

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)



Lt. Shree Chimanbhai Shukla

PGDCA – SEM-2 - C#.NET

Shree H.N.Shukla College,
Street No. 2, Vaishali Nagar,
Nr. Amrapali Railway Crossing,
Raiya Road, Rajkot.



Shree H.N.Shukla College,
Street No. 3, Vaishali Nagar,
Nr. Amrapali Railway Crossing,
Raiya Road, Rajkot.

Website: hnsgroupofcolleges.org
Email : hnsinfo@hnshukla.com

Subject: C#- PO

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

CS – 09: PROGRAMMING WITH C#		
Unit No.	Topics	Details
1	Introduction	Introduction to visual studio 2008 Visual studio editions Visual studio IDE
	C# Basics	<ul style="list-style-type: none">▪ Variables, Constants, Strings▪ Data types▪ Arrays▪ Decision statements▪ Loop statements▪ Exception using try-catch-finally▪ NameSpace▪ Class▪ Object▪ Struct
2	Inheritance	<ul style="list-style-type: none">▪ Inheriting a class▪ Sealed class▪ Overloading an operator▪ Overloading a method▪ Overloading an Indexer▪ Creating an Interface▪ Implementing an Interface▪ Inheriting an Interface
	Pointers and Delegates	<ul style="list-style-type: none">▪ Pointers▪ Pointers to Arrays▪ Pointers to Structures▪ Delegate▪ Declaring and Instantiating Delegate▪ Multicast delegate▪ Creating events▪ Chaining events▪ Firing an event

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

3	Threading in C#	<ul style="list-style-type: none">▪ Introduction▪ Difference between process and thread▪ The thread class▪ Multithreading▪ Thread Priorities▪ Thread Synchronization
	Collection and Generics	Understanding Collections: ArrayList, BitArray, HashTable, Queue, SortedList, Stack, Generics, Generic List, Generic Stack, Generic Queue, Generic HashSet
4	Reflection in C#	Reflection, Why we need Reflection?, Using Reflection, Dynamic loading and reflection
	Windows Forms and Control Programming	Windows Forms: MsgBox, DialogBox, Handling Mouse, Events, Handling Key Events Basic Control Programming For Following: Controls, Button, Label, TextBox, RichTextBox, RadioButton, CheckBox ListBox, CheckedListBox, ComboBox, ListView, TreeView, ImageList, PictureBox Panel, GroupBox, TabControl, ScrollBar ToolTip, NotifyIcon, Timer, ProgressBar
5	ADO.NET Programming	Architecture of ADO. NET Data providers in ADO.NET: Connection Command DataReader DataAdapter DataSet: DataTable DataView DataColumn DataRow DataRelation DataReader DataGridView Control Introduction to LINQ Using LINQ to Dataset Example

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Chapter-2 :- Inheritance

Topics	
Inheritance	<ul style="list-style-type: none">▪ Inheriting a class▪ Sealed class▪ Overloading an operator▪ Overloading a method▪ Overloading an Indexer▪ Creating an Interface▪ Implementing an Interface▪ Inheriting an Interface
Pointers and Delegates	<ul style="list-style-type: none">▪ Pointers▪ Pointers to Arrays▪ Pointers to Structures▪ Delegate▪ Declaring and Instantiating Delegate▪ Multicast delegate▪ Creating events▪ Chaining events▪ Firing an event

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

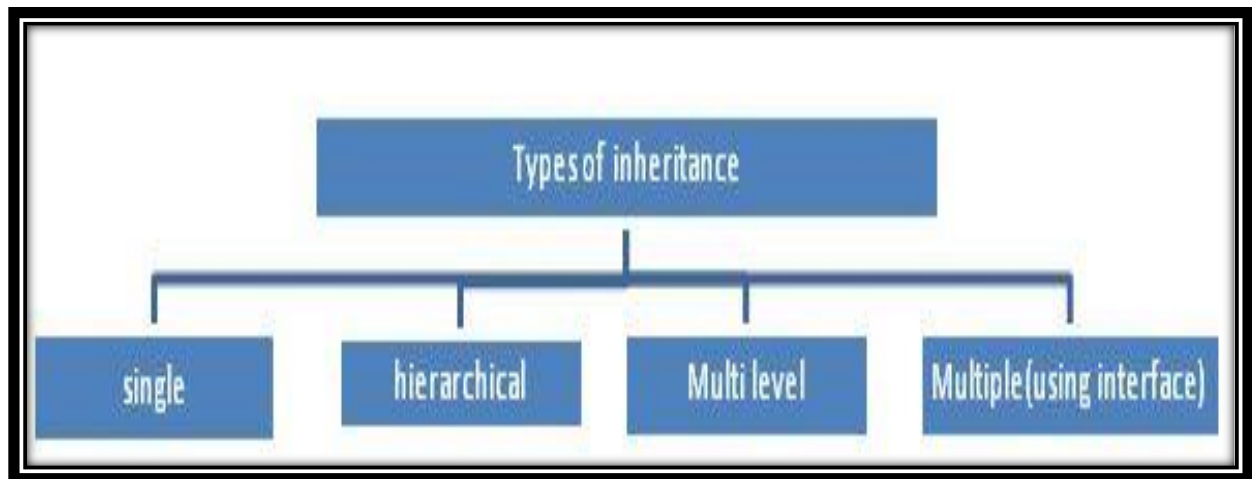
(Affiliated to Saurashtra University)

Topic: What is Inheritance?

Ans:

- ❖ In C#, inheritance is the process in which one object acquires all the properties and behaviors of its parent object automatically
- ❖ In C#, the class which inherits the members another class is known as derived class and the class whose members are inherited is known as base class
- ❖ The advantage of inheritance is code reusability (that is you can reuse the members of your parent class. So, there is no need to define the member again. So, less code is required in the class.
- ❖ Following are the types of inheritance:

NOTE: Multiple inheritance is not supported in C# through class.

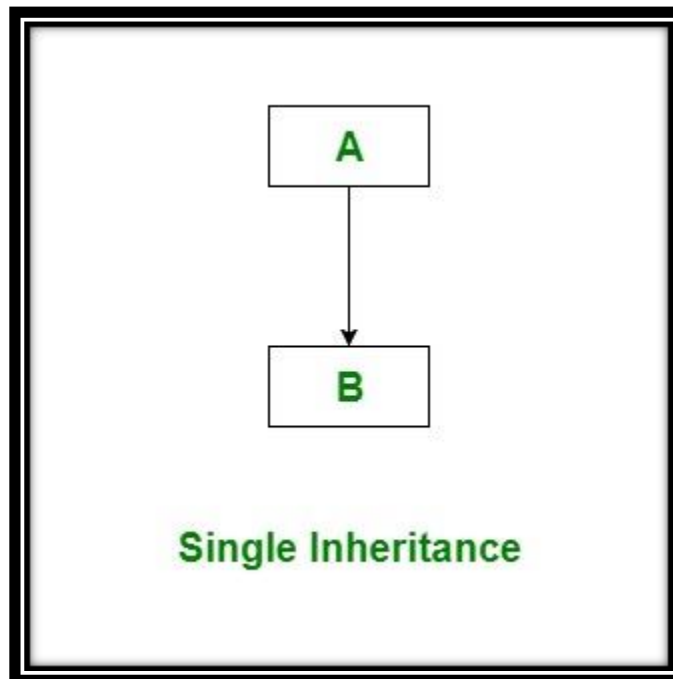


SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

1) Single Inheritance:

It is the type of inheritance in which there is one base class and one derived class.



Example:

```
public class parent
{
    Public void displayp()
    {
        Console.WriteLine("I am parent");
    }
}
Public class son:parent
{
    Public void displays()
    {
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT


(Affiliated to Saurashtra University)

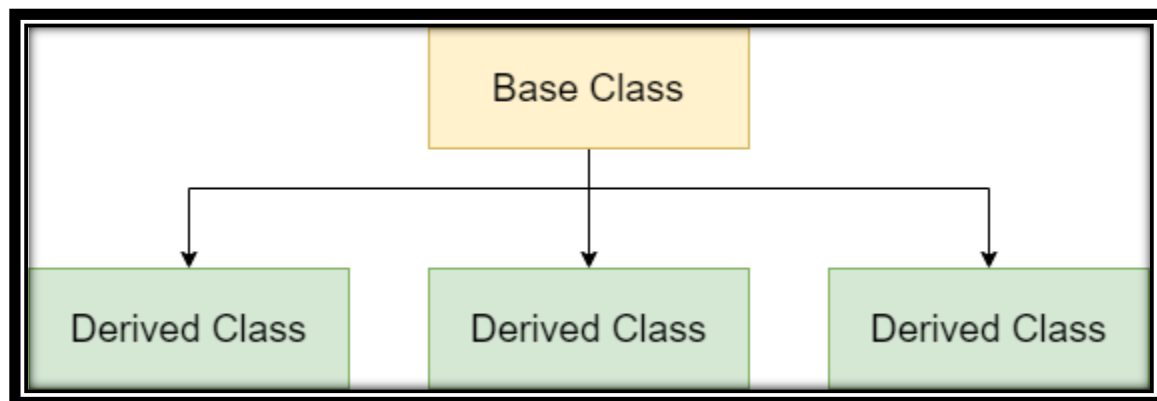
```
        Console.WriteLine("I am son");  
    }  
}
```

Public class program

```
{  
    Public static void main()  
    {  
        son s=new son();  
        s.Displayp();  
        s.Displays();  
    }  
}
```

2) Hierarchical Inheritance:

 In this type of inheritance, multiple classes derives from one base class.



SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Example:

```
public class parent
{
    Public void displayp()
    {
        Console.WriteLine("I am parent");
    }
}
Public class Child1:parent
{
    Public void displaychild1()
    {
        Console.WriteLine("Child1");
    }
}
Public class Child2:parent
{
    Public void displaychild2()
    {
        Console.WriteLine("Child2");
    }
}
Public class program
{
    Public static void main()
    {
        child1 c1=new child1();
        child c2=new child2();
        c1.displaychild1();
        c1.displayp();
    }
}
```

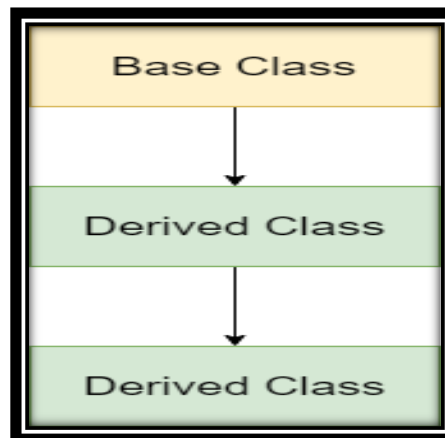

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

```
        c2.displaychild2();
        c2.displayp();
    }
}
```

3) Multilevel Inheritance:

- In this type of inheritance, a class inherits derived/child class which in turn inherits another class
- It's like a child inherits the traits of his/her parents, and parents inherit the traits of their grandparents.



Example:

```
public class Grandparent
{
    Public void displaygp()
    {
        Console.WriteLine("Grand Parent");
    }
}
Public class Parent:Grandparent
{
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

```
Public void displayp()
{
    Console.WriteLine("Parent");
}

Public class child:Parent
{
    Public void displayc()
    {
        Console.WriteLine("Child");
    }
}

Public class program
{
    Public static void main()
    {
        child c=new child();
        c.displayc();
        c.displayp();
        c.displaygp();
    }
}
```

MCQ

1) The concept of parent and child is known as	Inheritance
2) Which type of inheritance is not supported by C# directly?	MultipleIn
3) In which type of inheritance a class inherits from another class, which in turn inherits another class?	Multilevel

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Topic: Write a short note on sealed class

Ans:

Sealed Class:

- Sealed classes are used to restrict the users from inheriting the class.
- A class can be sealed by using the *sealed* keyword.
- The keyword tells the compiler that the class is sealed, and therefore, cannot be extended.
- No class can be derived from a sealed class.

Syntax:

```
sealed class class_name
```

```
{  
    // data members  
    // methods  
    .  
    .  
    .  
}
```

- A method can also be sealed, and in that case, the method cannot be overridden. However, a method can be sealed in the classes in which they have been inherited.

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

```
sealed class A
```

```
{  
  
    public void display()  
  
    {  
  
        Console.WriteLine("Hello");  
  
    }  
  
}
```

```
class B : A
```

```
{  
  
    public void display1()  
  
    {  
  
        Console.WriteLine("hi");  
  
    }  
  
}
```

Output: Error indicating that sealed class can not be inherited

MCQ

1)class is used to restrict the class from being inherited.	Sealed
2) Sealed methods can not be	Overridden
3) No class can be derived from sealed class (T/F)	True

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Topic: Explain Method overloading.

Ans:

- Method overloading means the method with same name but different parameters.

```
class test
{
    public void add(int a, int b)
    {
        Console.WriteLine(a + b);
    }
    public void add(int a, int b, int c)
    {
        Console.WriteLine(a + b + c);
    }
}
class Program
{
    static void Main(string[] args)
    {
        test t1 = new test();
        t1.add(4, 5);
        t1.add(1, 2, 3);
        Console.ReadKey();
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Topic: Write a short note on Constructor

Ans:

- In C#, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C# has the same name as class name.
- There can be two types of constructors in C#.
 - Default Constructor
 - Parameterized Constructor
 - Copy Constructor
 - Static Constructor
 - Private Constructor
- **Default Constructor:**
 - A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.


Example:

```
Class test
{
    Public test()
    {
        Console.WriteLine("Default Constructor");
    }
}
Class program{
    static void Main(string[] args)
    {
        test t1 = new test();
        Console.ReadKey();
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Parametrised Constructor:

 A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects

```
class test
{
    public int a, b;
    public test(int x, int y)
    {
        a = x;
        b = y;
    }
}

class Program
{
    static void Main(string[] args)
    {
        test t1 = new test(4, 5);
        Console.WriteLine(t1.a);
        Console.WriteLine(t1.b);
        Console.ReadKey();
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

<u>MCQ</u>	
1) Generally, the constructor is used for	Initialization of members object
2) The constructor with zero arguments is known as	Default constructor
3) Can the class have multiple constructors?	Yes
4) Constructor has same name as class name (T/F)?	True

Topic: Explain Operator Overloading

Ans:

- Operator overloading gives the ability to use the same operator to do various operations
- It provides additional capabilities to C# operators when they are applied to user-defined data types.
- Only the predefined set of C# operators can be overloaded.
- An operator can be overloaded by defining a function to it.
- The function of the operator is declared by using the **operator keyword**.

Syntax:

Access specifier className operator Operator_symbol (parameters)

```
{  
    // Code  
}
```

Overloading Unary Operator

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

```
class test
{
    public int x;
    public test(int a)
    {
        x = a;
    }
    public static test operator -(test t1)
    {
        t1.x = -t1.x;
        return t1;
    }
    public void display()
    {
        Console.WriteLine("Number 1 is" + x);
    }
}

class Program
{
    static void Main(string[] args)
    {
        test t1 = new test(2);
        t1 = -t1;
        t1.display();
        Console.ReadKey();
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Topic: What is interface?

Ans:

- Interface in C# is a blueprint of a class.
- It cannot have method body and cannot be instantiated.
- It is used to achieve multiple inheritance which can't be achieved by class.
- Interface does not implement any method itself. The methods of the interface are implemented by the class that inherits the interface

```
interface A
{
    void display();
}
class test : A
{
    public void display()
    {
        Console.WriteLine("Hello");
    }
}
class test1:A
{
    public void display()
    {
        Console.WriteLine("Hi");
    }
}
```


SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

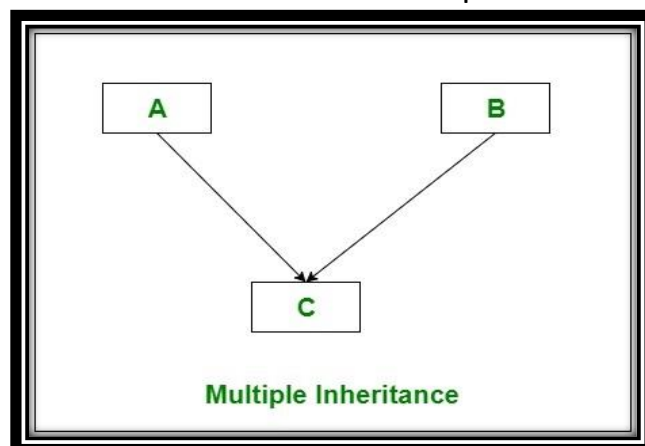
(Affiliated to Saurashtra University)

```
class Program
{
    static void Main(string[] args)
    {
        test t1=new test();
        t1.display();
        test1 t=new test1();
        t.display();
        Console.ReadKey();
    }
}
```

Topic: How to achieve multiple inheritance in c# using interface?

Ans:

-  In Multiple inheritance, one class can have more than one parent or base class and inherit features from all its parent classes.



SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

But, C# does not support multiple inheritance directly, so interface is used to achieve multiple inheritance in C#

```
interface vehicle
{
    void display();
}
interface car
{
    void display1();
}
class mercides :test,test1
{
    public void display()
    {
        Console.WriteLine("Hello");
    }
    public void display1()
    {
        Console.WriteLine("Hi");
    }
}
class Program
{
    static void Main(string[] args)
    {
        Mercides m=new mercides();
        m.display();
        m.display1();
        Console.ReadKey();
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Topic: What is property? Explain in detail

Ans:

- Property in C# is a member of a class that provides a flexible mechanism for classes to access private fields.
- Internally, C# properties are special methods called accessors.
- A C# property has two accessors, get property accessor and set property accessor.
- A get accessor returns a property value, and a set accessor assigns a new value.
- The `value` keyword represents the value of a property.
- Properties can be read-write, read-only, or write-only.
- The read-write property implements both, a get and a set accessor.
- A write-only property implements a set accessor, but no get accessor.
- A read-only property implements a get accessor, but no set accessor.

```
class test
{
    private int roll;
    public int rollnumber
    {
        get
        {
            return roll;
        }
        set
        {
            roll = value;
        }
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

```
class Program
{
    static void Main(string[] args)
    {
        test t1=new test();
        t1.rollnumber=1;
        Console.WriteLine(t1.rollnumber);
        Console.ReadKey();
    }
}
```

Topic: What is indexer? Explain in detail

- ❖ C# indexers are usually known as smart arrays.
- ❖ A C# indexer is a class property that allows you to access a member variable of a class or struct using the features of an array.
- ❖ In C#, indexers are created using this keyword. Indexers in C# are applicable on both classes and structs.
- ❖ Defining an indexer allows you to create a class like that can allows its items to be accessed an array. Instances of that class can be accessed using the [] array access operator.

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Syntax:

```
<modifier> <returntype> this [argument list]
{
    get
    {
    }
    set
    {
    }
}
```

<modifier> : private, public, protected

<return type> : can be any valid c# data types

this: keyword to indicate the object of current class

[argument-list]: Specifies the parameters of the indexer

Features:

- Indexers are always created with **this** keyword.
- Parameterized property are called indexer.
- Indexers are implemented through get and set accessors for the [] operator.
- ref and out parameter modifiers are not permitted in indexer.
- The formal parameter list of an indexer corresponds to that of a method and at least one parameter should be specified.
- Indexer is an instance member so can't be static but property can be static.

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

- Indexers are used on group of elements.
- Indexer is identified by its signature where as a property is identified its name.
- Indexers are accessed using indexes whereas properties are accessed by names.
- Indexer can be overloaded.

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

```
class student
{
    public string[] names = new string[5];
    public string this[int i]
    {
        get
        {
            return names[i];
        }
        set
        {
            names[i] = value;
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        student s1 = new student();
        s1[0] = "xyz";
        s1[1] = "xxx";
        s1[2] = "abc";
        s1[3] = "aaa";
        s1[4] = "bbb";

        for (int i = 0; i < 5; i++)
        {
            Console.WriteLine(s1[i]);
        }
        Console.ReadKey();
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

<u>Indexer</u>	<u>Property</u>
1) Indexer cannot be static	1) Property can be static
2) Indexer are accessed using index	2) Property are accessed by its names
3) Indexer can be overloaded	3) Property can not be overloaded
4) The name of indexer is this keyword	4) The name of property can be any name
5) Indexers are parametrized	5) Properties are not parameterized.

Topic: What is Delegate? List out types of delegates and explain in detail

Ans:

- Delegates provides the way which tells which method is to be called when an event is triggered.
- Example: If you click on button on form, the program would call a specific method. In simple, it is a type that represents reference to methods with particular parameters list and return type and then calls the method in a program for execution when it is needed.
- Delegate type can be declared using delegate keyword. Once a delegate is declared, delegate object will refer and call those methods whose return type and parameter list matches with delegate declaration.

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Syntax:

[modifier] delegate [return type] [delegatename] ([parameterlist]);

Note: A delegate will call only a method which agrees with its signature and return type.

Types of Delegate:

- 1) Single Cast Delegate
- 2) Multi Cast Delegate

1) Single Cast Delegate:

- It is a kind of delegate that can refer to a single method at one time.
- Single Cast delegate refers to the single method with matching signature.
- Single Cast delegate derives from System.Delegate Class.

Example:

```
public delegate void mydele(int x,int y);
class A
{
    public void add(int x, int y)
    {
        Console.WriteLine("Sum is" + (x + y));
    }
    public void sub(int x, int y)
    {
        Console.WriteLine("Sub is" +(x - y));
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

```
class Program
{
    static void Main(string[] args)
    {
        A a1 = new A();
        mydele d1 = new mydele(a1.add);
        d1(5, 5);
        mydele d2 = new mydele(a1.sub);
        d2(2, 1);
        Console.ReadKey();
    }
}
```

2) Multi Cast Delegate:

- It is a kind of delegate that can refer to multiple methods at one time that have same signature.
- Multicast delegates are also known as **Combinable Delegates**.

Example:

```
public delegate void mydele(int x,int y);
class A
{
    public void add(int x, int y)
    {
        Console.WriteLine("Sum is" + (x + y));
    }
    public void sub(int x, int y)
    {
        Console.WriteLine("Sub is" + (x - y));
    }
}
```




SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

```
class Program
{
    static void Main(string[] args)
    {
        A a1 = new A();
        mydele d1 = new mydele(a1.add);
        mydele d2 = new mydele(a1.sub);
        mydele d3 = d1 + d2;
        d3(5, 6);

        Console.ReadKey();
    }
}
```

Event Delegate:

-  The applications and windows communicate with predefined messages
-  These messages contain various pieces of information to determine both windows and applications actions
-  .Net considers this messages as events.

Example:

Button click is one type of event.

Syntax:

Modifier static event deleagtename eventname;

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Defining the event is a 2 step process.

- 1) You need to define a delegate type that will hold the list of methods to be called when the event is fired.
- 2) Next, you declare an event using event keyword

Example:

```
public delegate void mydelegate();

class Program
{
    public static event mydelegate e1;

    static void Main(string[] args)
    {
        e1 += new mydelegate(Hello);
        e1 += new mydelegate(Hi);
        e1.Invoke();
    }
}
```

```
static void Hello()
{
    Console.WriteLine("Hello");
}
static void Hi()
{
    Console.WriteLine("Hi");
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Topic: What is Pointer in c#? Why it is known as unsafe concept?

Ans:

- ❏ Pointer is the variable which is used to store the address of another variable.
- ❏ When we are using pointers directly in C#, it will generate an error because for c#, pointer is unsafe concept so including pointer in program directly will execute the program in unsafe mode.
- ❏ In C#, unsafe is a keyword to denote a section of code that is not managed by common language runtime
- ❏ In c#, you can define the pointers in unsafe context,
- ❏ To execute the programs in unsafe mode.
 - Go to view tab
 - Select Solution Explorer
 - Expand Solution Explorer- Double click on property button
 - Go to build tab
 - Select the option of “Allow Unsafe Code” and mark it as check

```
class Program
{
    static unsafe void Main(string[] args)
    {
        int a = 10;
        int* p;
        p = &a;
        Console.WriteLine("Address is" + (int)(p));
        Console.WriteLine("Value is" + *p);
        Console.ReadKey();
    }
}
```

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Topic: Write a short note on pointer to array

Ans:

- ❖ In C#, an array name and a pointer to a data type are not the same variable type.
- ❖ For example, `int *p` and `int[] p`, are not same type. You can increment the pointer variable `p` because it is not fixed in memory but an array address is fixed in memory, and you can't increment that.
- ❖ Therefore, if you need to access an array data using a pointer variable, as we traditionally do in C, or C++, you need to fix the pointer using the fixed keyword.

Example:

```
class Program
{
    static unsafe void Main(string[] args)
    {
        int[] a = new int[3] { 10, 11, 12 };
        fixed(int *p=a)

            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine((int)(p + i));
                Console.WriteLine(*(p + i));
            }
        Console.ReadKey();
    }
}
```


SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Topic: Write a short note on pointer to structure

Ans:

- Unlike C/C++, Structures in C# can have members that are methods, fields, indexers, operator, methods, properties or events.
- Pointers are variables that store the addresses of the same type of variable i.e. an int pointer can store an address of an integer, a char pointer can store an address of a char
- Members of the structure can be accessed in 2 ways:
 - Using arrow operator:
 - If the members of structure are public then you can directly access them using arrow operator (->)
 - If they are private then you can define methods for accessing the values and use pointers to access the methods.
 - The arrow operator can be used to access structure variables as well as methods.

Syntax:

Pointername->membername;

SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)

Example:

```
struct student
{
    public int roll;
    public int marks;

    public student(int r, int m)
    {
        roll = r;
        marks = m;
    }
};
class Program
{
    static unsafe void Main(string[] args)
    {

        student s1 = new student(1, 20);
        student s2 = new student(2, 30);
        student* p1 = &s1;
        student* p2 = &s2;
        Console.WriteLine("Student1 Data");
        Console.WriteLine("Roll no is" + p1->roll);
        Console.WriteLine("Marks is" + p1->marks);

        Console.WriteLine("Student 2 Data");
        Console.WriteLine("Roll no is" + p2->roll);
        Console.WriteLine("Marks is" + p2->marks);
        Console.ReadKey();
    }
}
```