1 . Develop a c# program to simulate simple calculator for Addition , Subtraction , Multiplication , Division and mod operations. Read the operator and operands through console .

```
using System;
class Calculator
{
   static void Main()
   {
      while(true)
      {
         Console.WriteLine("Simple Calculator in C#");
         Console.WriteLine("1. Addition");
         Console.WriteLine("2. Subtraction");
         Console.WriteLine("3. Multiplication");
         Console.WriteLine("4. Division");
         Console.WriteLine("5. Exit");
         Console.Write("Enter your choice  ");
         int choice = Convert.ToInt32(Console.ReadLine());
         Console.Write("Enter the first number: ");
         double num1 = Convert.ToDouble(Console.ReadLine());
         Console.Write("Enter the second number: ");
         double num2 = Convert.ToDouble(Console.ReadLine());
         double result = 0;
         switch (choice)
         {
```

```
            case 1:
                result = num1 + num2;
                break;
            case 2:
                result = num1 - num2;
                break;
            case 3:
                result = num1 * num2;
                break;
            case 4:
                if (num2 != 0)
                    result = num1 / num2;
                else
                    Console.WriteLine("Error: Division by zero is not allowed.");
                break;
            default:
                Console.WriteLine("Invalid choice");
                continue;
        }

        Console.WriteLine($"Result: {result}\n");
    }
}
}
```

2. Develop a c# program to print Armstrong number between 1 to 1000

```
using System;
class ArmstrongNumbers
{
   // Function to calculate the number of digits in a number
   static int CountDigits(int num)
   {
      int count = 0;
      while (num > 0)
      {
         num /= 10;
         count++;
      }
      return count;
   }

   // Function to check if a number is an Armstrong number
   static bool IsArmstrong(int num)
   {
      int originalNum = num;
      int numDigits = CountDigits(num);
      int sum = 0;

      while (num > 0)
```

```
        {
            int digit = num % 10;
            sum += (int)Math.Pow(digit, numDigits);
            num /= 10;
        }

        return sum == originalNum;
    }

    static void Main()
    {
        Console.WriteLine("Armstrong numbers between 1 and 1000:");

        for (int i = 1; i <= 1000; i++)
        {
            if (IsArmstrong(i))
            {
                Console.WriteLine(i);
            }
        }
}
}
```

3. Develop a C# program to list all substrings in a given string . [Hint: use of Substring () method]

```
using System;
class Program
{
   static void Main()
   {
      Console.Write("Enter a string: ");
      string inputString = Console.ReadLine();
      Console.WriteLine("All substrings in the given string are:");
      ListAllSubstrings(inputString);
   }
   static void ListAllSubstrings(string input)
   {
      for (int i = 0; i < input.Length; i++)
      {
         for (int j = 1; j <= input.Length - i; j++)
         {
            string substring = input.Substring(i, j);
            Console.WriteLine(substring);
         }
      }
}   }
```

4. Develop a C# program to demonstrate Division by Zero and Index Out of Range exceptions

```
using System;

class Program

{

   static void Main()

   {

      try

      {

         // Division by Zero Exception

         int numerator = 10;

         int denominator = 0;

      // This will throw a DivideByZeroException

         int result = numerator / denominator;

// This line won't be executed

         Console.WriteLine($"Result of division: {result}" );

      }

      catch (DivideByZeroException ex)

      {

         Console.WriteLine($"Exception caught: {ex.Message}");

      }


      try
```

```
        {
            // Index Out of Range Exception
            int[] numbers = { 1, 2, 3, 4, 5 };
            int index = 10;
    int value = numbers[index];      // This will throw an IndexOutOfRangeException
Console.WriteLine($"Value at index {index}: {value}");   // This line won't be executed
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine($"Exception caught: {ex.Message}");
        }
    }
}
```

5. Develop a C# program to generate and print Pascal Triangle using Two Dimensional arrays

```
using System;

class PascalTriangle

{

   static void Main()

   {

      Console.Write("Enter the number of rows for Pascal's Triangle: ");

      int numRows = Convert.ToInt32(Console.ReadLine());

      int[][] triangle = GeneratePascalsTriangle(numRows);

      PrintPascalsTriangle(triangle);

   }

   static int[][] GeneratePascalsTriangle(int numRows)

   {

      int[][] triangle = new int[numRows][];

      for (int i = 0; i < numRows; i++)

      {

         triangle[i] = new int[i + 1];

         triangle[i][0] = 1;

         for (int j = 1; j < i; j++)

         {

            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
```

```
            }
            triangle[i][i] = 1;
        }
        return triangle;
    }
    static void PrintPascalsTriangle(int[][] triangle)
    {
        Console.WriteLine("Pascal's Triangle:");
        for (int i = 0; i < triangle.Length; i++)
        {
            for (int j = 0; j < triangle[i].Length; j++)
            {
                Console.Write(triangle[i][j] + " ");
            }
            Console.WriteLine();
        }
    }
}
```

6. Develop a C# program to generate and print Floyds Triangle using Jagged arrays

```
using System;

class Program
{
   static void Main()
   {
      Console.Write("Enter the number of rows for Floyd's Triangle: ");
      int numberOfRows = int.Parse(Console.ReadLine());
      int[][] floydsTriangle = GenerateFloydsTriangle(numberOfRows);
      Console.WriteLine("\nFloyd's Triangle:");
      PrintFloydsTriangle(floydsTriangle);
   }

   static int[][] GenerateFloydsTriangle(int rows)
   {
      int[][] triangle = new int[rows][];

      int currentValue = 1;

      for (int i = 0; i < rows; i++)
      {
         triangle[i] = new int[i + 1];

         for (int j = 0; j <= i; j++)
         {
            triangle[i][j] = currentValue;
            currentValue++;
         }
      }
```

```
        return triangle;
    }

    static void PrintFloydsTriangle(int[][] triangle)
    {
        for (int i = 0; i < triangle.Length; i++)
        {
            for (int j = 0; j < triangle[i].Length; j++)
            {
                Console.Write(triangle[i][j] + " ");

            }

            Console.WriteLine();

        }

    }

}
```

7. Develop a C# program to read a text file and copy the file contents to another text file.

```
using System;
using System.IO;
class Program
{
   static void Main()
   {
      Console.Write("Enter the source file path: ");
      string sourceFilePath = Console.ReadLine();
      Console.Write("Enter the destination file path: ");
      string destinationFilePath = Console.ReadLine();

      try
      {
         // Read the contents of the source file
         string fileContents = File.ReadAllText(sourceFilePath);

         // Write the contents to the destination file
         File.WriteAllText(destinationFilePath, fileContents);

         Console.WriteLine("File contents successfully copied from '{0}' to '{1}'", sourceFilePath, destinationFilePath);
      }
      catch (Exception ex)
```

```
        {
            Console.WriteLine("An error occurred: " + ex.Message);
        }
    }
}
```

8. Develop a C# program to implement Stack with push and pop operations [Hint: Use class , get/set properties , methods for push and pop and main method]

```
using System;

class StackImplementation
{
   static void Main()
   {
      // Create a stack instance
      Stack stack = new Stack();

      while (true)
      {
         Console.WriteLine("Enter the choice:");
         Console.WriteLine("1. Push");
         Console.WriteLine("2. Pop");
         Console.WriteLine("3. Display");
         Console.WriteLine("4. Exit");

         int choice = GetChoice();

         switch (choice)
         {
            case 1:
               Console.WriteLine("Enter the value to push:");
               int valueToPush = GetValue();
               stack.Push(valueToPush);
               break;

            case 2:
               int poppedValue = stack.Pop();
               if (poppedValue != -1)
                  Console.WriteLine($"Popped element: {poppedValue}");
               break;
```

```
            case 3:
                stack.Display();
                break;

            case 4:
                Console.WriteLine("Exiting the program.");
                Environment.Exit(0);
                break;

            default:
                Console.WriteLine("Invalid choice. Please choose a valid option.");
                break;
        }
    }
}

static int GetChoice()
{
    int choice;
    while (!int.TryParse(Console.ReadLine(), out choice))
    {
        Console.WriteLine("Invalid input. Please enter a number.");
    }
    return choice;
}

static int GetValue()
{
    int value;
    while (!int.TryParse(Console.ReadLine(), out value))
    {
        Console.WriteLine("Invalid input. Please enter a number.");
    }
    return value;
}
}
```

```
class Stack
{
    private const int MaxSize = 10;
    private int[] items;
    private int top;

    public Stack()
    {
        items = new int[MaxSize];
        top = -1;
    }

    public void Push(int value)
    {
        if (top == MaxSize - 1)
        {
            Console.WriteLine("Stack overflow! Cannot push more elements.");
            return;
        }

        items[++top] = value;
        Console.WriteLine($"Pushed element: {value}");
    }

    public int Pop()
    {
        if (top == -1)
        {
            Console.WriteLine("Stack underflow! Cannot pop from an empty stack.");
            return -1; // Return a sentinel value indicating underflow
        }

        int poppedValue = items[top--];
        return poppedValue;
    }

    public void Display()
```

```
    {
        if (top == -1)
        {
            Console.WriteLine("Stack is empty.");
            return;
        }

        Console.WriteLine("Stack elements:");
        for (int i = top; i >= 0; i--)
        {
            Console.WriteLine(items[i]);
        }
    }
}
```

9. Design a class "Complex" with data members, constructor and method for overloading a binary operator '+'. Develop a C# program to read Two complex number and Print the results of addition.

```
using System;

class Complex
{
   private double real;
   private double imaginary;

   // Constructor
   public Complex(double real, double imaginary)
   {
      this.real = real;
      this.imaginary = imaginary;
   }

   // Overloaded addition operator
   public static Complex operator +(Complex c1, Complex c2)
   {
      double realSum = c1.real + c2.real;
      double imaginarySum = c1.imaginary + c2.imaginary;
      return new Complex(realSum, imaginarySum);
   }

   // Display the complex number
   public void Display()
   {
      Console.WriteLine($"Result: {real} + {imaginary}i");
   }
}

class Program
{
   static void Main()
   {
```

```
        Console.WriteLine("Enter the first complex number:");
        Console.Write("Real part: ");
        double real1 = Convert.ToDouble(Console.ReadLine());
        Console.Write("Imaginary part: ");
        double imaginary1 = Convert.ToDouble(Console.ReadLine());

        Console.WriteLine("\nEnter the second complex number:");
        Console.Write("Real part: ");
        double real2 = Convert.ToDouble(Console.ReadLine());
        Console.Write("Imaginary part: ");
        double imaginary2 = Convert.ToDouble(Console.ReadLine());

        // Create Complex objects
        Complex complex1 = new Complex(real1, imaginary1);
        Complex complex2 = new Complex(real2, imaginary2);

        // Use the overloaded addition operator
        Complex result = complex1 + complex2;

        // Display the result
        Console.WriteLine("\nFirst complex number:");
        complex1.Display();
        Console.WriteLine("\nSecond complex number:");
        complex2.Display();
        Console.WriteLine("\nResult of addition:");
        result.Display();
    }
}
```

10. Develop a C# program to create a class named shape. Create three sub classes namely: circle, triangle and square, each class has two member functions named draw () and erase (). Demonstrate polymorphism concepts by developing suitable methods, defining member data and main program.

```
using System;

class Shape
{
   public virtual void Draw()
   {
      Console.WriteLine("Drawing a generic shape.");
   }

   public virtual void Erase()
   {
      Console.WriteLine("Erasing a generic shape.");
   }
}

class Circle : Shape
{
   public override void Draw()
   {
      Console.WriteLine("Drawing a circle.");
   }

   public override void Erase()
   {
      Console.WriteLine("Erasing a circle.");
   }
}

class Triangle : Shape
{
   public override void Draw()
   {
      Console.WriteLine("Drawing a triangle.");
   }
```

```
    public override void Erase()
    {
        Console.WriteLine("Erasing a triangle.");
    }
}

class Square : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Drawing a square.");
    }

    public override void Erase()
    {
        Console.WriteLine("Erasing a square.");
    }
}

class Program
{
    static void Main()
    {
        // Create instances of the derived classes
        Shape circle = new Circle();
        Shape triangle = new Triangle();
        Shape square = new Square();

        // Demonstrate polymorphism through the Draw and Erase methods
        Console.WriteLine("Demonstrating polymorphism:");
        DrawAndErase(circle);
        DrawAndErase(triangle);
        DrawAndErase(square);
    }

    static void DrawAndErase(Shape shape)
    {
        shape.Draw();
        shape.Erase();
        Console.WriteLine();
```

```
    }
}
```

11. Develop a C# program to create an abstract class Shape with abstract methods calculate Area() and calculate Perimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.

```
using System;

abstract class Shape
{
    public abstract double CalculateArea();
    public abstract double CalculatePerimeter();
}

class Circle : Shape
{
    private double radius;

    public Circle(double radius)
    {
        this.radius = radius;
    }

    public override double CalculateArea()
    {
        return Math.PI * radius * radius;
    }

    public override double CalculatePerimeter()
    {
        return 2 * Math.PI * radius;
    }
}

class Triangle : Shape
{
    private double side1, side2, side3;
```

```
    public Triangle(double side1, double side2, double side3)
    {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }

    public override double CalculateArea()
    {
        // Using Heron's formula to calculate the area of a triangle
        double s = (side1 + side2 + side3) / 2;
        return Math.Sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }

    public override double CalculatePerimeter()
    {
        return side1 + side2 + side3;
    }
}

class Program
{
    static void Main()
    {
        // Input for Circle
        Console.WriteLine("Enter the radius of the circle:");
        double circleRadius = Convert.ToDouble(Console.ReadLine());
        Circle circle = new Circle(circleRadius);

      // Input for Triangle
Console.WriteLine("Enter the side lengths of the triangle (separated by  spaces):");
        string[] triangleSides = Console.ReadLine().Split(' ');
        double side1 = Convert.ToDouble(triangleSides[0]);
        double side2 = Convert.ToDouble(triangleSides[1]);
        double side3 = Convert.ToDouble(triangleSides[2]);
        Triangle triangle = new Triangle(side1, side2, side3);

// Calculate and display the area and perimeter of each shape
        Console.WriteLine("\nCircle - Area: " + circle.CalculateArea() + ", Perimeter:
" + circle.CalculatePerimeter());
```

```
    Console.WriteLine("Triangle - Area: " + triangle.CalculateArea() + ", Perimeter:
" + triangle.CalculatePerimeter());
    }
}
```

12. Develop a C# program to create an interface Resizable with methods resizeWidth(int width) and resizeHeight(int height) that allow an object to be resized. Create a class Rectangle that implements the Resizable interface and implements the resize methods

```
using System;

// Define the Resizable interface
interface Resizable
{
    void ResizeWidth(int width);
    void ResizeHeight(int height);
}

// Implement the Resizable interface in the Rectangle class
class Rectangle : Resizable
{
    private int width;
    private int height;

    public Rectangle(int width, int height)
    {
        this.width = width;
        this.height = height;
    }

    public void Display()
    {
        Console.WriteLine($"Rectangle - Width: {width}, Height: {height}");
    }

    public void ResizeWidth(int newWidth)
    {
        width = newWidth;
        Console.WriteLine($"Resized width to {newWidth}");
    }

    public void ResizeHeight(int newHeight)
    {
        height = newHeight;
```

```
        Console.WriteLine($"Resized height to {newHeight}");
    }
}

class Program
{
    static void Main()
    {
        // Input for initial values
        Console.WriteLine("Enter the initial width of the rectangle:");
        int initialWidth = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("Enter the initial height of the rectangle:");
        int initialHeight = Convert.ToInt32(Console.ReadLine());

        // Create an instance of Rectangle
        Rectangle rectangle = new Rectangle(initialWidth, initialHeight);

        // Display the original size of the rectangle
        Console.WriteLine("\nOriginal Size:");
        rectangle.Display();

        // Input for resized values
        Console.WriteLine("\nEnter the new width for resizing:");
        int newWidth = Convert.ToInt32(Console.ReadLine());
        rectangle.ResizeWidth(newWidth);

        Console.WriteLine("Enter the new height for resizing:");
        int newHeight = Convert.ToInt32(Console.ReadLine());
        rectangle.ResizeHeight(newHeight);

        // Display the updated size of the rectangle
        Console.WriteLine("\nUpdated Size:");
        rectangle.Display();
    }
}
```