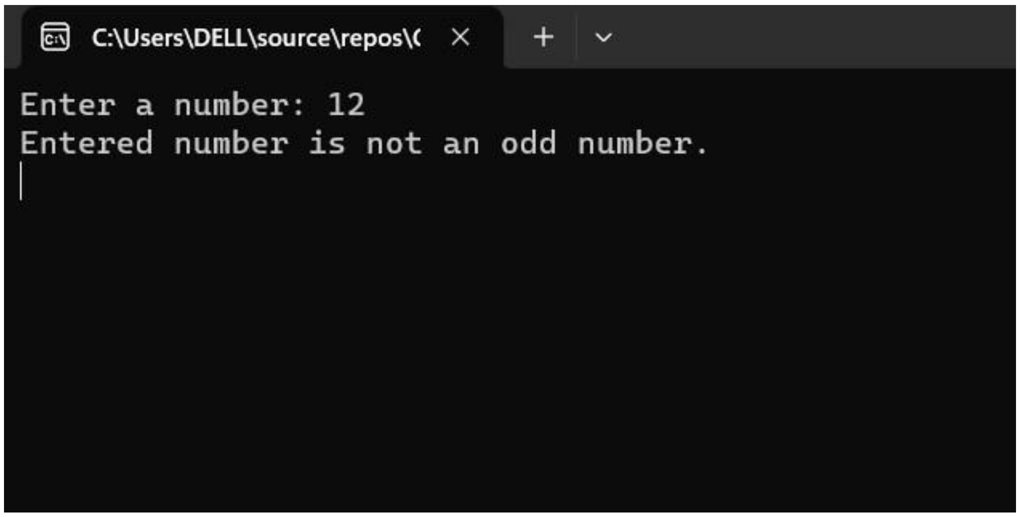
Practical - 5 Exceptions and Generics

(1) Write a program to accept a number from the user and throw an exception if the number is not an odd number.

Code:

```
using System;
using
System.Collections.Generic;
using System.Linq; using
System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
try
            {
                Console.Write("Enter a number: ");
int number = int.Parse(Console.ReadLine());

                if (number % 2 != 1) // Check if the number is not odd
                {
                    throw new ArgumentException("Entered number is not an odd
number.");                  }

                Console.WriteLine("You entered an odd number.");
            }
            catch (FormatException)
            {
                Console.WriteLine("Invalid input. Please enter a valid integer.");
            }
            catch (ArgumentException ex)
            {
                Console.WriteLine(ex.Message);
            }
            Console.ReadLine();
        }
    }
}
```

O/P:



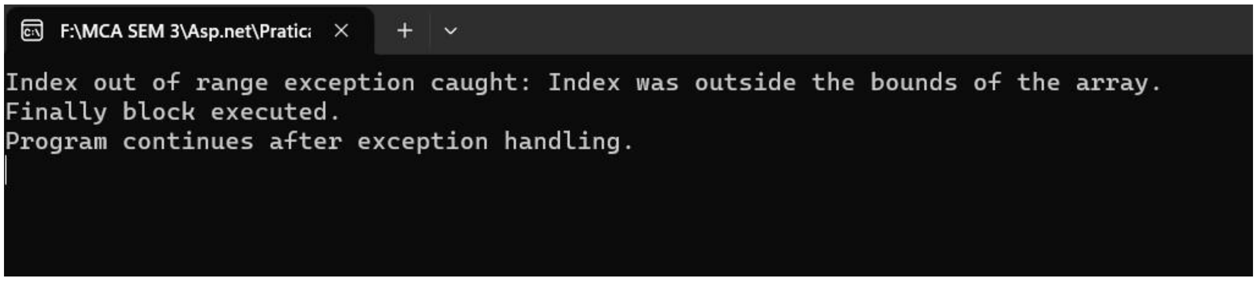
C:\Users\DELL\source\repos\(

```
Enter a number: 12
Entered number is not an odd number.
```

(2) Write a program to illustrate usage of try multiple catch with finally clause.

```
using System;
using
System.Collections.Generic;
using System.Linq; using
System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
try
{
                int[] numbers = { 1, 2, 3 };
int index = 4;
                int result = numbers[index]; // Attempt to access an out-of-range
index             }
            catch (IndexOutOfRangeException ex)
            {
                Console.WriteLine("Index out of range exception caught: " +
ex.Message);            }
            catch (DivideByZeroException ex)
            {
                Console.WriteLine("Divide by zero exception caught: " + ex.Message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("General exception caught: " + ex.Message);
                  }

                finally
                  {
                      Console.WriteLine("Finally block executed.");
              }

              Console.WriteLine("Program continues after exception handling.");
             Console.ReadLine();
          }


      }
  }
```

O/P:



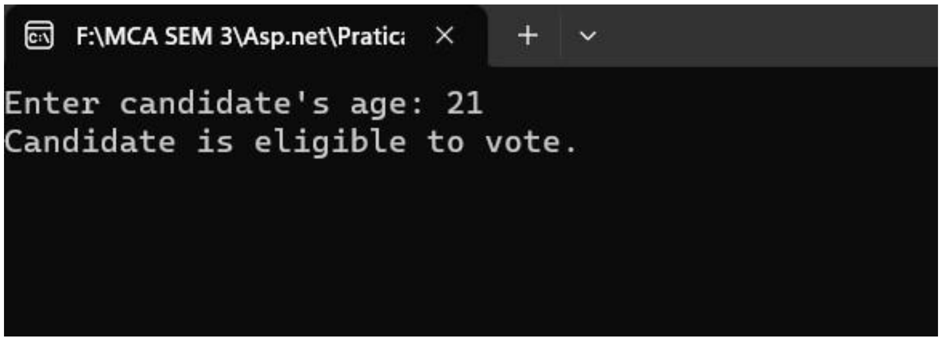
```
Index out of range exception caught: Index was outside the bounds of the array.
Finally block executed.
Program continues after exception handling.
```

(3) Write a program for creation of user defined exception to show whether candidate is eligible to caste vote.

Code:

```
using System;

// Custom exception class for voting eligibility
class NotEligibleToVoteException : Exception
{
    public NotEligibleToVoteException(string message) : base(message)
    {
    }
}
class Program
{
    static void Main(string[] args)
    {
try
        {
            Console.Write("Enter candidate's age: ");
int age = int.Parse(Console.ReadLine());

            if (age < 18)
            {
                throw new NotEligibleToVoteException("Candidate is not eligible to
vote.");

            }
else             {

              Console.WriteLine("Candidate is eligible to vote.");
            }
}
        catch (FormatException)
        {
            Console.WriteLine("Invalid input. Please enter a valid age as a number.");
        }
        catch (NotEligibleToVoteException ex)
        {
            Console.WriteLine(ex.Message);
        }
        Console.ReadLine();
    } }
```

O/P:



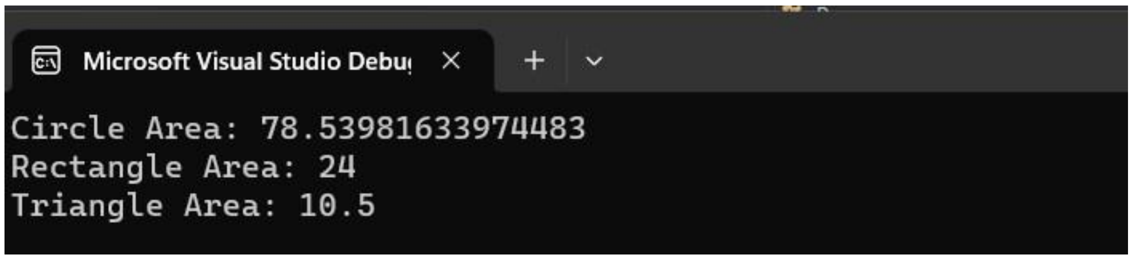
```
Enter candidate's age: 21
Candidate is eligible to vote.
```

(4) Write a program to calculate area of different shapes using Generic delegate.

Code:

```
using System;

// Define a generic delegate for area calculation
delegate double CalculateArea<T>(T shape);

// Create a base class for shapes abstract
class Shape
{     public abstract double
CalculateArea();
}

// Create classes for different shapes (Circle, Rectangle, Triangle) that inherit from
the base class class Circle : Shape
{
    public double Radius { get; set; }
    public Circle(double radius)
    {

       Radius = radius;
    }
    public override double CalculateArea()
    {
        return Math.PI * Math.Pow(Radius, 2);
    }
}
class Rectangle : Shape
{     public double Length { get; set;
}     public double Width { get; set;
}

    public Rectangle(double length, double width)
    {
        Length = length;
        Width = width;
    }
    public override double CalculateArea()
    {
        return Length * Width;
    }
}
class Triangle : Shape
{     public double BaseLength { get; set;
}     public double Height { get; set; }

    public Triangle(double baseLength, double height)
    {
        BaseLength = baseLength;
        Height = height;
    }
    public override double CalculateArea()
    {
        return 0.5 * BaseLength * Height;
    }
}
class Program
{
    static void Main(string[] args)
    {
        // Define a generic delegate instance for area calculation
        CalculateArea<Shape> areaCalculator = CalculateShapeArea;
        // Create instances of different shapes
Shape circle = new Circle(5.0);
        Shape rectangle = new Rectangle(4.0, 6.0);
        Shape triangle = new Triangle(7.0, 3.0);

        // Calculate and display the areas of different shapes using the delegate
                     Console.WriteLine($"Circle Area: {areaCalculator(circle)}");
        Console.WriteLine($"Rectangle Area: {areaCalculator(rectangle)}");
        Console.WriteLine($"Triangle Area: {areaCalculator(triangle)}");
}

    // Generic function to calculate the area of any shape implementing the Shape base
class
    static double CalculateShapeArea(Shape shape)
    {
        return shape.CalculateArea();
        Console.ReadLine();
    }

}
```

O/P:



```
Circle Area: 78.53981633974483
Rectangle Area: 24
Triangle Area: 10.5
```

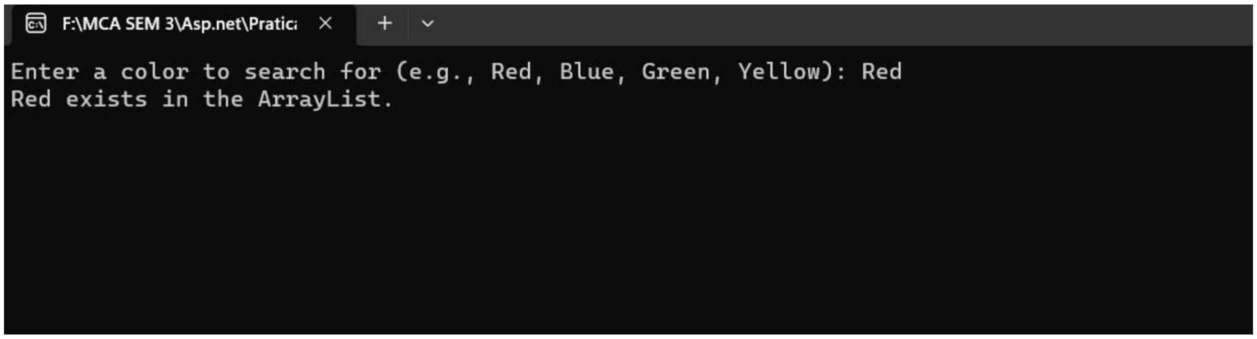
(5) Write a program to search color in given ArrayList of colors.

Code:

```
using System; using
System.Collections;
using System.Drawing; // Required for Color type

class Program
{
    static void Main()
    {
        ArrayList colorList = new ArrayList();
colorList.Add(Color.Red);
colorList.Add(Color.Blue);
colorList.Add(Color.Green);
colorList.Add(Color.Yellow);

        Console.Write("Enter a color to search for (e.g., Red, Blue, Green, Yellow): ");
string inputColor = Console.ReadLine();

        bool colorExists = false;
         foreach (Color color in
colorList)
        {
            if (string.Equals(color.Name, inputColor,
StringComparison.OrdinalIgnoreCase))
            {
                colorExists = true;
break;
```

```
                }
            }
            if (colorExists)
            {
                Console.WriteLine($"{inputColor} exists in the ArrayList.");
            }
    else
            {
                Console.WriteLine($"{inputColor} does not exist in the ArrayList.");
    }
            Console.ReadLine();
        } }
```

O/P:



```
Enter a color to search for (e.g., Red, Blue, Green, Yellow): Red
Red exists in the ArrayList.
```

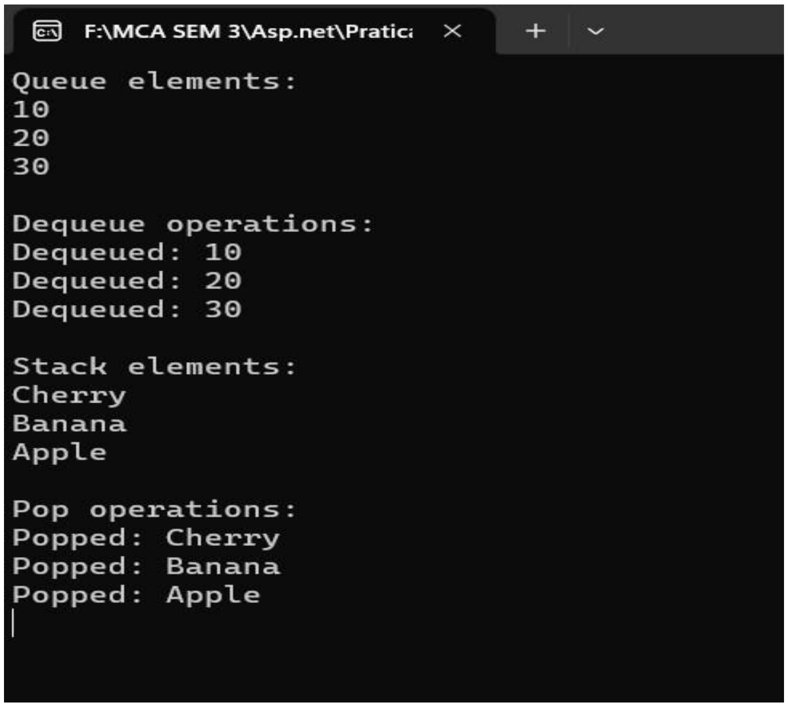
(6) Write a program to create a generic queue/stack and perform insert, delete and display operations on it.

Code:

```
using System;
using System.Collections.Generic;

class Program
{      static void
Main()
    {
        // Create a generic queue of integers
        Queue<int> myQueue = new Queue<int>();

        // Insert elements into the queue
myQueue.Enqueue(10);
myQueue.Enqueue(20);
myQueue.Enqueue(30);
        // Display the queue
        Console.WriteLine("Queue elements:");
foreach (int item in myQueue)
        {

              Console.WriteLine(item);
        }

        // Remove and display elements from the queue
Console.WriteLine("\nDequeue operations:");
while (myQueue.Count > 0)
        {
            int removedItem = myQueue.Dequeue();
Console.WriteLine($"Dequeued: {removedItem}");
        }

        // Create a generic stack of strings
        Stack<string> myStack = new Stack<string>();

        // Insert elements into the stack
myStack.Push("Apple");
myStack.Push("Banana");
myStack.Push("Cherry");

        // Display the stack
        Console.WriteLine("\nStack elements:");
foreach (string item in myStack)
        {
            Console.WriteLine(item);
        }

        // Remove and display elements from the queue
Console.WriteLine("\nPop operations:");         while
(myStack.Count > 0)
        {
            string poppedItem = myStack.Pop();
Console.WriteLine($"Popped: {poppedItem}");
        }
        Console.ReadLine();
    } }
```

O/P:

Practical - 5 Exceptions and Generics



```
Queue elements:
10
20
30

Dequeue operations:
Dequeued: 10
Dequeued: 20
Dequeued: 30

Stack elements:
Cherry
Banana
Apple

Pop operations:
Popped: Cherry
Popped: Banana
Popped: Apple
```

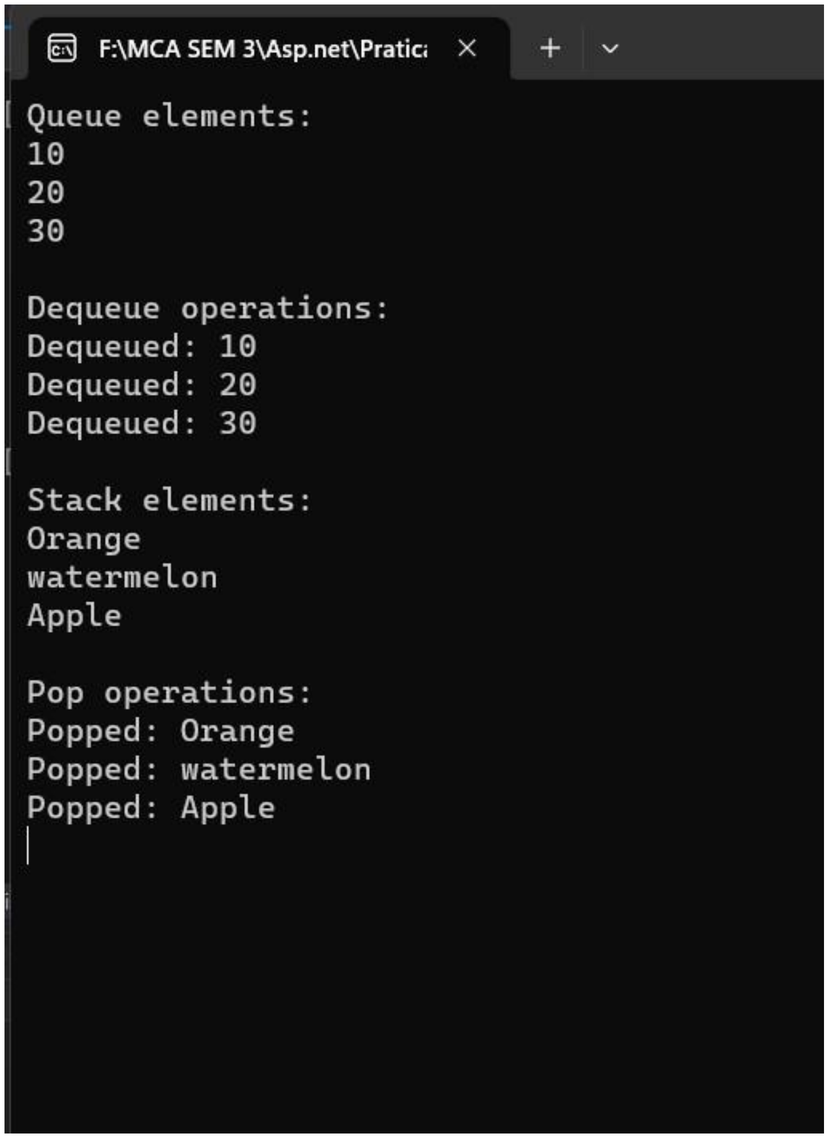
7. Write a program to create a nongeneric queue/stack and perform insert, delete and display operations on it.

Code:

```
using System;
using System.Collections;

class Program
{     static void
Main()
    {
        // Create a non-generic queue
        Queue myQueue = new Queue();

        // Insert elements into the queue
        myQueue.Enqueue(10);
myQueue.Enqueue(20);
myQueue.Enqueue(30);
        // Display the queue
        Console.WriteLine("Queue elements:");
foreach (int item in myQueue)
        {

              Console.WriteLine(item);
        }
        // Remove and display elements from the queue
Console.WriteLine("\nDequeue operations:");
while (myQueue.Count > 0)
        {
            int removedItem = (int)myQueue.Dequeue();
Console.WriteLine($"Dequeued: {removedItem}");
        }

        // Create a non-generic stack
        Stack myStack = new Stack();

        // Insert elements into the stack
myStack.Push("Apple");
myStack.Push("watermelon");
myStack.Push("Orange");

        // Display the stack
        Console.WriteLine("\nStack elements:");
foreach (string item in myStack)
        {
            Console.WriteLine(item);
        }

        // Remove and display elements from the stack
Console.WriteLine("\nPop operations:");         while
(myStack.Count > 0)
        {
            string poppedItem = (string)myStack.Pop();
Console.WriteLine($"Popped: {poppedItem}");
        }
        Console.ReadLine();
    }
}
```

O/P:

Practical - 5 Exceptions and Generics



F:\MCA SEM 3\Asp.net\Pratic

```
Queue elements:
10
20
30

Dequeue operations:
Dequeued: 10
Dequeued: 20
Dequeued: 30

Stack elements:
Orange
watermelon
Apple

Pop operations:
Popped: Orange
Popped: watermelon
Popped: Apple
```