C# Programming lab manual

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

        // Read operator and operands from the console
        Console.Write("Enter operator: ");
        char operation = Console.ReadKey().KeyChar;
        //Console.WriteLine(); // Move to the next line

        Console.Write("Enter operand 1: ");
        double operand1 = Convert.ToDouble(Console.ReadLine());

        Console.Write("Enter operand 2: ");
        double operand2 = Convert.ToDouble(Console.ReadLine());

        // Perform the selected operation
        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 is not
allowed.");
                    return;
                }
                break;

            case '%':
                if (operand2 != 0)
                {
                    result = operand1 % operand2;
                }
                else
                {
                    Console.WriteLine("Error: Modulus by zero is not
allowed.");
                    return;
                }
                break;

            default:
                Console.WriteLine("Error: Invalid operator.");
                return;
        }

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

        Console.ReadLine(); // Keep the console window open
    }
}
```

OUTPUT:

```
PS D:\Program1> dotnet run
Simple Arithmetic Calculator
Supported operations: +, -, *, /, %
Enter operator: +
Enter operand 1: 2
Enter operand 2: 3
Result: 2 + 3 = 5
PS D:\Program1> dotnet run
Simple Arithmetic Calculator
Supported operations: +, -, *, /, %
Enter operator: -
Enter operand 1: 8
Enter operand 2: 5
Result: 8 - 5 = 3
PS D:\Program1> dotnet run
Simple Arithmetic Calculator
Supported operations: +, -, *, /, %
Enter operator: *
Enter operand 1: 2
Enter operand 2: 4
Result: 2 * 4 = 8
PS D:\Program1> dotnet run
Simple Arithmetic Calculator
Supported operations: +, -, *, /, %
Enter operator: /
Enter operand 1: 10
Enter operand 2: 2
Result: 10 / 2 = 5
PS D:\Program1> dotnet run
Simple Arithmetic Calculator
Supported operations: +, -, *, /, %
Enter operator: %
Enter operand 1: 100
Enter operand 2: 5
Result: 100 % 5 = 0
```

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

```
using System;

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

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

        Console.ReadLine(); // Keep the console window open
    }

    static bool IsArmstrongNumber(int number)
    {
        int originalNumber = number;
        int sum = 0;
        int numDigits = (int)Math.Floor(Math.Log10(number)) + 1;

        while (number > 0)
        {
            int digit = number % 10;
            sum += (int)Math.Pow(digit, numDigits);
            number /= 10;
        }

        return sum == originalNumber;
    }
}
```

OUTPUT:

```
PS D:\Program2> dotnet run
Armstrong numbers between 1 and 1000:
1
2
3
4
5
6
7
8
9
153
370
371
407
```

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:");

        // Call the function to print all substrings
        PrintAllSubstrings(inputString);
    }

    static void PrintAllSubstrings(string? str)
    {
         if (str != null)
        {
        // Nested loops to generate all possible substrings
        for (int start = 0; start < str.Length; start++)
        {
            for (int length = 1; length <= str.Length - start; length++)
            {
                // Use the Substring method to extract the substring
                string substring = str.Substring(start, length);

                // Print the substring
                Console.WriteLine(substring);
            }
            Console.WriteLine("\n");
        }
    }
    }
}
```

OUTPUT:

```
PS D:\Program3> dotnet run
Enter a string: hello
All substrings:
h
he
hel
hell
hello


e
el
ell
ello


l
ll
llo


l
lo


o
```

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

```
using System;

class Program
{
    static void Main()
    {
        // Division by Zero Exception
        DivisionByZeroExceptionExample();

        // Index Out of Range Exception
        IndexOutOfRangeExceptionExample();

        Console.ReadLine(); // To keep the console window open
    }

    static void DivisionByZeroExceptionExample()
    {
        try
        {
            int numerator = 10;
            int denominator = 0;
            int result = numerator / denominator;
            Console.WriteLine("Result of division: " + result);
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine("Division by Zero Exception: " + ex.Message);
        }
    }

    static void IndexOutOfRangeExceptionExample()
    {
        try
        {
            int[] numbers = { 1, 2, 3, 4, 5 };
            int index = 10; // Accessing an index that is out of range
            int value = numbers[index];
            Console.WriteLine("Value at index " + index + ": " + value);
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine("Index Out of Range Exception: " + ex.Message);
        }
    }
}
```

OUTPUT:

```
PS D:\Program4> dotnet run
Division by Zero Exception: Attempted to divide by zero.
Index Out of Range Exception: Index was outside the bounds of the array.

//with denominator = 2
PS D:\Program4> dotnet run
Result of division: 5
Index Out of Range Exception: Index was outside the bounds of the array.

//with index = 2
PS D:\Program4> dotnet run
Division by Zero Exception: Attempted to divide by zero.
Value at index 2: 3
```

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

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Enter the number of rows for Pascal's Triangle: ");
        if (int.TryParse(Console.ReadLine(), out int numRows) && numRows > 0)
        {
            int[][] pascalsTriangle = GeneratePascalsTriangle(numRows);
            PrintPascalsTriangle(pascalsTriangle);
        }
        else
        {
            Console.WriteLine("Please enter a valid positive integer for the
number of rows.");
        }

        Console.ReadLine(); // To keep the console window open
    }

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; // First element in each row is always 1

            for (int j = 1; j < i; j++)
            {
                triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
            }
            triangle[i][i] = 1; // Last element in each row is always 1
        }
        return triangle;
    }

static void PrintPascalsTriangle(int[][] triangle)
    {
        Console.WriteLine("Pascal's Triangle:");

        foreach (var row in triangle)
        {
```

```
            foreach (var number in row)
            {
                Console.Write(number + " ");
            }
            Console.WriteLine();
        }
    }
}
```

OUTPUT:

```
PS D:\Program5> dotnet run
Enter the number of rows for Pascal's Triangle: 5
Pascal's Triangle:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

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: ");
        if (int.TryParse(Console.ReadLine(), out int n) && n > 0)
        {
            int[][] floydsTriangle = GenerateFloydsTriangle(n);
            PrintFloydsTriangle(floydsTriangle);
        }
        else
        {
            Console.WriteLine("Please enter a valid positive integer.");
        }
    }

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

        int number = 1;
        for (int i = 0; i < n; i++)
        {
            triangle[i] = new int[i + 1];
            for (int j = 0; j <= i; j++)
            {
                triangle[i][j] = number++;
            }
        }

        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();
        }
```

```
    }
}
```

OUTPUT:

```
PS D:\Program6> dotnet run
Enter the number of rows for Floyd's Triangle: 5
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

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.WriteLine("Enter the path of the source text file:");
        string? sourceFilePath = Console.ReadLine();//? is used for making
sourceFilePath nullable

        Console.WriteLine("Enter the path of the destination text file:");
        string? destinationFilePath = Console.ReadLine();//? is used for
making destinationFilePath nullable

        try
        {

            if (sourceFilePath != null)
            {
            // Read the content from the source file
            string? content = File.ReadAllText(sourceFilePath!);//! is a null-
forgiving operator


            // Write the content to the destination file
            File.WriteAllText(destinationFilePath!, content);//! is a null-
forgiving operator
            }

            Console.WriteLine("File content copied successfully!");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
}
```

OUTPUT:

```
PS D:\Program7> dotnet run
Enter the path of the source text file:
D:\Program7\source.txt
Enter the path of the destination text file:
D:\Program7\dest.txt
File content copied successfully!
```

8. Develop a C# 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 Stack<T>
{
    private T[] array;
    private int top;
    private int maxSize;

    public Stack(int size)
    {
        array = new T[size];
        top = -1;
        maxSize = size;
    }

    public void Push(T item)
    {
        if (top == maxSize - 1)
        {
            Console.WriteLine("Stack Overflow. Cannot push {0}.", item);
        }
        else
        {
            array[++top] = item;
            Console.WriteLine("{0} pushed to stack.", item);
        }
    }

    public T? Pop() // Return type marked as nullable
    {
        if (top == -1)
        {
            Console.WriteLine("Stack Underflow. Cannot pop from an empty
stack.");
            return default!; // Use null-forgiving operator
        }
        else
        {
            T poppedItem = array[top--];
            Console.WriteLine("{0} popped from stack.", poppedItem);
            return poppedItem;
        }
    }
}
```

```
class Program
{
    static void Main()
    {
        // Create a stack of integers
        Stack<int> intStack = new Stack<int>(5);

        // Push some elements onto the stack
        intStack.Push(10);
        intStack.Push(20);
        intStack.Push(30);

        // Pop elements from the stack
        int? poppedItem1 = intStack.Pop();
        int? poppedItem2 = intStack.Pop();

        // Display popped elements
        Console.WriteLine("Popped items: {0}, {1}", poppedItem1, poppedItem2);

        // Create a stack of strings
        Stack<string> stringStack = new Stack<string>(3);

        // Push some elements onto the stack
        stringStack.Push("Hello");
        stringStack.Push("World");

        // Pop elements from the stack
        string? poppedString = stringStack.Pop();
        // Display popped element
        Console.WriteLine("Popped string: {0}", poppedString);
    }
}
```

OUTPUT:

```
PS D:\Program8> dotnet run
10 pushed to stack.
20 pushed to stack.
30 pushed to stack.
30 popped from stack.
20 popped from stack.
Popped items: 30, 20
Hello pushed to stack.
World pushed to stack.
World popped from stack.
Popped string: World
```

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;
    }

    // Method for overloading the binary operator '+'
    public static Complex operator +(Complex c1, Complex c2)
    {
        double realSum = c1.real + c2.real;
        double imaginarySum = c1.imaginary + c2.imaginary;
        Console.WriteLine("Complex number: {0} + {1}i", c1.real,
c1.imaginary);
        Console.WriteLine("Complex number: {0} + {1}i", c2.real,
c2.imaginary);
        return new Complex(realSum, imaginarySum);
    }

    // Method to print the complex number
    public void Print()
    {
        Console.WriteLine("Complex number: {0} + {1}i", real, imaginary);
    }
}

class Program
{
    static void Main()
    {
        // Read the real and imaginary parts of the first complex number
        Console.WriteLine("Enter the real part of the first complex number:");
        double real1 = double.Parse(Console.ReadLine());
        Console.WriteLine("Enter the imaginary part of the first complex
number:");
        double imaginary1 = double.Parse(Console.ReadLine());
```

```
        // Read the real and imaginary parts of the second complex number
        Console.WriteLine("Enter the real part of the second complex
number:");
        double real2 = double.Parse(Console.ReadLine());
        Console.WriteLine("Enter the imaginary part of the second complex
number:");
        double imaginary2 = double.Parse(Console.ReadLine());

        // Create Complex objects for the two input complex numbers
        Complex complex1 = new Complex(real1, imaginary1);
        Complex complex2 = new Complex(real2, imaginary2);

        // Perform addition using the overloaded '+' operator
        Complex result = complex1 + complex2;

        // Print the result
        Console.WriteLine("Result of addition:");
        result.Print();
    }
}
```

OUTPUT:

```
PS D:\Program9> dotnet run
Enter the real part of the first complex number:
2
Enter the imaginary part of the first complex number:
3
Enter the real part of the second complex number:
4
Enter the imaginary part of the second complex number:
5
Complex number: 2 + 3i
Complex number: 4 + 5i
Result of addition:
Complex number: 6 + 8i
```

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;
// Base class Shape
class Shape
{
    // Member functions
    public virtual void Draw()
    {
        Console.WriteLine("Drawing shape...");
    }

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

// Subclass Circle
class Circle : Shape
{
    // Member functions
    public override void Draw()
    {
        Console.WriteLine("Drawing circle...");
    }

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

// Subclass Triangle
class Triangle : Shape
{
    // Member functions
    public override void Draw()
    {
        Console.WriteLine("Drawing triangle...");
    }

    public override void Erase()
    {
```

```
        Console.WriteLine("Erasing triangle...");
    }
}

// Subclass Square
class Square : Shape
{
    // Member functions
    public override void Draw()
    {
        Console.WriteLine("Drawing square...");
    }

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

class Program
{
    static void Main(string[] args)
    {
        // Polymorphism demonstration
        Shape[] shapes = { new Circle(), new Triangle(), new Square() };

        foreach (Shape shape in shapes)
        {
            shape.Draw();
            shape.Erase();
            Console.WriteLine();
        }
    }
}
```

OUTPUT:

```
PS D:\Program10> dotnet run
Drawing circle...
Erasing circle...

Drawing triangle...
Erasing triangle...

Drawing square...
Erasing square...
```

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

// Subclass Circle
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;
    }
}

// Subclass Triangle
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()
    {
        // Heron's formula to calculate 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(string[] args)
    {
        Circle circle = new Circle(5);
        Console.WriteLine("Circle Area: " + circle.CalculateArea());
        Console.WriteLine("Circle Perimeter: " + circle.CalculatePerimeter());

        Triangle triangle = new Triangle(3, 4, 5);
        Console.WriteLine("Triangle Area: " + triangle.CalculateArea());
        Console.WriteLine("Triangle Perimeter: " +
triangle.CalculatePerimeter());
    }
}
```

OUTPUT:

```
PS D:\Program11> dotnet run
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
Triangle Area: 6
Triangle Perimeter: 12
```

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 width)
    {
        this.width = width;
    }

    public void ResizeHeight(int height)
    {
        this.height = height;
    }
    public void DisplayDimensions()
    {
        Console.WriteLine($"Width: {width}, Height: {height}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Create a rectangle
        Rectangle rectangle = new Rectangle(10, 20);
        Console.WriteLine("Original dimensions:");
        rectangle.DisplayDimensions();

        // Resize width and height
```

```
        rectangle.ResizeWidth(15);
        rectangle.ResizeHeight(25);

        Console.WriteLine("\nResized dimensions:");
        rectangle.DisplayDimensions();
    }
}
```

OUTPUT:

```
PS D:\Program12> dotnet run
Original dimensions:
Width: 10, Height: 20

Resized dimensions:
Width: 15, Height: 25
```