GLA UNIVERSITY
ESTD. 2010
ऋते ज्ञानान्न मुक्तिः

GLA
UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

Accredited with A+ Grade by NAAC
3.46 Score

12-B Status from UGC

(SESSION 2023-2024)

MCA –IIIrd SEMESTER

PRACTICAL FILE – .Net Framework using C#Sharp

SUBMITTED TO:	**SUBMITTED BY:**
DR. Sachendra Singh Chauhan	NAME – Mohd Adnan
(ASSISTANT PROFESSOR)	SECTION – B ROLL NO. – 34

DEPT. OF C.E.A. UNIV. ROLL NO. - 2284200120

DATE :- 25-oct-2023

Lab Assignment :- 5

Ques:-1 Create a C# program that intentionally throws a DivideByZeroException when dividing by zero. Catch the exception and handle it gracefully.

Ans:-
```
using System;


class Program
{
    static void Main()
    {
        try
        {
            // Intentionally dividing by zero
            int numerator = 10;
            int denominator = 0;
            int result = DivideByZeroExample(numerator,
denominator);

            Console.WriteLine($"Result: {result}");
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine($"Exception caught: {ex.Message}");
            // Handle the exception gracefully, for example, by
providing a default result
            Console.WriteLine("Default result: 0");
        }
    }

    static int DivideByZeroExample(int numerator, int
denominator)
    {
        if (denominator == 0)
        {
            throw new DivideByZeroException("Cannot divide by
zero.");
        }


        return numerator / denominator;
    }
}
```

Ques:-2 Write a program that attempts to access an array element at an index that is out of bounds. Use a try-catch block to handle the IndexOutOfRangeException.

Ans:- using System;

```
class Program
{
    static void Main()
    {
        int[] numbers = { 1, 2, 3, 4, 5 };
        try
        {
            // Attempting to access an array element at an out-of-
bounds index
            int index = 10;
            int value = numbers[index];
            Console.WriteLine($"Value at index {index}: {value}");
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine($"Exception caught: {ex.Message}");
            // Handle the exception gracefully, for example, by
providing a default value
            Console.WriteLine("Default value: 0");
        }
    }
}
```

Ques:-3 Create a C# program that uses a try-catch block to handle an exception when converting a string to an integer using int.Parse(). Handle the FormatException that may occur.

Ans:-
```
using System;


class Program
{
    static void Main()
    {
        try
        {
            // Attempting to convert a string to an integer
            string input = "123abc";
            int number = ConvertStringToInt(input);
            Console.WriteLine($"Converted number: {number}");
        }
        catch (FormatException ex)
        {
            Console.WriteLine($"Exception caught: {ex.Message}");
            // Handle the exception gracefully, for example, by
providing a default value
            Console.WriteLine("Default value: 0");
        }
    }

    static int ConvertStringToInt(string input)
    {
        return int.Parse(input);
    }
}
```

Ques:-4 Implement a C# program that uses a custom exception class. Create a custom exception and throw it in your code when a specific condition is met.

Ans:- using System;

```
// Custom exception class
```

```
public class CustomException : Exception
{
    public CustomException(string message) : base(message)
    {
    }
}

class Program
{
    static void Main()
    {
        try
        {
            // Simulating a condition that triggers the custom
exception
            int value = 5;
            if (value < 10)
            {
```

```
            throw new CustomException("Custom exception:
Value is less than 10.");
        }


        Console.WriteLine($"Value: {value}");
      }
      catch (CustomException ex)
      {
        Console.WriteLine($"Exception caught: {ex.Message}");
        // Handle the custom exception gracefully, for example,
by providing a default value
        Console.WriteLine("Default value: 0");
      }
   }
}
```

Ques:-5 Build a C# program that demonstrates the use of multiple catch blocks for different exception types. Handle exceptions such as IndexOutOfRangeException, FormatException, and InvalidOperationException.

Ans:-
```
using System;

class Program
{
    static void Main()
    {
        try
        {
            // Simulating various scenarios that may throw different
exceptions
            // Scenario 1: IndexOutOfRangeException
            int[] numbers = { 1, 2, 3 };
            int index = 10;
            int value = numbers[index];

            // Scenario 2: FormatException
            string input = "abc";
            int parsedValue = int.Parse(input);
            // Scenario 3: InvalidOperationException
            throw new InvalidOperationException("Invalid operation
occurred.");
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine($"IndexOutOfRangeException caught:
{ex.Message}");
            // Handle the exception specifically for
IndexOutOfRangeException
        }
        catch (FormatException ex)
        {
            Console.WriteLine($"FormatException caught:
{ex.Message}");
            // Handle the exception specifically for FormatException
        }
        catch (InvalidOperationException ex)
        {
            Console.WriteLine($"InvalidOperationException caught:
{ex.Message}");
            // Handle the exception specifically for
InvalidOperationException
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Generic Exception caught:
{ex.Message}");
            // Handle any other exceptions that were not caught
specifically
        }
    }
}
```

Ques:-6 Create a C# program that includes nested try-catch blocks. Throw an exception in an inner block and catch it in the outer block. Explain the flow of execution.

Ans:- using System;

```
class Program
{
    static void Main()
    {
        try
        {
            Console.WriteLine("Outer try block starts.");

            try
            {
                Console.WriteLine("Inner try block starts.");

                // Simulating an exception in the inner block
                int result = Divide(10, 0);

                Console.WriteLine($"Result: {result}"); // This line
won't be executed
            }
            catch (DivideByZeroException innerException)
            {
                Console.WriteLine($"Inner catch block caught:
{innerException.Message}");
                // Handle the exception at the inner level
            }

            Console.WriteLine("Inner try block ends.");
        }
        catch (Exception outerException)
        {
            Console.WriteLine($"Outer catch block caught:
{outerException.Message}");
            // Handle the exception at the outer level
        }

        Console.WriteLine("Outer try block ends.");
    }
    static int Divide(int numerator, int denominator)
    {
        // Attempting to divide by zero
        return numerator / denominator;
    }
}
```

Ques:-7 Implement a program that divides two numbers entered by the user. Handle exceptions like division by zero and invalid input. Continue to prompt the user for valid input until a valid division is performed.

Ans:-
```
using System;

class Program
{
    static void Main()
    {
        bool isValidInput = false;
        int numerator = 0, denominator = 0, result = 0;
    do
    {
        try
        {
            Console.Write("Enter the numerator: ");
            numerator = int.Parse(Console.ReadLine());

            Console.Write("Enter the denominator: ");
            denominator = int.Parse(Console.ReadLine());

            // Attempting to perform the division
            result = DivideNumbers(numerator, denominator);
            isValidInput = true;
        }
        catch (FormatException)
        {
            Console.WriteLine("Invalid input. Please enter valid
integers.");
```

```
            }
            catch (DivideByZeroException)
            {
                Console.WriteLine("Cannot divide by zero. Please
enter a non-zero denominator.");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"An unexpected error occurred:
{ex.Message}");
            }
        } while (!isValidInput);

        Console.WriteLine($"Result of {numerator} /
{denominator} = {result}");
    }

    static int DivideNumbers(int numerator, int denominator)
    {
        if (denominator == 0)
        {
            throw new DivideByZeroException("Cannot divide by
zero.");
        }

        return numerator / denominator;
    }
}
```

Ques:-8 Develop a C# program that demonstrates how to use the throw statement to rethrow an exception. Catch the rethrown exception and handle it appropriately.

Ans:- using System;

```
class Program
{
    static void Main()
    {
        try
        {
            // Simulating an initial exception
            ProcessData();
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Caught in Main:
{ex.GetType().Name} - {ex.Message}");
        }
    }

    static void ProcessData()
    {
        try
        {
            // Simulating an exception
            int result = Divide(10, 0);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Caught in ProcessData:
{ex.GetType().Name} - {ex.Message}");

            // Rethrowing the exception
            Console.WriteLine("Rethrowing the exception...");
            throw;
        }
    }

    static int Divide(int numerator, int denominator)
    {
        if (denominator == 0)
        {
            throw new DivideByZeroException("Cannot divide by
zero.");
        }
        return numerator / denominator;
    }
}
```

Ques:-9 Develop a program that simulates a simple calculator with basic arithmetic operations (addition, subtraction, multiplication, and division). Use exception handling to catch and handle various type of exceptions that may occur.

Ans:- using System;

```
class Calculator
{
    static void Main()
    {
        try
        {
            Console.WriteLine("Simple Calculator");

            // Get input from the user
            Console.Write("Enter the first number: ");
            double num1 = GetValidNumberInput();

            Console.Write("Enter the operation (+, -, *, /): ");
            char operation = GetValidOperationInput();

            Console.Write("Enter the second number: ");
            double num2 = GetValidNumberInput();

            // Perform the calculation based on the chosen
operation
            double result = Calculate(num1, operation, num2);

            // Display the result
            Console.WriteLine($"Result: {num1} {operation} {num2}
= {result}");
        }
        catch (FormatException)
        {
            Console.WriteLine("Invalid input. Please enter valid
numbers.");
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Cannot divide by zero. Please enter a
non-zero divisor.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An unexpected error occurred:
{ex.Message}");
        }
    }

    static double GetValidNumberInput()
    {
        string input = Console.ReadLine();
        if (!double.TryParse(input, out double number))
        {
            throw new FormatException("Invalid number format.");
        }
        return number;
    }

    static char GetValidOperationInput()
    {
        char operation = char.Parse(Console.ReadLine());
        if (operation != '+' && operation != '-' && operation != '*'
&& operation != '/')
        {
            throw new InvalidOperationException("Invalid
operation. Please enter +, -, *, or /.");
        }
        return operation;
    }
    static double Calculate(double num1, char operation, double
num2)
    {
        switch (operation)
        {
            case '+':
                return num1 + num2;
            case '-':
                return num1 - num2;
            case '*':
                return num1 * num2;
            case '/':
                if (num2 == 0)
                {
                    throw new DivideByZeroException("Cannot divide
by zero.");
                }
                return num1 / num2;
            default:
```

```
            throw new InvalidOperationException("Invalid
operation.");
        }
    }
}
```

Ques:-10 Scenario You are developing a simple e-commerce application in C#. One of the features is a shopping cart that allows users to add items to their cart. The cart is represented as an array of integers, where each integer corresponds to an item's price. Users can input the price of an item they want to add to the cart. You want to handle exceptions gracefully to ensure a smooth user experience. If the user enters an invalid price, your code should catch and handle the exception appropriately. Question: Write a C# program that simulates adding items to a shopping cart. The program should take user input for the price of items and store them in an array. Implement exception handling with multiple catch blocks to handle various scenarios. Specifically, you should handle the following.

Ans:-
```
using System;
```

```
class ShoppingCart
{
    static void Main()
    {
        try
        {
            Console.WriteLine("Shopping Cart Simulator");

            // Set the maximum number of items in the cart
            int maxItems = 5;
            int[] cart = new int[maxItems];
            int itemCount = 0;

            // Allow users to add items to the cart
            while (itemCount < maxItems)
            {
                Console.Write($"Enter the price for item {itemCount +
1}: ");
```

```
            int price = GetValidPriceInput();

            // Add the price to the cart
            cart[itemCount] = price;
            itemCount++;

            Console.WriteLine("Item added to the cart.");
        }

        // Display the items in the cart
        Console.WriteLine("Items in the cart:");
        for (int i = 0; i < itemCount; i++)
        {
            Console.WriteLine($"Item {i + 1}: ${cart[i]}");
        }
    }
    catch (FormatException)
    {
            Console.WriteLine("Invalid input. Please enter a valid
price (numeric value).");
        }
        catch (OverflowException)
        {
            Console.WriteLine("Price entered is too large. Please
enter a smaller value.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An unexpected error occurred:
{ex.Message}");
        }
    }

    static int GetValidPriceInput()
    {
        string input = Console.ReadLine();
        if (!int.TryParse(input, out int price) || price < 0)
        {
            throw new FormatException("Invalid price format or
negative price.");
        }

        return price;
    }
}
```

- .