UNIVERSITY SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY
Sector 16C, Dwarka, Delhi 110075, India

C#.Net Programming Lab
ICT407T

Submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science Engineering

Submitted By:
Bhuwanesh Jung Thapa
(40516403222)
B. Tech CSE (7th Sem)

Submitted to
Mr. Nitendra Singh

Exp 1: Write a simple program to display "Hello C# .Net" and demonstrate use of basic data types, constants, and variables.

Code:

```
using System;
namespace HelloCSharp
{
   class Program
   {
      static void Main(string[] args)
      {
         // Display a welcome message
         Console.WriteLine("Hello C# .Net");

         // Constant declaration
         const double PI = 3.14159;

         // Variable declarations with basic data types
         int age = 21;
         float height = 5.9f;
         double weight = 63.5;
         char grade = 'A';
         string name = "Bhuwanesh";
         bool isStudent = true;

         // Display values
         Console.WriteLine("\n--- Basic Data Types Demonstration ---");
         Console.WriteLine("Name: " + name);
         Console.WriteLine("Age: " + age);
         Console.WriteLine("Height: " + height + " feet");
         Console.WriteLine("Weight: " + weight + " kg");
         Console.WriteLine("Grade: " + grade);
         Console.WriteLine("Is Student: " + isStudent);
         Console.WriteLine("Value of PI (Constant): " + PI);

         // Perform a simple calculation using variables
         double bmi = weight / (height * height);
         Console.WriteLine("\nCalculated BMI: " + bmi);
      } }}
```

Output:

```
Hello C# .Net
```

```
Name: Bhuwanesh
Age: 21
Height: 5.9 feet
Weight: 63.5 kg
Grade: A
Is Student: True
Value of PI (Constant): 3.14159

Calculated BMI: 1.8244791666666667
```

EXP2: Write a program to check whether a given number is an Armstrong number using loops and conditionals.

Code:

```
using System;

namespace ArmstrongCheck
{
   class Program
   {
      static void Main(string[] args)
      {
         int num, originalNum, remainder, result = 0;
         Console.Write("Enter a number: ");
         num = Convert.ToInt32(Console.ReadLine());
         originalNum = num;
         int digits = num.ToString().Length;
           while (num > 0)
         {
            remainder = num % 10;
            result += (int)Math.Pow(remainder, digits);
            num /= 10;
         }

         // Check Armstrong condition
         if (result == originalNum)
            Console.WriteLine(originalNum + " is an Armstrong number.");
         else
            Console.WriteLine(originalNum + " is NOT an Armstrong number.");
      }}}
```

Output:

```
Enter a number: 153
153 is an Armstrong number.
```

```
Enter a number: 123
123 is NOT an Armstrong number.
```

EXP 3: Program to reverse a string and check if it is a palindrome.
Code:

```
using System;

namespace PalindromeCheck
{
   class Program
   {
      static void Main(string[] args)
      {
         // Input a string from user
         Console.Write("Enter a string: ");
         string input = Console.ReadLine();

         // Convert to lowercase to make the check case-insensitive
         string str = input.ToLower();

         // Reverse the string using a loop
         string reversed = "";
         for (int i = str.Length - 1; i >= 0; i--)
         {
            reversed += str[i];
         }

         // Display reversed string
         Console.WriteLine("Reversed String: " + reversed);

         // Check if palindrome
         if (str == reversed)
            Console.WriteLine(input + " is a Palindrome.");
         else
            Console.WriteLine(input + " is NOT a Palindrome.");
      }
   }
}
```

Output:

```
Enter a string: Madam
Reversed String: madam
Madam is a Palindrome.
```

EXP 4 : Create a Student class with methods to calculate percentage and assign grade.

Code:

```
using System;

namespace StudentGradeCalculator
{
   class Student
   {
      // Data members
      public string Name;
      public int RollNo;
      public float Marks1, Marks2, Marks3;  // Marks in 3 subjects

      // Method to calculate percentage
      public float CalculatePercentage()
      {
         float total = Marks1 + Marks2 + Marks3;
         float percentage = (total / 300) * 100;
         return percentage;
      }

      // Method to assign grade based on percentage
      public string AssignGrade(float percentage)
      {
         if (percentage >= 90)
            return "A+";
         else if (percentage >= 75)
            return "A";
         else if (percentage >= 60)
            return "B";
         else if (percentage >= 45)
            return "C";
         else
            return "Fail";
      }

      // Display student details
      public void DisplayDetails()
      {
         float percentage = CalculatePercentage();
         string grade = AssignGrade(percentage);

         Console.WriteLine("\n--- Student Details ---");
         Console.WriteLine("Name: " + Name);
         Console.WriteLine("Roll No: " + RollNo);
         Console.WriteLine("Marks: {0}, {1}, {2}", Marks1, Marks2, Marks3);
         Console.WriteLine("Percentage: " + percentage + "%");
         Console.WriteLine("Grade: " + grade);
```

```
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      // Create a Student object
      Student s1 = new Student();

      // Input details
      Console.Write("Enter Name: ");
      s1.Name = Console.ReadLine();

      Console.Write("Enter Roll Number: ");
      s1.RollNo = Convert.ToInt32(Console.ReadLine());

      Console.Write("Enter Marks in Subject 1: ");
      s1.Marks1 = float.Parse(Console.ReadLine());

      Console.Write("Enter Marks in Subject 2: ");
      s1.Marks2 = float.Parse(Console.ReadLine());

      Console.Write("Enter Marks in Subject 3: ");
      s1.Marks3 = float.Parse(Console.ReadLine());

      // Display calculated details
      s1.DisplayDetails();
    }
  }
}
```

Output:

```
Enter Name: Bhuwanesh
Enter Roll Number: 101
Enter Marks in Subject 1: 85
Enter Marks in Subject 2: 92
Enter Marks in Subject 3: 78

--- Student Details ---
Name: Bhuwanesh
Roll No: 101
Marks: 85, 92, 78
Percentage: 85%
Grade: A
```

EXP 5 : Implement a Complex Number class with constructor overloading and method overloading for addition and subtraction.

```
using System;

namespace ComplexNumberDemo
{
   class Complex
   {
      // Data members
      private double real;
      private double imag;

      // --- Constructor Overloading ---

      // Default constructor
      public Complex()
      {
         real = 0;
         imag = 0;
      }

      // Parameterized constructor
      public Complex(double r, double i)
      {
         real = r;
         imag = i;
      }

      // --- Method Overloading ---

      // Method 1: Add two complex numbers (object as parameter)
      public Complex Add(Complex c)
      {
         return new Complex(this.real + c.real, this.imag + c.imag);
      }

      // Method 2: Add using real and imaginary parts
      public Complex Add(double r, double i)
      {
         return new Complex(this.real + r, this.imag + i);
      }

      // Method 3: Subtract two complex numbers (object as parameter)
      public Complex Subtract(Complex c)
      {
         return new Complex(this.real - c.real, this.imag - c.imag);
      }
```

```
    // Method 4: Subtract using real and imaginary parts
    public Complex Subtract(double r, double i)
    {
        return new Complex(this.real - r, this.imag - i);
    }

    // Display method
    public void Display()
    {
        Console.WriteLine($"{real} + {imag}i");
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Creating complex numbers using different constructors
        Complex c1 = new Complex(4.5, 3.2);
        Complex c2 = new Complex(2.1, 1.3);
        Complex c3 = new Complex();

        Console.WriteLine("First Complex Number: ");
        c1.Display();

        Console.WriteLine("Second Complex Number: ");
        c2.Display();

        // Addition using object parameter
        c3 = c1.Add(c2);
        Console.Write("Addition (object method): ");
        c3.Display();

        // Addition using real and imaginary parameters
        c3 = c1.Add(1.5, 2.5);
        Console.Write("Addition (value method): ");
        c3.Display();

        // Subtraction using object parameter
        c3 = c1.Subtract(c2);
        Console.Write("Subtraction (object method): ");
        c3.Display();

        // Subtraction using real and imaginary parameters
        c3 = c1.Subtract(1.0, 0.5);
        Console.Write("Subtraction (value method): ");
        c3.Display();
    }
}
```

}

Output:

```
First Complex Number:
4.5 + 3.2i
Second Complex Number:
2.1 + 1.3i
Addition (object method): 6.6 + 4.5i
Addition (value method): 6 + 5.7i
Subtraction (object method): 2.4 + 1.9i
Subtraction (value method): 3.5 + 2.7i
```

EXP 6: Create a base class Employee and derived classes Manager, Developer with overridden method Calculate Salary ().

Code:

```
using System;
namespace EmployeeSalaryDemo
{
   // Base class
   class Employee
   { // Common properties
      public string Name;
      public int ID;
      public double BasicSalary;

      // Constructor
 public Employee(string name, int id, double basicSalary)
        {  Name = name;
           ID = id;
           BasicSalary = basicSalary;
      } // Virtual method to calculate salary (can be overridden)
      public virtual double CalculateSalary()
      {return BasicSalary;       }

      // Display employee details
      public virtual void Display()
      {
         Console.WriteLine($"Name: {Name}");
         Console.WriteLine($"Employee ID: {ID}");
         Console.WriteLine($"Basic Salary: {BasicSalary}");
      }
   }

   // Derived class: Manager
   class Manager : Employee
   {
```

```
    public double Allowance;

    public Manager(string name, int id, double basicSalary, double allowance)
        : base(name, id, basicSalary)
    {
        Allowance = allowance;
    }

    // Overridden method
    public override double CalculateSalary()
    {
        return BasicSalary + Allowance;
    }

    public override void Display()
    {
        base.Display();
        Console.WriteLine($"Allowance: {Allowance}");
        Console.WriteLine($"Total Salary: {CalculateSalary()}");
    }
}

// Derived class: Developer
class Developer : Employee
{
    public double Bonus;

    public Developer(string name, int id, double basicSalary, double bonus)
        : base(name, id, basicSalary)
    {
        Bonus = bonus;
    }

    // Overridden method
    public override double CalculateSalary()
    {
        return BasicSalary + Bonus;
    }

    public override void Display()
    {
        base.Display();
        Console.WriteLine($"Bonus: {Bonus}");
        Console.WriteLine($"Total Salary: {CalculateSalary()}");
    }
}

class Program
{
```

```
        static void Main(string[] args)
        {
            // Create Manager object
            Manager mgr = new Manager("Alice Johnson", 101, 50000, 15000);

            // Create Developer object
            Developer dev = new Developer("Bhuwanesh Thapa", 102, 40000, 8000);

            Console.WriteLine("--- Manager Details ---");
            mgr.Display();

            Console.WriteLine("\n--- Developer Details ---");
            dev.Display();
        }
    }
}
```

Output:

```
--- Manager Details ---
Name: Alice Johnson
Employee ID: 101
Basic Salary: 50000
Allowance: 15000
Total Salary: 65000

--- Developer Details ---
Name: Bhuwanesh Thapa
Employee ID: 102
Basic Salary: 40000
Bonus: 8000
Total Salary: 48000
```

Exp 7 : Define an abstract class Shape with abstract method Area (). Implement subclasses Circle and Rectangle.

Code:

```
using System;

namespace ShapeAreaDemo
{
    // Abstract base class
    abstract class Shape
    {
        // Abstract method (must be implemented by derived classes)
        public abstract double Area();
    }

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

        // Constructor
```

```
    public Circle(double r)
    {
        radius = r;
    }

    // Override abstract method
    public override double Area()
    {
        return Math.PI * radius * radius;
    }
}

// Derived class: Rectangle
class Rectangle : Shape
{
    private double length;
    private double width;

    // Constructor
    public Rectangle(double l, double w)
    {
        length = l;
        width = w;
    }

    // Override abstract method
    public override double Area()
    {
        return length * width;
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Create objects of Circle and Rectangle
        Shape circle = new Circle(5.0);
        Shape rectangle = new Rectangle(8.0, 4.0);

        // Display the areas using polymorphism
        Console.WriteLine("--- Shape Area Calculation ---");
        Console.WriteLine("Area of Circle: " + circle.Area());
        Console.WriteLine("Area of Rectangle: " + rectangle.Area());
    }
}
}
```

Output:

```
--- Shape Area Calculation ---
Area of Circle: 78.53981633974483
Area of Rectangle: 32
```

Exp 8: Write a program that reads a number from user and throws a custom exception if it is negative.

```
using System;

namespace CustomExceptionDemo
{
   // Step 1: Create a custom exception class
   class NegativeNumberException : Exception
   {
      public NegativeNumberException(string message) : base(message)
      {
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            // Step 2: Read number from user
            Console.Write("Enter a number: ");
            int number = Convert.ToInt32(Console.ReadLine());

            // Step 3: Check condition and throw custom exception if negative
            if (number < 0)
            {
               throw new NegativeNumberException("Negative numbers are not allowed!");
            }

            // Step 4: If valid number
            Console.WriteLine("You entered: " + number);
         }
         catch (NegativeNumberException ex)
         {
            // Step 5: Catch custom exception
            Console.WriteLine("Custom Exception Caught: " + ex.Message);
         }
         catch (FormatException)
         {
            // Handles invalid (non-numeric) input
```

```
            Console.WriteLine("Error: Please enter a valid integer number.");
        }
        finally
        {
            // Step 6: Always executed
            Console.WriteLine("Program execution completed.");
        }
    }
  }
}
```

Output:

```
Enter a number: 10
You entered: 10
Program execution completed.
```

```
Enter a number: -5
Custom Exception Caught: Negative numbers are not allowed!
Program execution completed.
```

```
Enter a number: abc
Error: Please enter a valid integer number.
Program execution completed.
```

Exp 9 : Create a delegate for string operations (ToUpper, ToLower) and trigger an event when operation is done.

Code:

```
using System;
namespace DelegateEventDemo
{
  // Step 1: Declare a delegate for string operations
  public delegate string StringOperation(string input);

  // Step 2: Declare a class that performs string operations
  class StringProcessor
  {
    // Declare an event that triggers after operation completion
    public event EventHandler<string> OperationCompleted;

    // Method to perform the operation
    public void Process(StringOperation operation, string input)
    {
      string result = operation(input); // Execute delegate
      OnOperationCompleted(result);     // Trigger event
    }
```

```
    // Step 3: Event trigger method
    protected virtual void OnOperationCompleted(string result)
    {
        if (OperationCompleted != null)
        {
            OperationCompleted(this, result);
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Step 4: Create object of StringProcessor
        StringProcessor processor = new StringProcessor();

        // Subscribe to the event
        processor.OperationCompleted += (sender, result) =>
        {
            Console.WriteLine("Operation Completed! Result: " + result);
        };

        // Step 5: Create delegate instances
        StringOperation toUpper = s => s.ToUpper();
        StringOperation toLower = s => s.ToLower();

        // Input from user
        Console.Write("Enter a string: ");
        string input = Console.ReadLine();

        Console.WriteLine("\nChoose operation: ");
        Console.WriteLine("1. ToUpper");
        Console.WriteLine("2. ToLower");
        Console.Write("Enter choice: ");
        int choice = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine(); // spacing

        // Step 6: Perform operation using delegate
        switch (choice)
        {
            case 1:
                processor.Process(toUpper, input);
                break;
            case 2:
                processor.Process(toLower, input);
                break;
```

```
            default:
                Console.WriteLine("Invalid choice!");
                break;
        }}}}
```

Output:

```
Enter a string: Bhuwanesh
Choose operation:
1. ToUpper
2. ToLower
Enter choice: 1

Operation Completed! Result: BHUWANESH

Enter a string: HELLO WORLD
Choose operation:
1. ToUpper
2. ToLower
Enter choice: 2

Operation Completed! Result: hello world
```

Exp 10 : Write a program to demonstrate multithreading by printing numbers in two separate threads. Then extend it to use async/await for file read/write.

Part 1: Multithreading Example — Printing Numbers Using Two Threads

Code :

```
using System;
using System.Threading;


namespace MultithreadingDemo
{
   class Program
   {
      // Method to print even numbers
      static void PrintEvenNumbers()
      {
         for (int i = 2; i <= 10; i += 2)
         {
            Console.WriteLine("Even Thread: " + i);
            Thread.Sleep(500); // Simulate delay
         }
      }

      // Method to print odd numbers
      static void PrintOddNumbers()
```

```
    {
        for (int i = 1; i <= 10; i += 2)
        {
            Console.WriteLine("Odd Thread: " + i);
            Thread.Sleep(500);
        }
    }

    static void Main(string[] args)
    {
        Console.WriteLine("=== MULTITHREADING DEMO ===");

        // Create two threads
        Thread evenThread = new Thread(PrintEvenNumbers);
        Thread oddThread = new Thread(PrintOddNumbers);

        // Start both threads
        evenThread.Start();
        oddThread.Start();

        // Wait for both threads to finish
        evenThread.Join();
        oddThread.Join();

        Console.WriteLine("Both threads completed execution!");
    }
  }

}
```

Output:

```
=== MULTITHREADING DEMO ===
Even Thread: 2
Odd Thread: 1
Even Thread: 4
Odd Thread: 3
Even Thread: 6
Odd Thread: 5
Even Thread: 8
Odd Thread: 7
Even Thread: 10
Odd Thread: 9
Both threads completed execution!
```

Part 2 : Using async/await for File Read & Write

```
using System;
using System.IO;
using System.Threading.Tasks;

namespace AsyncFileDemo
{
   class Program
   {
      // Asynchronous method to write to a file
      static async Task WriteToFileAsync(string filePath, string content)
      {
         using (StreamWriter writer = new StreamWriter(filePath))
         {
            await writer.WriteAsync(content);
         }
         Console.WriteLine(“ File writing completed asynchronously!");
      }

      // Asynchronous method to read from a file
      static async Task ReadFromFileAsync(string filePath)
      {
         using (StreamReader reader = new StreamReader(filePath))
         {
            string data = await reader.ReadToEndAsync();
            Console.WriteLine("\n📖 File content read asynchronously:");
            Console.WriteLine(data);
         }
      }

      // Main async entry point
      static async Task Main(string[] args)
      {
         Console.WriteLine("=== ASYNC/AWAIT FILE DEMO ===");
         string filePath = "output.txt";
         string content = "Hello from async file write!\nThis text is written asynchronously using await.";
```

```
        // Write to file asynchronously
        await WriteToFileAsync(filePath, content);
       // Read from file asynchronously
        await ReadFromFileAsync(filePath);
       Console.WriteLine("\nProgram completed successfully!");
    } }}
```

Output:

```
Hello from async file write!
This text is written asynchronously using await.

Program completed successfully!
```