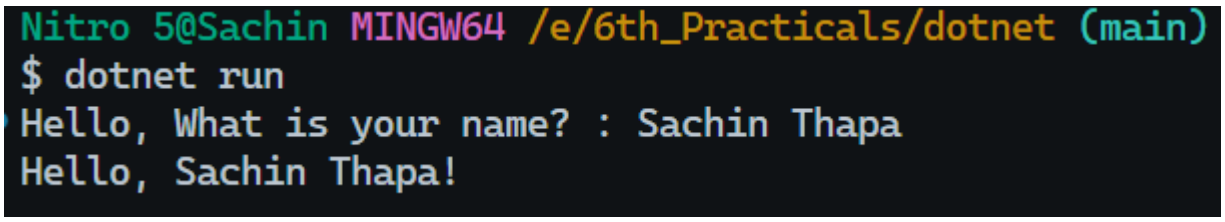
1. Write a program to demonstrate the basic C# structure.

```
using System;

namespace Practical.Codes
{
   public class Labs
   {
      public static void PrintName()
      {
         System.Console.Write("Hello, What is your name? : ");
         string name = Console.ReadLine() ?? string.Empty;
         System.Console.WriteLine($"Hello, {name}!");
      }
   }
}
```



2. Write a program to demonstrate the basics of class and object.

```
namespace Practical.Codes
{
    public class ClassDemo
    {
        public string Name { get; set; }
        public int Age { get; set; }

        public ClassDemo(string name, int age)
        {
            Name = name;
            Age = age;
        }

        public void Display()
        {
            Console.WriteLine($"Name: {Name} and Age: {Age}");
        }
    }

    public class ObjectDemo
    {
        public static void ClassObject()
        {
            ClassDemo obj = new ClassDemo("Sachin Thapa", 25);
```

```
        obj.Display();
    }
  }
}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
Name: Sachin Thapa and Age: 25
```

3. Write a program to illustrate encapsulation with properties and indexers and constructor.

```
class StudentProfile
{
    private string name = String.Empty;
    private int age;
    private readonly string[]? subject;

    // Constructor
    public StudentProfile(string name, int age, string[] subject)
    {
        this.name = name;
        this.age = age;
        this.subject = subject;
    }

    // Properties
    public string Name
    {
        get { return name; }
        set
        {
            if (!string.IsNullOrEmpty(value) && value.Length > 0)
            {
                name = value;
```

```
        }
    }
}

public int Age
{
    get { return age; }
    set
    {
        if (value > 0)
        {
            age = value;
        }
    }
}

// Indexer
public string this[int index]
{
    get
    {
        if (subject == null || index < 0 || index >= subject.Length)
        {
            return "Index is out of range.";
        }
        return subject[index];
```

```
        }
        set
        {

            if (subject != null && index >= 0 && index < subject.Length)
            {
                subject[index] = value;
            }

        }
    }

}

public class PropsAndIndApp
{
    public static void PropertiesAndIndexerDemo()
    {
        // Creating an instance of StudentProfile
        string[] subjects = { "Math", "Science", "History" };
        StudentProfile student = new StudentProfile("John Doe", 20, subjects);

        // Access properties
        Console.WriteLine($"Name: {student.Name}");
        Console.WriteLine($"Age: {student.Age}");
```

```
        // Access indexer
        Console.WriteLine($"Subject at index 1: {student[1]}");

        student[1] = "Biology";
        Console.WriteLine($"Updated Subject at index 1: {student[1]}");

        student[2] = "Geography";
        Console.WriteLine($"Added Subject at index 2: {student[2]}");

        Console.WriteLine($"Subject at index 0: {student[0]}");
        Console.WriteLine($"Subject at index 1: {student[1]}");
    }

}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
Name: John Doe
Age: 20
Subject at index 1: Science
Updated Subject at index 1: Biology
Added Subject at index 2: Geography
Subject at index 0: Math
Subject at index 1: Biology
```

4. Write a program that reflects the overloading and overriding of constructor and function.

```
namespace _04OLOR
{
    class Shape
    {
        public virtual void Display()
        {
            Console.WriteLine("This is a shape Class .");
        }
    }
    class Rectangle : Shape
    {
        private double length;
        private double width;
        public Rectangle(double length, double width)
        {
            this.length = length;
            this.width = width;
        }

        public Rectangle(double sideLength)
        {
            length = sideLength;
```

```
        width = sideLength;
    }

    public override void Display()
    {
        Console.WriteLine($"This is a rectangle with length {length} and width {width}.");
    } }

public class ShapeDisplayApp
{
    public static void RunProgram()
    {
        Shape shape = new Shape();
        shape.Display();

        Rectangle rectangle1 = new Rectangle(7, 8);
        rectangle1.Display();

        Rectangle rectangle2 = new Rectangle(4);
        rectangle2.Display();

    }
}
}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
This is a shape Class .
This is a rectangle with length 7 and width 8.
This is a rectangle with length 4 and width 4.
```

5. WAP to show the use case of the Method Hiding and the Method Overloading along with the new Keyword

```
using System;
namespace _05VirtualAndOverride
{
   class BaseClass
   {
      // use of the virtual keyword
      public virtual void OverrideExample()
      {
         Console.WriteLine("BaseClass OverrideExample");
      }

      public void HideExample()
      {
         Console.WriteLine("BaseClass HideExample");
      }
   }

   class DerivedClass : BaseClass
   {
      // use of the override keyword
      public override void OverrideExample()
      {
         Console.WriteLine("DerivedClass OverrideExample");
```

```
    }

    // use of the new Keyword
    public new void HideExample()
    {
        Console.WriteLine("DerivedClass HideExample");
    }
}

public class PolymorphismDemo
{
    public static void RunProgram()
    {
        DerivedClass obj = new DerivedClass();

        obj.OverrideExample();
        obj.HideExample();
    }
}
}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
DerivedClass OverrideExample
DerivedClass HideExample
```

6. WAP to show the struct and the enum

```
namespace EnumAndStruct
{
    enum OrderStatus
    {
        Pending,
        Shipped,
        Delivered,
        Cancelled
    }

    struct Order
    {
        public int OrderId;
        public string CustomerName;
        public OrderStatus Status;

        public void Display()
        {
            Console.WriteLine($"Order ID: {OrderId}");
            Console.WriteLine($"Customer: {CustomerName}");
            Console.WriteLine($"Status: {Status}");
        }
    }
```

```
    public class EnumStruct
    {
        public static void RunProgram()
        {
            Order order1;
            order1.OrderId = 101;
            order1.CustomerName = "Sachin Thapa";
            order1.Status = OrderStatus.Shipped;
            order1.Display();
            order1.Status = OrderStatus.Delivered;
            Console.WriteLine("\nAfter Delivery:");
            order1.Display();
        }
    }

}
```

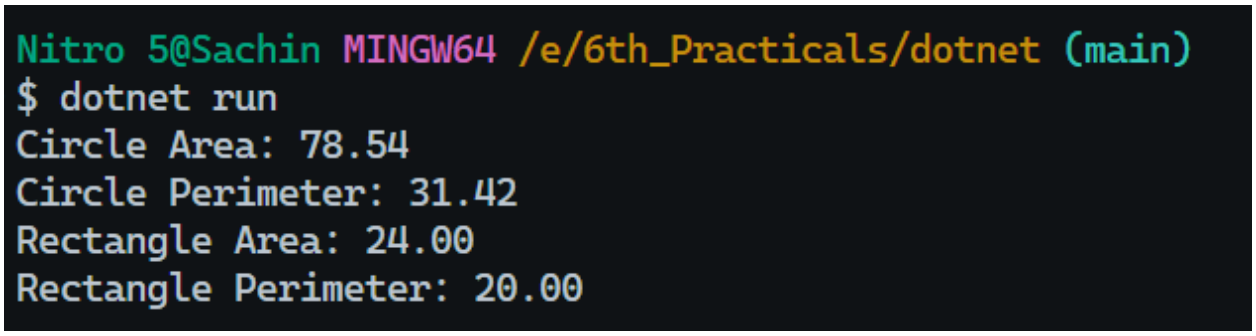
```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
Order ID: 101
Customer: Sachin Thapa
Status: Shipped

After Delivery:
Order ID: 101
Customer: Sachin Thapa
Status: Delivered
```

7. WAP to show the implementation of the Abstract class and the sealed class

```
namespace AbstractAndSealedClass
{
   abstract class Shape
   {
      public abstract double Area();
      public abstract double Perimeter();
   }

   class Circle : Shape
   {
      private double radius;

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

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

      public override double Perimeter()
      {
```

```
        return 2 * Math.PI * radius;
    }
}

sealed class Rectangle : Shape
{
    private double length;
    private double width;

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

    public override double Area()
    {
        return length * width;
    }

    public sealed override double Perimeter()
    {
        return 2 * (length + width);
    }
}
```

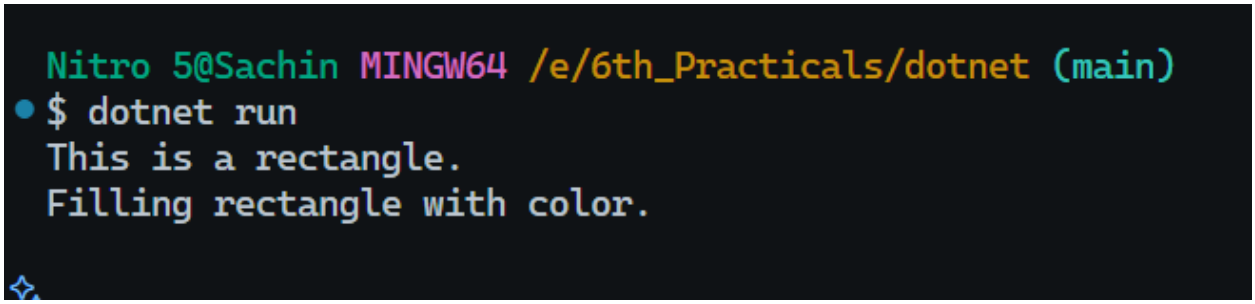
```
    public class ShapeAreaPerimeter
    {
        public static void RunProgram()
        {
            Shape circle = new Circle(5);
            Console.WriteLine($"Circle Area: {circle.Area()}");
            Console.WriteLine($"Circle Perimeter: {circle.Perimeter()}");

            Shape rectangle = new Rectangle(4, 6);
            Console.WriteLine($"Rectangle Area: {rectangle.Area()}");
            Console.WriteLine($"Rectangle Perimeter: {rectangle.Perimeter()}");
        }
    }
}
```



8. Write a program to implement multiple inheritance with the use of interfaces.

```
namespace Interface
{
   interface IShape
   {
      void Display();
   }

   interface IColor
   {
      void FillColor();
   }

   class Rectangle : IShape, IColor
   {
      public void Display()
      {
         Console.WriteLine("This is a rectangle.");
      }

      public void FillColor()
      {
         Console.WriteLine("Filling rectangle with color.");
      }
   }
```

```
    public class InterfaceExample
    {
        public static void RunProgram()
        {
            Rectangle rectangle = new Rectangle();
            rectangle.Display();
            rectangle.FillColor();


        }
    }
}
```



9. Write the Program to show the Error Handling in dotnet using the try catch and finally Block.

```
using System;

// User-defined exception
public class InvalidInputException : Exception
{
    public InvalidInputException(string message) : base(message)
    {
    }
}

public class ErrorHandlingExample
{
    public static void RunProgram()
    {
        try
        {
            Console.Write("Enter numerator: ");
            int numerator = Convert.ToInt32(Console.ReadLine());

            Console.Write("Enter denominator: ");
            int denominator = Convert.ToInt32(Console.ReadLine());

            if (numerator < 0 || denominator < 0)
```

```
        {
            throw new InvalidInputException("Negative numbers are not allowed.");
        }
        int result = numerator / denominator;

        Console.WriteLine($"Result: {result}");
    }
    catch (InvalidInputException ex)
    {
        Console.WriteLine($"Custom Error: {ex.Message}");
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine($"Built-in Error: Cannot divide by zero.{ex.Message}");
    }
    finally
    {
        Console.WriteLine("Execution completed.");
    }
  }
}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
Enter numerator: 5
Enter denominator: 0
Built-in Error: Cannot divide by zero.Attempted to divide by zero.
Execution completed.
```

10. Write a program to demonstrate use of Delegate and Events.

```
namespace DeleGateAndEvent
{
    using System;

    delegate void EventHandler();

    class EventPublisher
    {
        public event EventHandler? MyEvent;

        public void TriggerEvent()
        {

            if (MyEvent != null)
            {
                Console.WriteLine("Event triggered.");

                MyEvent.Invoke();
            }
        }
    }

    class EventSubscriber
    {
```

```
    public void HandleEvent()
    {
        Console.WriteLine("Event handled.");
    }
}

public class EventDemo
{
    public static void RunProgram()
    {

        EventPublisher publisher = new EventPublisher();

        EventSubscriber subscriber = new EventSubscriber();

        publisher.MyEvent += subscriber.HandleEvent;

        publisher.TriggerEvent();

    }
}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
Event triggered.
Event handled.
```

11. Write a program to show the use of generic classes and methods.

```
namespace Collection
{
    using System;
    using System.Collections.Generic;

    public class StudentModel
    {
        public string Name { get; set; } = String.Empty;
        public int Age { get; set; }
        public string Grade { get; set; } = String.Empty;
    }

    public class StudentList
    {
        private List<StudentModel> students = new List<StudentModel>();

        public void AddStudent(StudentModel student)
        {
            students.Add(student);
        }

        public void DisplayStudents()
        {
            foreach (var student in students)
            {
```

```
            Console.WriteLine($"Name: {student.Name}, Age: {student.Age}, Grade: {student.Grade}");



        }
    }
}

public class StudentDisplay
{
    public static void RunProgram()
    {
        // Step 5: Create a StudentList of Student objects
        StudentList studentList = new StudentList();

        // Step 6: Add some students to the list
        studentList.AddStudent(new StudentModel { Name = "John Doe", Age = 20, Grade = "A" });
        studentList.AddStudent(new StudentModel { Name = "Jane Smith", Age = 22, Grade = "B" });
        studentList.AddStudent(new StudentModel { Name = "Alice Brown", Age = 19, Grade = "A" });

        // Step 7: Display all students
        Console.WriteLine("Student List:");
        studentList.DisplayStudents();
    }
}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
Student List:
Name: Sachin Thapa, Age: 20, Grade: A
Name: Ram Bahik, Age: 22, Grade: B
Name: Sita kandel, Age: 19, Grade: A
```

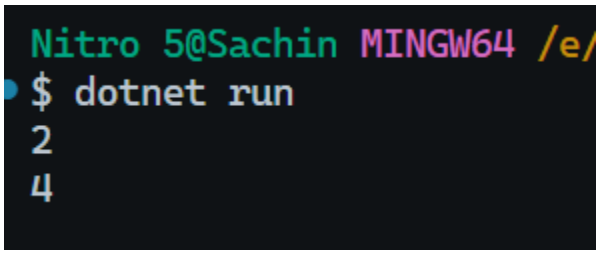
12. Demonstrate Asynchronous programming with async, await. Task in C#.

```
using System;
using System.Threading.Tasks;
namespace AsyncDemo
{
   class AsyncDataHandler
   {
      static async Task<string> GetDataAsync()
      {
         Console.WriteLine("GetDataAsync started...");
         await Task.Delay(3000);
         Console.WriteLine("GetDataAsync finished.");
         return "Hello from async method!";
      }
      public static async Task Run()
      {
         Console.WriteLine("Main started.");
         string result = await GetDataAsync();
         Console.WriteLine($"Result: {result}");
         Console.WriteLine("Main ended.");
      }
   }
}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
Main started.
GetDataAsync started...
GetDataAsync finished.
Result: Hello from async method!
Main ended.
```

13. Write a program to demonstrate the use of the method as a condition in the LINQ.

```
class LINQExample
{
    public void DemonstrateLINQ()
    {

        var numbers = new List<int> { 1, 2, 3, 4, 5 };
        IEnumerable<int> evenNumbers = numbers.Where(n => n % 2 == 0).ToList();
        foreach (var number in evenNumbers)
        {
            Console.WriteLine(number);
        }
    }

}
```



14. Write a program to show the file handing in the C#

```
using System;
using System.IO;

namespace FileHandlingStreamDemo
{
   class FileHandling
   {
      public static void RunProgram()
      {
         string filePath = "./note/file.txt";

         // Step 1: Write to the file using StreamWriter
         using (StreamWriter writer = new StreamWriter(filePath))
         {
            writer.WriteLine("Hello, this is line 1.");
            writer.WriteLine("This is line 2.");
            writer.WriteLine("C# file handling using StreamWriter and StreamReader.");
         }
         Console.WriteLine("Data written to file using StreamWriter.\n");

         // Step 2: Read from the file using StreamReader
         Console.WriteLine("Reading from file using StreamReader:");

         using (StreamReader reader = new StreamReader(filePath))
         {
```

```
            string? line;
            while ((line = reader.ReadLine()) != null)
            {
                Console.WriteLine(line);
            }
        }
    }
  }
}
```

```
Nitro 5@Sachin MINGW64 /e/6th_Practicals/dotnet (main)
$ dotnet run
Data written to file using StreamWriter.

Reading from file using StreamReader:
Hello, this is line 1.
This is line 2.
C# file handling using StreamWriter and StreamReader.
```