DAYANANDA SAGAR ACADEMY OF TECHNOLOGY & MANAGEMENT
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082
(Affiliated to Visvesvaraya Technological University, Belagavi and Approved by AICTE, New Delhi)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(CE, CSE, ECE, EEE, ISE, ME Courses Accredited by NBA, New Delhi
Accredited by NAAC, A+)

C# PROGRAMMING

V Semester

Course Code :21CSL582

[As per the Choice Based Credit System Scheme]

Scheme:2021

Version-1

w.e.f 3rd Nov 2023

Editorial Committee

C# PROGRAMMING Faculty, Dept of CSE

Approved by

HOD, Dept.of CSE

Document Log

Name of the Document	C# Programming Lab Manual
Scheme	2021
Current Version number and date	V1/03-11-2023
Subject code	21CSL582
Editorial Committee	Prof.Chethana V Prof.Krishna Prasad
Computer Programmer	Mr Vinay Kumar C M
Approved by	HOD, Dept.of CSE

Table of Contents

<u>Vision of the Department</u>

Epitomize CSE graduate to carve a niche globally in the field of computer science to excel in the world of information technology and automation by imparting knowledge to sustain skills for the changing trends in the society and industry.

<u>Mission of the Department</u>

M1: To educate students to become excellent engineers in a confident and creative environment through world-class pedagogy.

M2: Enhancing the knowledge in the changing technology trends by giving hands-on experience through continuous educationand by making them to organize &participate in various events.

M3: Impart skills in the field of IT and its related areas with a focus on developing the required competencies and virtues to meet the industry expectations.

M4: Ensure quality research and innovations to fulfill industry, government & social needs.

M5: Impart entrepreneurship and consultancy skills to students todevelop self-sustaining life skills in multi-disciplinary areas.

Program Educational Outcomes:

PEO1: Engage in professional practice to promote the development of innovative systems and optimized solutions for Computer Science and Engineering.

PEO2: Adapt to different roles and responsibilities in multidisciplinary working environment by respecting professionalism and ethical practices within organization and society at national and international level.

PEO3: Graduates will engage in life-long learning and professional development to acclimate the rapidly changing work environment and develop entrepreneurship skills.

Program Specific Outcomes (PSO)

PSO1:**Foundation of Mathematical Concepts:** Ability to use mathematical methodologies to solve theproblem using suitable mathematical analysis, data structure and suitable algorithm.

PSO2:**Foundation of Computer System:** Ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.

PSO3:**Foundations of Software Development:**Ability to grasp the software development lifecycle and methodologies of software systems. Possess competent skills and knowledge of software design process. Familiarity and practical proficiency with a broad area of programming concepts and provide new ideas and innovations towards research.

PSO4:**Foundations of Multi-Disciplinary Work:**Ability to acquire leadership skills to perform professional activities with social responsibilities, through excellent flexibility to function in multi-disciplinary work environment with self-learning skills.

List of Program Outcomes

PO1	**Engineering Knowledge**: Apply knowledge of mathematics and science, with fundamentals of Computer Science & Engineering to be able to solve complex engineering problems related to CSE.
PO2	**Problem Analysis**: Identify, Formulate, review research literature and analyze complex engineering problems related to CSE and reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
PO3	**Design/Development of solutions**: Design solutions for complex engineering problems related to CSE and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety and the cultural societal and environmental considerations.
PO4	**Conduct Investigations of Complex problems**: Use research–based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern Tool Usage**: Create, Select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to computer science related complex engineering activities with an understanding of the limitations
PO6	**The Engineer and Society**: Apply Reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the CSE professional engineering practice .
PO7	**Environment and Sustainability:** Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development
PO8	**Ethics**: Apply Ethical Principles and commit to professional ethics and responsibilities and norms of the engineering practice .
PO9	**Individual and Team Work**: Function effectively as an individual and as a member or leader in diverse teams and in multidisciplinary Settings.
PO10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large such as able to comprehend and with write effective reports and design documentation, make effective presentations and give and receive clear instructions.
PO11	**Project Management and Finance**: Demonstrate knowledge and understanding of the engineering management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
PO12	**Life-Long Learning**: Recognize the need for and have the preparation and ability to engage in independent and life-long learning the broadest context of technological change

Course Details

Course Code:21CSL582 **CIE Marks:**50
Teaching Hours/Week(L:T:P:S):(0:0:2:0) **Exam Hours:**02
Credits:1 **SIE Marks:**50

Course Outcomes: At the end of the course,the student will be able to:

CO1	Apply concepts of OOPs in developing solutions to problems
CO2	Develop programs to involving basic features, handling exception and text files
CO3	Make use of modern tools to develop C# programming and applications

Syllabus

Laboratory Experiments:

Implement the following programs using C# Programming Language

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
2	Develop c# program to print Armstrong Number between 1 to 1000
3	Develop a c# program to list all substrings in a given string [Hint:use of Substring() method]
4	Develop C# program to demonstrate division by zero and index out of range exceptions
5	Develop a C# program to generate and print pascal triangle using two dimensional array
6	Develop a c# program to generate and print floyds triangle using jagged arrays
7	Develop a c# program to read a text file and copy the file contents to another text file
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
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.

10	Develop a C# program to create a class named shape. Create three subclasses 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
11	Develop a c# program to create an abstract class shape with abstract method 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
12	Develop a C# program to create an interface Resizable with methods reizeWidth(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

Assessment Details (both CIE and SEE):

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each course. The student has to secure not less than 35% (18 Marks out of 50) in the semester-end examination (SEE). The student has to secure a minimum of 40% (40 marks out of 100) in the sum total of the CIE(Continuous Internal Evaluation)and SEE (Semester End Examination)taken together.

Continuous Internal Evaluation (CIE):

CIE marks for the practical course is 50 Marks.

The split-up of CIE marks for record/ journal and test are in the ratio 60:40.

• Each experiment to be evaluated for conduction with observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments designed by the faculty who is handling the laboratory session and is made known to students at the beginning of the practical session.

• Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10marks.

• Total marks scored by the students are scaled downed to 30 marks (60% of maximum marks). • Weightage to be given for neatness and submission of record/write-up on time.

• Department shall conduct 02 tests for 100 marks, the first test shall be conducted after the 8th week of the semester and the second test shall be conducted after the 14th week of the semester.

• In each test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.

• The suitable rubrics can be designed to evaluate each student's performance and learning ability. Rubrics suggested in Annexure-II of Regulation book

• The average of 02 tests is scaled down to 20 marks (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and average marks of two tests is the total CIE marks scored by the student.

Semester End Evaluation(SEE):

• SEE marks for the practical course is 50Marks.

• SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the University

• All laboratory experiments are to be included for practical examination.

• (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners.

• Students can pick one question (experiment) from the questions lot prepared by the internal/external examiners jointly.

• Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

• General rubrics suggested for SEE are mentioned here, write up -20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

• The duration of SEE is 02hours Rubrics suggested in Annexure-II of Regulation book

CHAPTER-1

INTRODUCTION

The Creation of C#:C# was created at Microsoft late in the 1990s and was part of Microsoft's overall .NET strategy. It was first released in its alpha version in the middle of 2000. C# is directly related to C, C++, and Java. The family tree for C# is shown in Figure-1

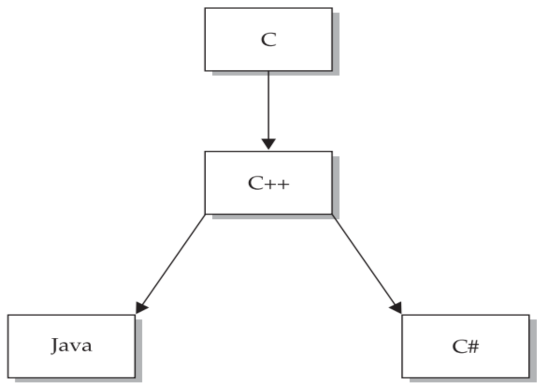

Figure-1

How C# Relates to the .NET Framework : Although C# is a computer language that can be studied on its own, it has a special relationship to its runtime environment, the .NET Framework. The reason for this is twofold. First, C# was initially designed by Microsoft to create code for the .NET Framework. Second, the libraries used by C# are the ones defined by the .NET Framework. Thus, even though it is theoretically possible to separate C# the language from the .NET environment, the two are closely linked. The .NET Framework defines an environment that supports the development and execution of highly distributed, component-based applications. It enables differing computer languages to work together and provides for security, program portability, and a common programming model for the Windows platform. As it relates to C#, the .NET Framework defines two very important entities. The first is the Common Language Runtime (CLR). This is the system that manages the execution of your program. Along with other benefits, the Common Language Runtime is the part of the .NET Framework that enables programs to be portable, supports mixed-language programming, and provides for secure execution. The second entity is the .NET class library. This library gives your program access to the runtime environment. For example, if you want to perform I/O, such as displaying something on the screen, you will use the .NET class library to do it.

An Overview of C#: At the center of C# is object-oriented programming (OOP). The object-oriented methodology is inseparable from C#, and all C# programs are to at least some extent object oriented. Because of its importance to C#, it is useful to understand OOP's basic principles before you write even a simple C# program.

To support the principles of object-oriented programming, all OOP languages, including C#, have three traits in common: encapsulation, polymorphism, and inheritance. Let's examine each.

Encapsulation is a programming mechanism that binds together code and the data it manipulates, and that keeps both safe from outside interference and misuse. In an objectoriented language, code and data can be bound together in such a way that a self-contained black box is created. Within the box are all necessary data and code. When code and data are linked together in this fashion, an object is created. In other words, an object is the device that supports encapsulation. C#'s basic unit of encapsulation is the class. A class defines the form of an object. It specifies both the data and the code that will operate on that data. C# uses a class specification to construct objects. Objects are instances of a class. Thus, a class is essentially a set of plans that specify how to build an object.

Polymorphism (from Greek, meaning "many forms") is the quality that allows one interface to access a general class of actions. A simple example of polymorphism is found in the steering wheel of an automobile. The steering wheel (the interface) is the same no matter what type of actual steering mechanism is used. That is, the steering wheel works the same whether your car has manual steering, power steering, or rack-and-pinion steering. More generally, the concept of polymorphism is often expressed by the phrase "one interface, multiple methods."

Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of hierarchical classification. If you think about it, most knowledge is made manageable by hierarchical (that is, top-down) classifications. For example, a Red Delicious apple is part of the classification apple, which in turn is part of the fruit class, which is under the larger class food. That is, the food class possesses certain qualities (edible, nutritious, and so on) which also, logically, apply to its subclass, fruit.

Some other features of C# are:

Exception handling: An exception is an error that occurs at runtime. . Exception handling streamlines errorhandling by allowing your program to define a block of code, called an exception handler, that is executed automatically when an error occurs. Exception handling is also important because C# defines standard exceptions for common program errors, such as divide-by-zero or index-out-of-range. To respond to these errors, your program must watch for and handle these exceptions.

Interface: An interface defines a set of methods that will be implemented by a class. An interface does not, itself, implement any method. Thus, an interface is a purely logical construct that describes functionality without specifying implementation. Interfaces are syntactically similar to abstract classes. However, in an interface, no method can include a body. That is, an interface provides no implementation whatsoever. It specifies what must be done, but not how. Once an interface is defined, any number of classes can implement it. Also, one class can implement any number of interfaces.

CHAPTER-2

STEPS TO INSTALL MICROSOFT VISUAL STUDIO IDE ON YOUR COMPUTER

Visual Studio is Microsoft's integrated programming environment (IDE). It lets you edit, compile, run, and debug a C# program, all without leaving its well-thought-out environment. Visual Studio offers convenience and helps manage your programs. It is most effective for larger projects, but it can be used to great success with smaller programs. Here are the steps for installing MICROSOFT VISUAL STUDIO.

Step 1 - Make sure your computer is ready for Visual Studio

Step 2 - Determine which version and edition of Visual Studio to install

Step 3 - Initiate the installation

Step 4 - Choose workloads

Step 5 - Choose individual components (optional)

Step 6 - Install language packs (optional)

Step 7 - Select the installation location (optional)

Step 8 - Start developing

The below links provides each step of installation:

https://learn.microsoft.com/en-us/visualstudio/install/install-visual-studio?view=vs-2022

https://www.guru99.com/download-install-visual-studio.html

CHAPTER-3
LAB PROGRAMS

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("----------------------------");

    // Read the operator from the user
    Console.Write("Enter operator (+, -, *, /, %): ");
    char operation = Console.ReadKey().KeyChar;
    Console.WriteLine(); // Move to the next line

    // Read the operands from the user
    Console.Write("Enter first operand: ");
    double operand1 = double.Parse(Console.ReadLine());

    Console.Write("Enter second operand: ");
    double operand2 = double.Parse(Console.ReadLine());

    double result = 0;

    // Perform the calculation based on the operator
    switch (operation)
    {
      case '+':
        result = operand1 + operand2;
        break;
      case '-':
        result = operand1 - operand2;
        break;
      case '*':
        result = operand1 * operand2;
        break;
      case '/':
        // Check for division by zero
        if (operand2 != 0)
        {
          result = operand1 / operand2;
        }
        else
        {
          Console.WriteLine("Error: Division by zero is not allowed.");
```

```
                return;
            }
            break;
        case '%':
            // Check for division by zero
            if (operand2 != 0)
            {
                result = operand1 % operand2;
            }
            else
            {
                Console.WriteLine("Error: Modulo by zero is not allowed.");
                return;
            }
            break;
        default:
            Console.WriteLine("Error: Invalid operator.");
            return;
    }

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

output:

```
Simple Arithmetic Calculator
----------------------------
Enter operator (+, -, *, /, %): +
Enter first operand: 8
Enter second operand: 9
Result: 8 + 9 = 17
```

2.Develop 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 int order(int x)
   {
      int n = 0;
      while (x != 0)
      {
         n++;
         x = x / 10;
      }
      return n;
   }

   static bool IsArmstrongNumber(int num)
   {

      double arm = 0,rem,c;
      int n = order(num);
      c = num;
      while(num>0)
      {
         rem = num % 10;
         arm = Math.Pow(rem,n) + arm;
         num = num/10;

      }
      if (c == arm)
         return true;
      else
      return false;
   }
}
```

Output:

```
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 SubstringExample
{
   static void Main()
   {
      Console.WriteLine("Enter a string:");
      string input = Console.ReadLine();

      Console.WriteLine("\nAll substrings:");
      ListSubstrings(input);
   }

   static void ListSubstrings(string str)
   {
      int length = str.Length;
      for (int start = 0; start < length; start++)
      {
         for (int end = start + 1; end <= length; end++)
         {
            string substring = str.Substring(start, end - start);
            Console.WriteLine(substring);
         }
      }
   }
}
```

Output:

```
Enter a string:
ABC

All substrings:
A
AB
ABC
B
BC
C
```

4. Develop C# program to demonstrate division by zero and index out of range exceptions

```
using System;

class Program
{
   static void Main()
   {
      DivisionByZeroExceptionExample();           // Division by Zero Exception
     IndexOutOfRangeExceptionExample();             // Index Out of Range Exception
   }

   static void DivisionByZeroExceptionExample()
   {
      try
      {
         int numerator = 10;
         int denominator = 0;

         int result = numerator / denominator;
            // The following line won't be executed if an exception occurs
         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 };

         // Accessing an index that is out of the array's bounds
         int value = numbers[10];
        // The following line won't be executed if an exception occurs
         Console.WriteLine($"Value at index 10: {value}");
      }
      catch (IndexOutOfRangeException ex)
      {
         Console.WriteLine($"Index Out of Range Exception: {ex.Message}");
      }
   }
}
```

Output:

```
Division by Zero Exception: Attempted to divide by zero.
Index Out of Range Exception: Index was outside the bounds of the array.
```

5. Develop a C# program to generate and print pascal triangle using two dimensional array

```
using System;

class Program
{
   static void Main()
   {
       Console.Write("Enter the number of rows for Pascal's Triangle: ");
       int numRows = Convert.ToInt32(Console.ReadLine());

       // Generate and print Pascal's Triangle
       int[,] pascalsTriangle = GeneratePascalsTriangle(numRows);
       PrintPascalsTriangle(pascalsTriangle);
   }

   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);
      Console.WriteLine("Pascal's Triangle:");
      for (int i = 0; i < numRows; i++)  {
           // Add spacing to center the triangle
           Console.Write(new string(' ', (numRows - i)));

           for (int j = 0; j <= i; j++)  {
               Console.Write($"{triangle[i, j]} ");   }
             Console.WriteLine();
       } }  }
```

Ouput:

```
Enter the number of rows for Pascal's Triangle: 6
Pascal's Triangle:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 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: ");
      int numRows = Convert.ToInt32(Console.ReadLine());
            // Generate and print Floyd's Triangle
      int[][] floydsTriangle = GenerateFloydsTriangle(numRows);
      PrintFloydsTriangle(floydsTriangle);
   }

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

Output:

```
Enter the number of rows for Floyd's Triangle: 4
Floyd's Triangle:
    1
   1 2
  1 2 3
 1 2 3 4
```

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

      Console.Write("Enter the path of the destination text file: ");
      string destinationFilePath = Console.ReadLine();

      // Read and copy the file contents
      CopyFileContents(sourceFilePath, destinationFilePath);

      Console.WriteLine("File contents copied successfully!");

         /* Display the content */
         string fileContent = File.ReadAllText(destinationFilePath);
         Console.WriteLine("File Content:");
         Console.WriteLine(fileContent);
   }

   static void CopyFileContents(string sourceFilePath, string destinationFilePath)
   {
      try
      {
         // Read the contents of the source file
         string fileContents = File.ReadAllText(sourceFilePath);

         // Write the contents to the destination file
         File.WriteAllText(destinationFilePath, fileContents);
      }
      catch (Exception ex)
      {
         Console.WriteLine($"An error occurred: {ex.Message}");
      }
   }
}
```

Output:

```
Enter the path of the source text file: file1.txt
Enter the path of the destination text file: file2.txt
File contents copied successfully!
(Content of file2.txt displayed)
```

Example path: C:\Users\Hp\source\repos\readfile\readfile\bin\Debug\net8.0

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 StackProgram
{
   static void Main()
   {
      Stack stack = new Stack();           // Create a stack

      stack.Push(10);   // Push elements onto the stack
      stack.Push(20);
      stack.Push(30);
     Console.WriteLine("Stack elements:");    // Display the stack
      stack.Display();

      // Pop an element from the stack
      int poppedElement = stack.Pop();
      Console.WriteLine($"Popped element: {poppedElement}");

      // Display the stack after popping
      Console.WriteLine("Stack elements after popping:");
      stack.Display();
   }
}

class Stack
{
   private const int MaxSize = 10;
   private int[] array = new int[MaxSize];
   private int top = -1;

   // Property to check if the stack is empty
   public bool IsEmpty
   {
      get { return top == -1; }
   }
  // Property to check if the stack is full
   public bool IsFull
   {
      get { return top == MaxSize - 1; }
   }

   // Property to get the current top index
   public int Top
   {
      get { return top; }
   }
```

```
    // Method to push an element onto the stack
    public void Push(int element)
    {
        if (IsFull)
        {
            Console.WriteLine("Stack overflow! Cannot push more elements.");
        }
        else
        {
            array[++top] = element;
            Console.WriteLine($"Pushed element: {element}");
        }
    }
   // Method to pop an element from the stack
    public int Pop()
    {
        if (IsEmpty)
        {
            Console.WriteLine("Stack underflow! Cannot pop from an empty stack.");
            return -1; // Return a default value in case of underflow
        }
        else
        {
            int poppedElement = array[top--];
            return poppedElement;
        }
    }
    // Method to display the elements of the stack
    public void Display()
    {
        if (IsEmpty) {
            Console.WriteLine("Stack is empty.");  }
        else
        {
            for (int i = 0; i <= top; i++)   {
                Console.Write($"{array[i]} ");  }
            Console.WriteLine();
        }
    }
}
```

Output:

```
Pushed element: 10
Pushed element: 20
Pushed element: 30
Stack elements:
10 20 30
Popped element: 30
Stack elements after popping:
10 20
```

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 to initialize the complex number
    public Complex(double real, double imaginary)
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    // Overloaded + operator for complex number addition
    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);
    }

    // Method to display the complex number
    public void Display()
    {
        Console.WriteLine($"Complex Number: {real} + {imaginary}i");
    }
}

class Program
{
    static void Main()
    {
        // Read two complex numbers from the user
        Console.Write("Enter the real part of the first complex number: ");
        double real1 = Convert.ToDouble(Console.ReadLine());

        Console.Write("Enter the imaginary part of the first complex number: ");
        double imaginary1 = Convert.ToDouble(Console.ReadLine());

        Console.Write("Enter the real part of the second complex number: ");
        double real2 = Convert.ToDouble(Console.ReadLine());

        Console.Write("Enter the imaginary part of the second complex number: ");
        double imaginary2 = Convert.ToDouble(Console.ReadLine());

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

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

        // Display the results
        Console.WriteLine("\nResults:");
        complex1.Display();
        complex2.Display();
        sum.Display();
    }
}
```

Output:

```
Enter the real part of the first complex number: 2
Enter the imaginary part of the first complex number: 5
Enter the real part of the second complex number: 8
Enter the imaginary part of the second complex number: 9

Results:
Complex Number: 2 + 5i
Complex Number: 8 + 9i
Complex Number: 10 + 14i
```

10. Develop a C# program to create a class named shape. Create three subclasses 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 function to draw the shape
   public virtual void Draw()
   {
      Console.WriteLine("Drawing a generic shape");
   }

   // Member function to erase the shape
   public virtual void Erase()
   {
      Console.WriteLine("Erasing a generic shape");
   }
}

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

   // Member function to erase a circle
   public override void Erase()
   {
      Console.WriteLine("Erasing a circle");
   }
}

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

   // Member function to erase a triangle
   public override void Erase()
```

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

// Subclass Square
class Square : Shape
{
    // Member function to draw a square
    public override void Draw()
    {
        Console.WriteLine("Drawing a square");
    }
  // Member function to erase a square
    public override void Erase()
    {
        Console.WriteLine("Erasing a square");
    }
}

class Program
{
    static void Main()
    {
        // Demonstrate polymorphism
       // Create an array of shapes
        Shape[] shapes = new Shape[3];

        // Instantiate objects of different subclasses
        shapes[0] = new Circle();
        shapes[1] = new Triangle();
        shapes[2] = new Square();

        // Iterate through the array and call Draw and Erase methods
        foreach (Shape shape in shapes)
        {
            shape.Draw();
            shape.Erase();
            Console.WriteLine(); // Add a newline for better readability
        }
    }
}
```

Output:

```
Drawing a circle
Erasing a circle
Drawing a triangle
Erasing a triangle
Drawing a square
Erasing a square
```

11. Develop a c# program to create an abstract class shape with abstract method 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 base class Shape
abstract class Shape
{
   // Abstract method to calculate area
   public abstract double CalculateArea();

   // Abstract method to calculate perimeter
   public abstract double CalculatePerimeter();
}

// Subclass Circle
class Circle : Shape
{
   private double radius;

   // Constructor for Circle
   public Circle(double radius)
   {
      this.radius = radius;
   }
   // Implementation of CalculateArea for Circle
   public override double CalculateArea()
   {
      return Math.PI * radius * radius;
   }

   // Implementation of CalculatePerimeter for Circle
   public override double CalculatePerimeter()
   {
      return 2 * Math.PI * radius;
   }
}

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

   // Constructor for Triangle
   public Triangle(double side1, double side2, double side3)
   {
      this.side1 = side1;
      this.side2 = side2;
```

```
        this.side3 = side3;
    }

    // Implementation of CalculateArea for Triangle
    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));
    }

    // Implementation of CalculatePerimeter for Triangle
    public override double CalculatePerimeter()
    {
        return side1 + side2 + side3;
    }
}

class Program
{
    static void Main()
    {
        // Demonstrate using the abstract class and its subclasses

        // Create instances of Circle and Triangle
        Circle circle = new Circle(5);
        Triangle triangle = new Triangle(3, 4, 5);

        // Calculate and display area and perimeter for Circle
        Console.WriteLine("Circle - Area: {0}, Perimeter: {1}", circle.CalculateArea(),
circle.CalculatePerimeter());

        // Calculate and display area and perimeter for Triangle
        Console.WriteLine("Triangle - Area: {0}, Perimeter: {1}", triangle.CalculateArea(),
triangle.CalculatePerimeter());
    }
}
```

Output:

```
Circle - Area: 78.53981633974483, Perimeter: 31.41592653589793
Triangle - Area: 6, Perimeter: 12
```

12. Develop a C# program to create an interface Resizable with methods reizeWidth(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;

   // Constructor for Rectangle
   public Rectangle(int width, int height)
   {
      this.width = width;
      this.height = height;
   }
    // Implementation of ResizeWidth method
   public void ResizeWidth(int newWidth)
   {
      if (newWidth > 0)
      {
         width = newWidth;
         Console.WriteLine($"Width resized to {width}");
      }
      else
      {
         Console.WriteLine("Invalid width. Width must be greater than 0.");
      }
   }
     // Implementation of ResizeHeight method
   public void ResizeHeight(int newHeight)
   {
      if (newHeight > 0)
      {
         height = newHeight;
         Console.WriteLine($"Height resized to {height}");
      }
      else
      {
         Console.WriteLine("Invalid height. Height must be greater than 0.");
```

```
        }
    }

    // Method to display the current dimensions of the rectangle
    public void DisplayDimensions()
    {
        Console.WriteLine($"Rectangle Dimensions - Width: {width}, Height: {height}");
    }
}

class Program
{
    static void Main()
    {
        // Demonstrate using the Resizable interface and Rectangle class

        // Create an instance of Rectangle
        Rectangle rectangle = new Rectangle(5, 8);

        // Display initial dimensions
        Console.WriteLine("Initial Rectangle Dimensions:");
        rectangle.DisplayDimensions();

        // Resize width and height using the Resizable interface methods
        rectangle.ResizeWidth(10);
        rectangle.ResizeHeight(12);

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

Output:

```
Initial Rectangle Dimensions:
Rectangle Dimensions - Width: 5, Height: 8
Width resized to 10
Height resized to 12

Updated Rectangle Dimensions:
Rectangle Dimensions - Width: 10, Height: 12
```

CHAPTER 4
BASIC VIVA QUESTIONS

1. What is Object Oriented Programming?
2. What is .NET framework?
3. Explain Encapsulation in C# programming
4. Explain Polymorphism in C# programming
5. Explain Inheritance in C# programming
6. Explain datatypes, variable in C# programming
7. What are the Control statements in C#?
8. What is typecasting?
9. What is exception handling?
10. Explain various types of operators
11. Explain arrays and strings in C# programming
12. Explain operator overloading
13. What is interface in C# ?
14. Explain conditional statements in C#?
15. Explain class and object in C# programming
16. What is this keyword in C#?
17. Explain stack operation using C# programming
18. Explain one dimensional and two dimensional array in C#
19. What is abstract class ?
20. What are the file operations in C# programming?

CHAPTER 5
CONTENT BEYOND SYLLABUS

1.Write a C# program to display the digits of an integer in reverse order.

```
using System;
class DoWhileDemo
{
   static void Main()
   {
      int num;
      int nextdigit;
      num = 198;
      Console.WriteLine("Number: " + num);
      Console.Write("Number in reverse order: ");
      do
      {
         nextdigit = num % 10;
         Console.Write(nextdigit);
         num = num / 10;
      } while (num > 0);
      Console.WriteLine();
   }
}
```

OUTPUT:

```
Number: 198
Number in reverse order: 891
```

2. Write a C# program to compute the sum and product of the numbers from 1 to 10.

```
using System;
class ProdSum
{
   static void Main()
   {
      int prod;
      int sum;
      int i;
      sum = 0;
      prod = 1;
      for (i = 1; i <= 10; i++)
      {
         sum = sum + i;
         prod = prod * i;
      }
      Console.WriteLine("Sum is " + sum);
      Console.WriteLine("Product is " + prod);
   }
}
```

Output:

```
Sum is 55
Product is 3628800
```

3. Write a C# program to Use bitwise OR to make a number odd.

```
using System;
class MakeOdd
{
   static void Main()
   {
      ushort num;
      ushort i;
      for (i = 1; i <= 5; i++)
      {
         num = i;
         Console.WriteLine("num: " + num);
         num = (ushort)(num | 1);
         Console.WriteLine("num after turning on bit zero: "
         + num + "\n");
      }
   }
}
```

Output:

```
num: 1
num after turning on bit zero: 1
num: 2
num after turning on bit zero: 3
num: 3
num after turning on bit zero: 3
num: 4
num after turning on bit zero: 5
num: 5
num after turning on bit zero: 5
```

4. Write a C# program to demonstrate factorial of a given number

```
class Program
{
   static void Main()
   {
      Console.Write("Enter a Number : ");
      int num = int.Parse(Console.ReadLine());
            long factorial = Factorial(num);
      Console.Write($"Factorial of {num} is: {factorial}");
      Console.ReadLine();
   }

   static long Factorial(int number)
   {
      if (number == 1)
         return 1;
      else
         return number * Factorial(number - 1);
   }
}
```

Output:

```
Enter a Number : 5
Factorial of 5 is: 120
```