Program 5.7 DEMONSTRATION OF C# EXPRESSIONS

```
using System;
class Expressions
{
    public static void Main( )
    {
    // Declaration and Initialization
    int a = 10, b = 5, c = 8, d = 2;
    float x = 6.4F, y = 3.0F;
    // Order of Evaluation
    int answer1 = a * b + c / d;
    int answer2 = a * (b + c) /d;
    // Modulo Operations
    int answer3 = a % c;
    float answer4 = x % y;
    // Logical Operations
    bool bool1 = a > b && c > d;
    bool bool2 = a < b && c > d;
    bool bool3 = a < b || c > d;
    bool bool4 = !(a-b == c);
    Console.WriteLine("Order of Evaluation");
    Console.WriteLine("  a * b + c / d = " + answer1);
    Console.WriteLine("  a * (b + c) / d = " + answer2);
    Console.WriteLine("Modulo Operations");
    Console.WriteLine("  a % c = " + answer3);
    Console.WriteLine("  x % y = " + answer4);
    Console.WriteLine("Logical Operations");
    Console.WriteLine("  a > b && c > d  = " + bool1);
    Console.WriteLine("  a < b && c > d  = " + bool2);
    Console.WriteLine("  a < b || c > d  = " + bool3);
    Console.WriteLine("  !(a-b) == c  = " + bool4);
  }
}
```

Program 6.1 | COUNTING WITH IF STATEMENT

```
using System;
class IfTest
{
    public static void Main( )
    {
       int i, count, count1, count2;
       float[] weight = { 45.0F,55.0F,47.0F,51.0F,54.0F };
       float[] height = { 176.5F,174.2F,168.0F,170.7F,169.0F };

       count  = 0;
       count1 = 0;
       count2 = 0;

       for (i = 0; i <= 4; i++)
       {
          if(weight[i] < 50.0 && height[i] > 170.0)
             {
             count1 = count1 + 1;  // Executed when condition is true
             }
          count = count + 1;      // Always executed
          }
          count2 = count - count1;
          Console.WriteLine("Number of persons with ... ");
       Console.WriteLine("Weight<50 and height>170 = "+count1);
       Console.WriteLine("Others = " + count2);
    }
}
```

Program 6.2 | EXPERIMENTING WITH IF....ELSE STATEMENT

```
using System;
class IfElseTest
{
  public static void Main( )
```



```
  {
    int[] number = { 50, 65, 56, 71, 81 };
    int even = 0, odd = 0;

    for (int i = 0; i < number.Length; i++)
    {
      if ((number[i] % 2) == 0)  // use of modulus operator
      {
        even += 1;  // counting EVEN numbers
      }
      else
      {
        odd += 1;  // counting ODD numbers

          }
    }

    Console.WriteLine("Even Numbers : " + even);
    Console.WriteLine("Odd Numbers : " + odd);
  }
}
```

Program 6.4 | NESTING IF....ELSE STATEMENTS

```
using System;
class IfElseNesting
{
  public static void Main( )
  {
    int a = 325, b = 712, c = 478;
    Console.Write("Largest value is : ");
    if (a > b)
    {
      if (a > c)
      {
      Console.WriteLine(a);
      }
      else
      {
      Console.WriteLine(c);
      }
    }
    else
    {
      if (c > b)
      {
      Console.WriteLine(c);
      }
      else
      {
      Console.WriteLine(b);
      }
    }
  }
}
```

Program 6.6 | TESTING THE SWITCH STATEMENT

```
using System;
class CityGuide
{
     public static void Main( )
     {
          Console.WriteLine("Select your choice");
```



```
          Console.WriteLine("London");
          Console.WriteLine("Bombay");
          Console.WriteLine("Paris");
          Console.WriteLine("Type your choice");
          String name = Console.ReadLine ( );

          switch (name)
          {
               case "Bombay":
                    Console.WriteLine("Bombay:Guide 5");
                    break;
               case "London":
                    Console.WriteLine("London:Guide 10");
                    break;
               case "Paris":
                    Console.WriteLine("Paris:Guide 15");
                    break;
               default:
                    Console.WriteLine ("Invalid choice");
                    break;
               }
        }
}
```

Program 7.2 PRINTING THE MULTIPLICATION TABLE USING DO....WHILE LOOP

```
class DoWhileTest
{
        public static void Main ( )
        {
            int row, column, y;
            row = 1;
            System.Console.WriteLine("Multiplication Table \n");
            do
            {
                column = 1;
                do
                {
                    y = row * column;
                    System.Console.Write(" " + y);
                    column = column + 1;
                }
                while (column <= 3);
                System.Console.WriteLine("\n");
                row = row + 1;
        }
```



```
        while (row <= 3);
    }
}
```

Program 7.4 COMPUTING THE 'POWER OF 2' USING FOR LOOP

```
using System;
class ForTest
{
   public static void Main ( )
   {
      long p;
      int n;
      double q;
      Console.WriteLine("2 to power -n      n      2 to power n");
      p = 1L;
      for (n = 0; n < 10; ++n)
      {
         if (n == 0)
            p = 1L;
```



```
         else
            p = p * 2;
         q = 1.0 / (double)p;
         Console.WriteLine ("{0:F6} {1:D} {2:D}" , q,n,p);
      {
     }
   }
```

Program 8.3 | NESTING OF METHODS

```
using System;
class Nesting
{
     void Largest ( int m, int n )
     {
          int large = Max ( m , n );   //Nesting
          Console.WriteLine( large );
     }
     int  Max (int a, int b)
     {
```



```
          int x = ( a > b ) ? a : b ;
          return ( x );
     }
}
class NestTesting
{
     public static void Main( )
     {
          Nesting next = new Nesting ( );
          next.Largest ( 100, 200 ) ;   //Method call
     }
}
```

Program 9.1 | SORTING A LIST OF NUMBERS

```
using System;
class NumberSorting
{
     public static void Main( )
     {
          int[ ]number = { 55, 40, 80, 65, 71 };
          int n = number.Length;
          Console.Write("Given list : ");
          for (int i = 0; i < n; i++)
          {
               Console.Write("   " + number[i]);
          }
          Console.WriteLine("\n");
          // Sorting begins
          for (int i = 0; i < n; i++)
          {
               for (int j = i+1; j < n; j++)
               {
                    if (number[i] < number[j])
                    {
                         // Interchange values
                         int temp = number[i];
                         number[i] = number[j];
                         number[j] = temp;
                    }
               }
          }
          Console.Write("Sorted list : ");
          for (int i = 0; i < n; i++)
          {
               Console.Write("   " + number[i]);
          }
          Console.WriteLine("   ");
     }
}
```

Program 9.2 | APPLICATION OF TWO-DIMENSIONAL ARRAYS

```
using System;
class MulTable
{
        static int ROWS = 20;
        static int COLUMNS = 20;

    public static void Main( )
    {
        int[,] product = new int[ROWS,COLUMNS];
```



```
        Console.WriteLine("MULTIPLICATION TABLE");
        Console.WriteLine("   ");
        int i,j;
        for (i=10; i<ROWS; i++)
        {
            for (j=10; j<COLUMNS; j++)
            {
                product[i,j] = i*j;
                Console.Write("    " +product[i,j]);
            }
            Console.WriteLine("    ");
        }
    }
}
```

Program 9.4 | USING ARRAYLIST CLASS

```
using System;
using System.Collections;
class City
{
      public static void Main( )
      {
            ArrayList n = new ArrayList ( );
            n.Add ("Madras");
            n.Add ("Bombay");
            n.Add ("Anand");
            n.Add ("Calcutta");
            n.Add  ("Delhi");
            Console.WriteLine("Capacity = "+ n.Capacity);
            Console.WriteLine("Elements present = " + n.Count);
            n.Sort( );
            for (int i = 0, i < n.Count; i ++)
            {
                  Console.WriteLine(n[i]);
            }
            Console.WriteLine( );
            n.RemoveAt(4);
            for (int i = 0 ; i < n.Count ; i ++)
            {
                  Console.WriteLine(n[i]);
            }
      }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;
namespace jaggedArrayExample
{
    class Program
    {
        static void Main(string[] args)
            {
                const int rows = 3;
                // declare the jagged array as 3 rows high
                int[][] jagArr = new int[rows][];
                //a row with 2 elements
                jagArr[0] = new int[2];
                // a row with 3 elements
                jagArr[1] = new int[3];
                // a row with 4 elements
                jagArr[2] = new int[4];
                // Fill some (but not all) elements of the rows
                jagArr[0][1] = 54;
                jagArr[1][0] = 26;
                jagArr[1][1] = 18;
                jagArr[2][0] = 72;
                jagArr[2][3] = 404;
                for (int i = 0; i < 2; i++)
                {
                    Console.WriteLine("This is jagged Array 1 having elements : [0][{0}] = {1}",
                    i, jagArr[0][i]);
                }
            for (int i = 0; i < 3; i++)
                {
                    Console.WriteLine("This is jagged Array 2 having elements : [1][{0}] = {1}",
                    i, jagArr[1][i]);
                }
            for (int i = 0; i < 4; i++)
                {
                Console.WriteLine("This is jagged Array 3 having elements : [2][{0}] = {1}",
                    i, jagArr[2][i]);
                }

            }
    }
}
```

Program 10.2 | COPYING, SEARCHING AND INSERTING STRINGS

```
using System;
using System.Collections.Generic;
using System.Text;
namespace SearchString
{
    class Program
    {
        public void Display()
        {
        string str1 = "";
        Console.Write("Enter a string : ");
        str1 = Console.ReadLine();
        string str2 = "";
        Console.Write("Enter another string string : ");
        str2 = Console.ReadLine();
        string str3 = "C# 2005 is developed in Visual Studio 2005 IDE";
        Console.WriteLine("String str3 is : {0}", str3);
        // the string copy method
        string str5 = string.Copy(str2);
        Console.WriteLine("String str5 is copied from str2: {0}", str5);
        Console.WriteLine("\nString str5 is {0} characters long. ", str5.Length);
        Console.WriteLine("The 10th character of string str3 is : {0}", str3[9]);
        // check if a string ends with a set of characters
        Console.WriteLine("String str3:{0}\nEnds with IDE?: {1}\n",
        str3,
        str3.EndsWith("IDE"));
        Console.WriteLine("Ends with Studio?: {0}",
        str3.EndsWith("Studio"));
        // return the index of the substring
        Console.WriteLine("\nThe first time character 'a' occured in string str1 at position : {0}", str1.
IndexOf("a")+1);
        string str6 = str2.Insert(6, "hello");
        Console.WriteLine("'hello' is inserted in string str6. String s6 is now : {0}\n", str6);
    }
    static void Main(string[] args)
    {
        Program prg = new Program();
        prg.Display();
        }
    }
}
```

Program 12.1 | APPLICATION OF CLASSES AND OBJECTS

```
using System;
class Rectangle
{
     public int length, width;                  //   Declaration of variables

     public void GetData(int x, int y)          //   Definition of method
     {
          length = x;
          width  = y;
     }

     public int RectArea()                      //        Definition of another method
     {
          int area = length * width;
          return (area);
     }
}

class RectArea                                  //   class with main method
{
     public static void Main( )
     {
          int area1,area2;                      //        Local variables
          Rectangle rect1 = new Rectangle();     //  Creating objects
          Rectangle rect2 = new Rectangle();

          rect1.length = 15;                    //   Accessing variables
          rect1.width  = 10;
          area1 = rect1.length * rect1.width;

          rect2.GetData(20,12);                 //   Accessing methods
          area2 = rect2.RectArea();

          Console.WriteLine("Area1 = " + area1);
          Console.WriteLine("Area2 = " + area2);
     }
}
```

Program 12.2 APPLICATION OF CONSTRUCTORS

```
using System;
class Rectangle
{
      public int length, width ;
      public Rectangle(int x, int y) // Defining constructor
      {
            length = x ;
            width = y ;
      }
      public int RectArea( )
      {
            return (length * width);
      }
}
class RectangleArea
{
      public static void Main( )
      {
            Rectangle rect1 = new Rectangle(15,10); // Calling constructor

            int area1 = rect1.RectArea( ) ;
            Console.WriteLine("Area1 = "+ area1) ;
      }
}
```

Program 12.3 | DEFINING AND USING STATIC MEMBERS

```
using System;
class Mathoperation
{
      public static float mul(float x, float y);
      {
            return x*y;
      }
      public static float divide(float x, float y)
      {
            return x/y ;
      }
}
class MathApplication
{
      public void static Main( )
      {
            float a = MathOperation.mul(4.0F,5.0F) ;


            float b = MathOperation.divide(a,2.0F) ;
            Console.WriteLine("b = "+ b) ;
      }
}
```

Program 13.1 | ILLUSTRATION OF A SIMPLE INHERITANCE

```
using System;
Class Item
{
    public void Company ( )                    // base class
    {
        Console.WriteLine("Item Code = XXX");
    }
  }

    class Fan : Item                                    // derived class
    {
        public void Model ( )
        {
            Console.WriteLine("Fan Model : Classic");
        }
    }

    class SimpleInheritance
    {
        public static void Main( )
        {
            Item item = new Item( ) ;
            Fan fan = new Fan( ) ;
            item.Company( ) ;
            fan.Company( ) ;
            fan.Model( ) ;
        }
    }
```

Program 13.2 APPLICATION OF SINGLE INHERITANCE

```
using System;
class Room                    // base class
{
      public int length;
      public int breadth;
      public Room (int x , int y)      // base constructor
```

```
      {
            length          =          x;
            breadth         =          y;
      }
      public int Area ( )
      {
            return (length * breadth );
      }
}
class BedRoom : Room    //Inheriting Room
{
      int height;
                              //subclass constructor
      public Bedroom (int x, int y, int z):base (x,y)
      {

            height = z;
      }
      public int Volume ( )
      {
            return (length * breadth * height);
      }
}
class InherTest
{
      public static void Main( )
      {
            BedRoom room1 = new BedRoom (14, 12, 10);
            int areal = room1.Area ( );          // superclass method
      int volume1 = room1.Volume ( );        // subclass method
            Console.WriteLine("Area1 =   " + areal1);
            Console.WriteLine("Volume1 = " + volume1);
      }
```

Program 13.3 | MULTILEVEL INHERITANCE

```
// Multilevel Inheritance and Function Overriding
using System;
using System.Collections.Generic;
```



```
using System.Text;
namespace MultiLevel_Inheritance
{
    class BaseClass
    {
        public int num1, num2, num3;
        public virtual int function1()
            {
            Console.WriteLine("Base Class: function1() returning 1 called.");
                return 1;
            }
        public virtual int function2()
        {
            Console.WriteLine("Base Class: function2() returning 2 called.");
            return 2;
        }
        public virtual int function3()
        {
            Console.WriteLine("Base Class: function3() returning 3 called.");
            return 3;
        }
        public BaseClass()
        {
            Console.WriteLine("Base Class: Constructor called.");
            num1 = function1();
            num2 = function2();
            num3 = num1 + num2 + function3();
        }
        public static int Main()
        {
            InheritedClass1 ic = new InheritedClass1();
            InheritedClass2 ic2 = new InheritedClass2();
            InheritedClass3 ic3 = new InheritedClass3();
            Console.WriteLine("The value of num1 is " + ic.num1);
            Console.WriteLine("The value of num2 is " + ic2.num2);
            Console.WriteLine("The value of num3 is " + ic3.num3);
            return 0;
        }
    }
    class InheritedClass1 : BaseClass
    {
        public InheritedClass1()
        {
            Console.WriteLine("Inherited Class1: Constructor called.");
        }
        public override int function1()
        {
            Console.WriteLine("Inhented Class: function1() returning 10 called.");
            return 10;
        }

    }
    class InheritedClass2 : BaseClass
    {
        public InheritedClass2()
        {
            Console.WriteLine("Inherited Class2: Constructor called.");
        }
        public override int function2()
        {
            Console.WriteLine("Inherited Class: function2 returning 20 called.");
            return 20;
        }

    }

    class InheritedClass3 : BaseClass
    {
        public InhentedClass3()
        {
            Console.WriteLine("Inherited Class3: Constructor called.");
        }
        public override int function3()
        {
            Console.WriteLine("Inherited Class: function3 returning 30 called.");
            return 30;
        }

    }
}
```

Program 13.4 | ILLUSTRATION OF METHOD OVERRIDING

```
using System;
class Super                          //base class
{
     protected int x ;
     public Super (int x)
     {
          this.x = x ;


     }
     public virtual void Display () // method defined with virtual

     {
          Console.WriteLine ("Super x = " + x);
```



```
     }
}
class Sub : Super                    //derived class
{
     int y;
     public Sub (int x, int y) : base (x)
     {
          this.y = y;
     }
     public override void Display ( )        // method defined again
                                             //with override
     {
          Console.WriteLine("Super x = " + x) ;
          Console.WriteLine("Sub y = " + y);

     }
}
class OverrideTest
{
     public static void Main( )
     {
          Sub s1 = new Sub (100,200) ;
          s1.Display ( ) ;
     }
}
```

Program 13.6 | OPERATION POLYMORPHISM

```
using System;
class Dog
{
}
class Cat
{
}
class Operation
{
     static void Call (Dog d)
     {
          Console.WriteLine ("Dog is called");
     }
     static void Call (Cat c)
     {
          Console.WriteLine (" Cat is called ");
     }
```



```
     public static void Main( )
     {
          Dog dog = new Dog( );
          Cat cat = new Cat ( );
          Call(dog); //invoking Call( )
          Call(cat); //again invoking Call( )
     }
}
```

Program 13.7 | INCLUSION POLYMORPHISM

```
using System;
class Maruthi
{
      public virtual void Display ( ) //virtual method
      {
            Console.WritLine("Maruthi car");
      }
}
class Esteem : Maruthi
{
      public override void Display( )
      {
            Console.WriteLine("Maruthi Esteem");
      }
}
class Zen : Maruthi
{
      public override void Display ( )
      {
            Console.WriteLine("Maruthi Zen");
      }
}
class Inclusion
{
      public static void Main( )
      {
            Maruthi m = new Maruthi ( );
            m = new Esteem ( );        //upcasting
            m.Display ( );
            m = new Zen ( );           //upcasting
            m.Display ( )
      }
}
```

Program 13.7 outputs:

```
Maruthi  : Esteem
Maruthi  : Zen
```

Program 14.1 IMPLEMENTATION OF MULTIPLE INTERFACES

```
using System;
interface Addition
{
     int Add ( );
}
interface Multiplication
{
     int Mul ( );
}
class Computation  : Addition, Multiplication
{
     int x, y;
     public Computation (int x, int y )                    //Constructor
     {
               this.x = x;
               this.y = y;
     }
     public int Add ( )                                    //Implement Add ( )
     {
               return ( x + y );
     }
     public int Mul ( )                                    //Implement Mul ( )
     {
               return ( x * y );
     }
}
class InterfaceTest1
{
     public static void Main( )
     {
               Computation com = new Computation (10,20);
               Addition add = (Addition ) com;                          // casting
               Console.WriteLine ("Sum = " + add.Add ( ));
               Multiplication mul = (Multiplication) com;               // casting
               Console.WriteLine("Product = " + mul.Mul ( ) );
     }
}
```

Program 14.2 | MULTIPLE IMPLEMENTATION OF AN INTERFACE

```
using System;
interface Area
{
     double Compute ( double x );
}
class Square  :  Area
{
     public double Compute (double x)
     {
          return (x * x)
     }
}
class Circle : Area
{
     public double Compute (double x)
     {
     return (Math.PI * x * x);
     }
}
class InterfaceTest2
{
     public static void Main( )
     {
          Square sqr = new Square ( );
          Circle cir = new Circle ( );
          Area area;
          area = sqr as Area;        //casting
          Console.WriteLine ("Area of Square ="
                              + area.Compute(10.0));
          area = cir as Area;        //casting

          Console.WriteLine ("Area of Circle="
                           + area.Compute(10.0));
     }
}
```

Program 15.1 | OVERLOADING UNARY MINUS

```
using System;
class Space
{

      int x, y, z;
      public Space ( int a, int b, int c )
      {

            x = a;
            y = b;
```



```
            z = c;
      }
      public void Display ( )
      {
            Console.Write(“ ” + x);
            Console.Write(“ ” + y);
            Console.Write(“ ” + z);
            Console.WriteLine( );

      }
      public static Space operator - (Space s)
      {
            s.x = -s.x;
            s.y = -s.y;
            s.z = -s.z;
      }
}

class SpaceTest
{
      public static void Main( )
      {
            Space s = new Space ( 10, -20, 30 );

            Console.Write(“ s : ”);
            s.Display( );

            -s;                  //activates opeator -( ) method

            Console.Write(“ s : ”);
            s.Display ( );
      }
}
```

Program 15.2 | OVERLOADING + OPERATOR

```
using System;
class Complex
{
      double x;              //real part
      double y;              //imaginary part
      public Complex ( )
      {
      }
      public Complex(double real, double imag)
      {
            x = real;
            y = imag;
      }
      public static Complex operator + (Complex c1, Complex c2)
      {
            Complex c3 = new Complex ();
            c3.x = c1.x + c2.x;
            c3.y = c1.y + c2.y;
            return (c3);
      }
      public void Display( )
      {
            Console.Write(x);
            Console.Write(" + j" + y);
            Console.WriteLine( );
      }
}
class ComplexTest
{
            public static void Main( )
            {
                  Complex a, b, c;
                  a = new Complex (2.5 , 3.5);
                  b = new Complex (1.6, 2.7);
                  c = a + b;
                  Console.Write(" a = ");
                  a.Display( );
                  Console.Write("b = ");
                  b.Display( );
```



```
                  Console.Write("c =");
                  c.Display( );
            }
      }
```

Program 16.1 | CREATING AND IMPLEMENTING A DELEGATE

```
using System;
//delegate declaration
delegate int ArithOp(int x, int y);
class MathOperation
{
    //delegate methods definition
    public static int Add(int a, int b)
    {
        return (a + b);
    }
```



```
        public static int Sub(int a, int b)
    {
        return (a - b);
    }
    }
        class DelegateTest
    {
        public static void Main( )
    {
        //delegate instances
        ArithOp operation1 = new ArithOp (MathOperation.Add);
        ArithOp operation2 = new ArithOp(MathOperation.Sub);
        //invoking delegates
        int result1 = operation1(200, 100);
        int result2 = operation2(200,100);
        Console.WriteLine("Result1 = " + result1);
        Console.WriteLine("Result2 = " + result2);
    }
}
```

Program 16.2 | **IMPLEMENTING MULTICAST DELEGATES**

```
using System;
delegate void MDelegate( );
class DM
{
     static public void Display( )
     {
          Console.WriteLine("NEW DELHI");
     }
     static public void Print( )
     {
          Console.WriteLine("NEW YORK");
     }
}
class MTest
{
     public static void Main( )
     {
          MDelegate m1 = new MDelegate(DM.Display);
          MDelegate m2 = new MDelegate (DM.Print);
          MDelegate m3 = m1 + m2;
          MDelegate m4 = m2 + m1;
          MDelegate m5 = m3 - m2;
          //invoking delegates
          m3( );
          m4( );
          M5( );
     }
}
```

Program 16.3 | IMPLEMENTING AN EVENT HANDLER

```
using System;
//delegate declaration first
public delegate void Edelegate(string str);

class EventClass
{
        //declaration of event
        public event Edelegate Status;
        public void TriggerEvent( )
    {
        if(Status != null)
        Status (" Event Triggered");
        }
    }
    class EventTest
    {
        public static void Main( )
        {
            EventClass ec = new EventClass( );


            EventTest  et = new EventTest( );
            ec.Status += new EDelegate(et.EventCatch);
        ec.TriggerEvent( );
    }
        public void EventCatch(string str)
    {
        Console.WriteLine(str);
    }
}
```