obj).Test();
        ((Interface1)obj).Show();
        Interface1 obj1 = new ImplementInterafce();
        obj1.Test();
        obj1.Show();
        Interface2 obj2 = new ImplementInterafce();
        obj2.Test();
        obj2.Show();
        Console.WriteLine("Press any key to exist.");
        Console.ReadKey();
    }
}
}

```

Output:



```

C:\Windows\system32\cmd.exe
Test mthod of interafce1 is implemented
Show mwthod of interafce1 is implemented
Test mthod of interafce1 is implemented
Show mwthod of interafce1 is implemented
Test mthod of interface2 is implemented
Show mwthod of interafce2 is implemented
Press any key to exist.

```

3) Write a C# program to overload binary operators.

#### Introduction:

Operator overloading gives the ability to use the same operator to do various operations. Binary Operators will work with two Operands. Examples of binary operators include the Arithmetic Operators (+, -, \*, /, %), Arithmetic Assignment operators (+=, -=, \*=, /=, %=) and Relational Operators etc. Overloading a binary operator is similar to overloading a unary operator, except that a binary operator requires an additional parameter. Only the predefined set of C# operators can be overloaded. To make operations on a user-defined data type is not as simple as the operations on a built-in data type. To use operators with user-defined data types, they need to be overloaded according to a programmer's requirement. An operator can be overloaded by defining a function to it. The function of the operator is declared by using the operator keyword.

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

Operators may be considered as functions internal to the compiler.

#### Syntax:

```
operator operator (object1, object2);  
operator + (a, b);
```

#### Code:

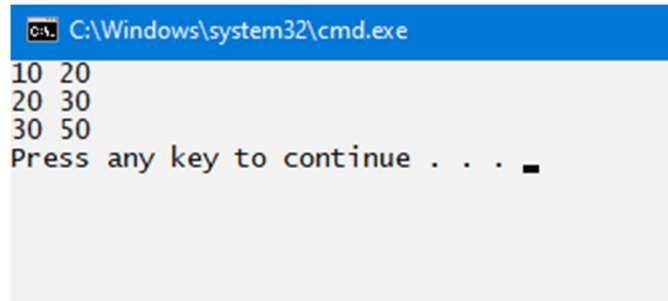
```
using System;  
class Complex  
{  
    private int x;  
    private int y;  
    public Complex()  
    {  
  
    }  
    public Complex(int i, int j)  
    {  
        x = i;  
        y = j;  
    }  
    public void ShowXY()  
    {  
        Console.WriteLine("{0} {1}", x, y);  
    }  
    public static Complex operator +(Complex c1, Complex c2)
```

```

{
    Complex temp = new Complex();
    temp.x = c1.x + c2.x;
    temp.y = c1.y + c2.y;
    return temp;
}
}
class MyClient
{
    public static void Main()
    {
        Complex c1 = new Complex(10, 20);
        c1.ShowXY(); // displays 10 & 20
        Complex c2 = new Complex(20, 30);
        c2.ShowXY(); // displays 20 & 30
        Complex c3 = new Complex();
        c3 = c1 + c2;
        c3.ShowXY(); // displays 30 & 50
    }
}

```

Output:



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\Windows\system32\cmd.exe". The command prompt area is white and shows the following output: "10 20", "20 30", and "30 50" on three separate lines. Below the output, the text "Press any key to continue . . ." is displayed, followed by a small black square cursor.

```

C:\Windows\system32\cmd.exe
10 20
20 30
30 50
Press any key to continue . . .

```



#### 4) Demonstrate dynamic binding in C# language.

##### Introduction:

Polymorphism provides a way for the derived class to give its own definition of a method that has already been defined by the base class. This process is called method overriding. In method overriding, the methods have same names, same signatures, same return types but are in different scopes (classes) in the hierarchy. C# uses virtual and override keywords to implement method overriding.

Binding is an association of function calls to an object. The binding of the function calls an object at the run time is called run time or dynamic or late binding. Late binding is achieved, using virtual methods, the virtual methods are overridden in the derived class as per the specific requirement. When there is a possibility that a method in the base class could be overridden in the derived class, this method is marked with the virtual keyword.

If the virtual method is required to have specific implementation in the derived class, it can be overridden, using override keyword. Using method overriding, we can resolve method invocation at the runtime. It is called late binding of behaviour of the object.

##### Syntax:

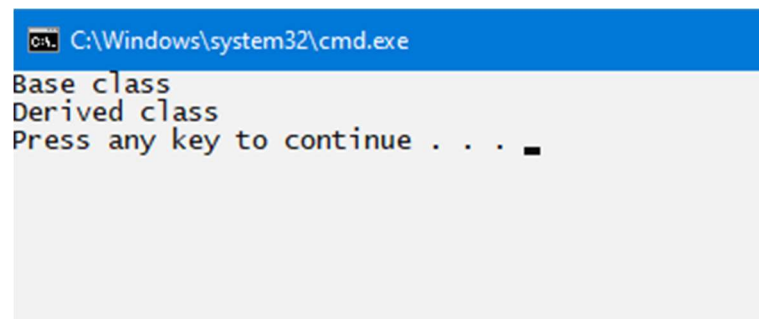
```
Class A{  
    public virtual Fun( )  
}  
Class B:A{  
    public override Fun( )  
}
```

##### Code:

```
using System;  
  
class baseClass  
{  
    public virtual void show()  
    {  
        Console.WriteLine("Base class");  
    }  
}  
class derived : baseClass
```

```
{
    public override void show()
    {
        Console.WriteLine("Derived class");
    }
}
class GFG
{
    public static void Main()
    {
        baseClass obj;
        obj = new baseClass();
        obj.show();
        obj = new derived();
        obj.show();
    }
}
```

Output:



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "0:1. C:\Windows\system32\cmd.exe". The command prompt area has a light gray background and displays the following text: "Base class", "Derived class", and "Press any key to continue . . . \_".

## 5) Write a C# Program to Implement Use of Indexers.

### Introduction:

An indexer allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a virtual array. You can then access the instance of this class using the array access operator ([ ]).

### Syntax:

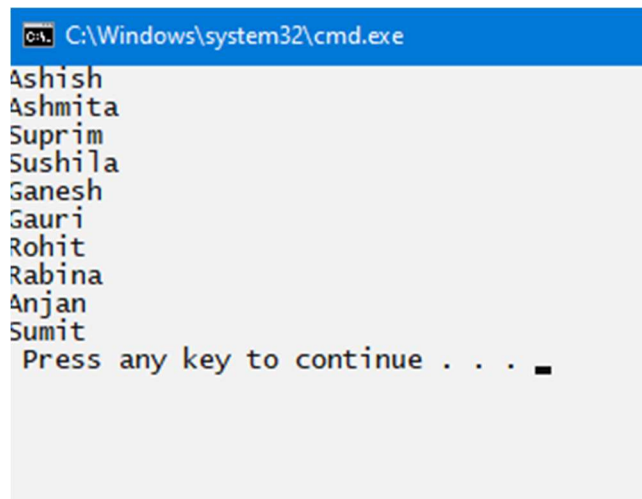
```
element-type this[int index] {  
  
    // The get accessor.  
    get {  
        // return the value specified by index  
    }  
  
    // The set accessor.  
    set {  
        // set the value specified by index  
    }  
}
```

### Code:

```
using System;  
namespace Indexer_example1{  
    class Program{  
        class IndexerClass{  
            private string[] names = new string[10];  
            public string this[int i]{  
                get{  
                    return names[i];  
                }  
                set{  
                    names[i] = value;  
                }  
            }  
        }  
    }  
}
```

```
}  
static void Main(string[] args){  
    IndexerClass Team = new IndexerClass();  
    Team[0] = "Ashish";  
    Team[1] = "Ashmita";  
    Team[2] = "Suprim";  
    Team[3] = "Sushila";  
    Team[4] = "Ganesh";  
    Team[5] = "Gauri";  
    Team[6] = "Rohit";  
    Team[7] = "Rabina";  
    Team[8] = "Anjan";  
    Team[9] = "Sumit";  
    for (int i = 0; i < 10; i++){  
        Console.WriteLine(Team[i]);  
    }  
    Console.ReadKey();  
}  
}
```

Output:



```
C:\Windows\system32\cmd.exe  
Ashish  
Ashmita  
Suprim  
Sushila  
Ganesh  
Gauri  
Rohit  
Rabina  
Anjan  
Sumit  
Press any key to continue . . .
```

6) Write a C# program to illustrate the concept of deconstruction with class.

#### Introduction:

Deconstruction occurs in an unconditional way just like a sequence of assignments. Pattern matching is similar, but in a more dynamic context; the input value has to match the pattern in order to execute the code that follows it. C# 7 introduces pattern matching in a couple of contexts and a few kinds of patterns, and there will likely be more in future releases. C# 7 provides two flavors of deconstruction: one for tuples and one for everything else. They follow the same syntax and have the same general features, but talking about them in the abstract can be confusing.

#### Syntax:

```
Class A{
    Public void A(){

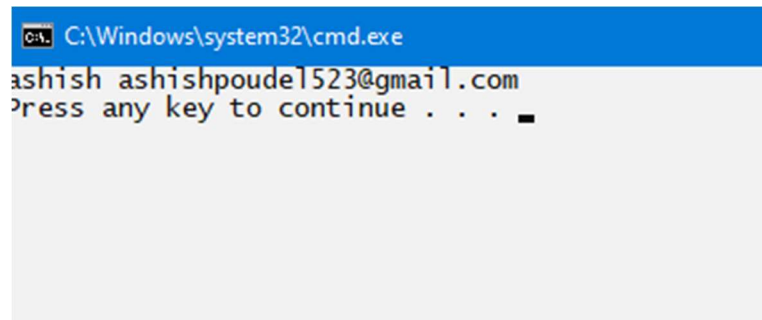
    }
    //creating deconstructor
    Public void A(out data_type identifier, out data_type identifier)
    {
    }
}
```

#### Code:

```
using System;
namespace CSharpFeatures
{
    public class Student
    {
        private string Name;
        private string Email;
        public Student(string name, string email)
        {
            this.Name = name;
            this.Email = email;
        }
        // creating deconstruct
        public void Deconstruct(out string name, out string email)
        {
            name = this.Name;
            email = this.Email;
        }
    }
}
```

```
class DeconstructExample
{
    static void Main(string[] args)
    {
        var student = new Student("ashish", "ashishpoude1523@gmail.com");
        var (name, email) = student;
        Console.WriteLine(name + " " + email);
    }
}
```

Output:



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\Windows\system32\cmd.exe". The command prompt shows the output "ashish ashishpoude1523@gmail.com" on the first line. The second line shows the prompt "Press any key to continue . . ." followed by a cursor.