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

```
using System;

class Calculator
{
   static void Main()
   {
      Console.WriteLine("Simple Arithmetic Calculator");
      Console.WriteLine("Supported Operations: +, -, *, /, %");

      Console.Write("Enter the first operand: ");
      double operand1 = Convert.ToDouble(Console.ReadLine());

      Console.Write("Enter the operator (+, -, *, /, %): ");
      char operation = Convert.ToChar(Console.ReadLine());

      Console.Write("Enter the second operand: ");
      double operand2 = Convert.ToDouble(Console.ReadLine());

      double result = 0;

      switch (operation)
      {
         case '+':
            result = operand1 + operand2;
            break;
         case '-':
            result = operand1 - operand2;
            break;
         case '*':
            result = operand1 * operand2;
            break;
         case '/':
            if (operand2 != 0)
               result = operand1 / operand2;
            else
               Console.WriteLine("Error: Division by zero!");
            break;
         case '%':
```

```
            if (operand2 != 0)
                result = operand1 % operand2;
            else
                Console.WriteLine("Error: Modulus by zero!");
            break;
        default:
            Console.WriteLine("Invalid operator entered.");
            return;
    }

    Console.WriteLine($"Result: {operand1} {operation} {operand2} =
{result}");
  }
}
```

2. Develop a C# program to print Armstrong Number between 1 to 1000.

```
using System;

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

    for (int number = 1; number <= 1000; number++)
    {
      if (IsArmstrongNumber(number))
      {
        Console.WriteLine(number);
      }
    }
  }

  static bool IsArmstrongNumber(int num)
  {
    int originalNumber, remainder, result = 0, n = 0;

    originalNumber = num;

    // Count the number of digits
```

```
        for (originalNumber = num; originalNumber != 0; originalNumber /=
10, ++n);

        originalNumber = num;

        // Calculate the sum of nth power of individual digits
        while (originalNumber != 0)
        {
            remainder = originalNumber % 10;
            result += (int)Math.Pow(remainder, n);
            originalNumber /= 10;
        }

        // Check if the number is Armstrong
        return num == result;
    }
}
```

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

```
using System;

class SubstringList
{
    static void Main()
    {
        Console.Write("Enter a string: ");
        string inputString = Console.ReadLine();

        Console.WriteLine("All Substrings:");

        for (int i = 0; i < inputString.Length; i++)
        {
            for (int j = i + 1; j <= inputString.Length; j++)
            {
                string substring = inputString.Substring(i, j - i);
                Console.WriteLine(substring);
            }
        }
    }
}
```

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

```
using System;

class ExceptionDemo
{
   static void Main()
   {
      Console.WriteLine("Choose an exception to demonstrate:");
      Console.WriteLine("1. Division by Zero");
      Console.WriteLine("2. Index Out of Range");

      Console.Write("Enter your choice (1 or 2): ");
      int choice = int.Parse(Console.ReadLine());

      try
      {
         switch (choice)
         {
            case 1:
               DemonstrateDivisionByZero();
               break;
            case 2:
               DemonstrateIndexOutOfRange();
               break;
            default:
               Console.WriteLine("Invalid choice. Please enter 1 or 2.");
               break;
         }
      }
      catch (Exception ex)
      {
         Console.WriteLine($"An error occurred: {ex.Message}");
      }
   }

   static void DemonstrateDivisionByZero()
   {
      Console.Write("Enter numerator: ");
      int numerator = int.Parse(Console.ReadLine());
```

```
        Console.Write("Enter denominator: ");
        int denominator = int.Parse(Console.ReadLine());

        int result = numerator / denominator;
        Console.WriteLine($"Result of division: {result}");
    }

    static void DemonstrateIndexOutOfRange()
    {
        Console.Write("Enter the size of the array: ");
        int size = int.Parse(Console.ReadLine());

        int[] numbers = new int[size];

        Console.Write("Enter the index to access: ");
        int index = int.Parse(Console.ReadLine());

        int value = numbers[index]; // This line will cause an IndexOutOfRangeException
        Console.WriteLine($"Value at index {index}: {value}");
    }
}
```

5. Develop a C# program to generate and printPascal 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 = int.Parse(Console.ReadLine());
        int[,] triangle = GeneratePascalsTriangle(numRows);

        Console.WriteLine("Pascal's Triangle:");
        PrintPascalsTriangle(triangle);
    }

    static int[,] GeneratePascalsTriangle(int numRows)
```

```
    {
        int[,] triangle = new int[numRows, numRows];
        for (int i = 0; i < numRows; i++)
        {
            for (int j = 0; j <= i; j++)
            {
                if (j == 0 || j == i)
                    triangle[i, j] = 1;
                else
                    triangle[i, j] = triangle[i - 1, j - 1] + triangle[i - 1, j];
            }
        }
        return triangle;
    }
    static void PrintPascalsTriangle(int[,] triangle)
    {
        int numRows = triangle.GetLength(0);
        for (int i = 0; i < numRows; i++)
        {
            for (int j = numRows;j>=i;j--)
          {
                  Console.Write(" ");
          }
            for (int j = 0; j <= i; 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 FloydsTriangle
{
    static void Main()
    {
        Console.Write("Enter the number of rows for Floyd's Triangle: ");
        int numRows = int.Parse(Console.ReadLine());
        int[][] triangle = GenerateFloydsTriangle(numRows);
```

```
        Console.WriteLine("Floyd's Triangle:");
        PrintFloydsTriangle(triangle);
    }
    static int[][] GenerateFloydsTriangle(int numRows)
    {
        int[][] triangle = new int[numRows][];
        int value = 1;
        for (int i = 0; i < numRows; i++)
        {
            triangle[i] = new int[i + 1];
            for (int j = 0; j <= i; j++)
            {
                triangle[i][j] = value++;
            }
        }
        return triangle;
    }

    static void PrintFloydsTriangle(int[][] triangle)
    {
        int numRows = triangle.Length;

        for (int i = 0; i < numRows; 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 FileCopy
{
    static void Main()
```

```
    {
        Console.Write("Enter the path of the source text file: ");
        string sourceFilePath = Console.ReadLine();

        Console.Write("Enter the path of the destination text file: ");
        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 copy successful.");
        }
        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;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        CustomStack stack = new CustomStack();

        Console.WriteLine("Welcome to the Stack Program!");

        while (true)
        {
            Console.WriteLine("\nChoose an option:");
            Console.WriteLine("1. Push an element onto the stack");
```

```
            Console.WriteLine("2. Pop an element from the stack");
            Console.WriteLine("3. Print the current stack");
            Console.WriteLine("4. Exit");

            Console.Write("Enter your choice (1-4): ");
            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1":
                    Console.Write("Enter an element to push onto the stack: ");
                    string element = Console.ReadLine();
                    stack.Push(element);
                    break;
                case "2":
                    stack.Pop();
                    break;
                case "3":
                    stack.PrintStack();
                    break;
                case "4":
                    Console.WriteLine("Exiting the program. Thank you!");
                    return;
                default:
                    Console.WriteLine("Invalid choice. Please enter a number
between 1 and 4.");
                    break;
            }
        }
    }
}

class CustomStack
{
    private Stack<object> stack = new Stack<object>();

    public void Push(object item)
    {
        stack.Push(item);
        Console.WriteLine($"Pushed '{item}' onto the stack.");
    }

    public object Pop()
```

```
    {
        if (stack.Count == 0)
        {
            Console.WriteLine("Stack Underflow. Cannot pop from an empty
stack.");
            return null;
        }

        object poppedItem = stack.Pop();
        Console.WriteLine($"Popped '{poppedItem}' from the stack.");
        return poppedItem;
    }

    public void PrintStack()
    {
        if (stack.Count == 0)
        {
            Console.WriteLine("Stack is empty.");
        }
        else
        {
            Console.Write("Current stack: ");
            foreach (var item in stack)
            {
                Console.Write("" + item + "' ");
            }
            Console.WriteLine();
        }
    }
}
```

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;

    public Complex(double real, double imaginary)
```

```
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    // Overloaded '+' operator for adding two complex numbers
    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);
    }

    public override string ToString()
    {
        return $"{real} + {imaginary}i";
    }
}

class Program
{
    static void Main()
    {
        Console.WriteLine("Enter the first complex number:");
        Complex complex1 = ReadComplexNumber();

        Console.WriteLine("Enter the second complex number:");
        Complex complex2 = ReadComplexNumber();

        // Add two complex numbers using the overloaded '+' operator
        Complex result = complex1 + complex2;

        Console.WriteLine($"Result of addition: {complex1} + {complex2}
= {result}");
    }

    static Complex ReadComplexNumber()
    {
        Console.Write("Enter the real part: ");
        double realPart = double.Parse(Console.ReadLine());

        Console.Write("Enter the imaginary part: ");
        double imaginaryPart = double.Parse(Console.ReadLine());
```

```
            return new Complex(realPart, imaginaryPart);
        }
    }
```

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()
   {
      DisplayShapeDetails(new Circle());
      DisplayShapeDetails(new Triangle());
      DisplayShapeDetails(new Square());
```

```
    }

    static void DisplayShapeDetails(Shape shape)
    {
        Console.WriteLine("Shape Details:");
        shape.Draw();
        shape.Erase();
        Console.WriteLine();
    }
}
```

11. Develop a C# program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). 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 sideA, sideB, sideC;

    public Triangle(double sideA, double sideB, double sideC)
    {
        this.sideA = sideA;
        this.sideB = sideB;
        this.sideC = sideC;
    }

    public override double CalculateArea()
    {
        double s = (sideA + sideB + sideC) / 2;
        return Math.Sqrt(s * (s - sideA) * (s - sideB) * (s - sideC));
    }

    public override double CalculatePerimeter()
    {
        return sideA + sideB + sideC;
    }
}

class Program
{
    static void Main()
    {
        Console.WriteLine("Enter the dimensions of the circle:");
        Console.Write("Enter the radius: ");
        double circleRadius = double.Parse(Console.ReadLine());

        Console.WriteLine("\nEnter the dimensions of the triangle:");
        Console.Write("Enter the length of side A: ");
        double triangleSideA = double.Parse(Console.ReadLine());

        Console.Write("Enter the length of side B: ");
        double triangleSideB = double.Parse(Console.ReadLine());

        Console.Write("Enter the length of side C: ");
        double triangleSideC = double.Parse(Console.ReadLine());

        Circle circle = new Circle(circleRadius);
```

```
        Triangle triangle = new Triangle(triangleSideA, triangleSideB, triangleSideC);

        DisplayShapeDetails(circle, "Circle");
        DisplayShapeDetails(triangle, "Triangle");
    }

    static void DisplayShapeDetails(Shape shape, string shapeName)
    {
        Console.WriteLine($"\n{shapeName} Details:");
        Console.WriteLine($"Area: {shape.CalculateArea()}");
        Console.WriteLine($"Perimeter: {shape.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 ResizeWidth(int newWidth)
    {
```

```
        width = newWidth;
        Console.WriteLine($"Width resized to: {width}");
    }

    public void ResizeHeight(int newHeight)
    {
        height = newHeight;
        Console.WriteLine($"Height resized to: {height}");
    }

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

class Program
{
    static void Main()
    {
        // Create an instance of the Rectangle class
        Rectangle rectangle = new Rectangle(10, 5);

        // Display original dimensions
        Console.WriteLine("Original Dimensions:");
        rectangle.DisplayDimensions();

        // Resize width and height
        rectangle.ResizeWidth(15);
        rectangle.ResizeHeight(8);

        // Display updated dimensions
        Console.WriteLine("\nUpdated Dimensions:");
        rectangle.DisplayDimensions();
    }
}
```