

# ASSIGNMENT 3

Software Tools and Techniques for CSE

Kunal Singh Rathore & Gurudayal Meena

23110181 & 23110125

GITHUB LINK:

[https://github.com/kunalsinghrathore23110181/STT\\_A3.git](https://github.com/kunalsinghrathore23110181/STT_A3.git)

# TABLE OF CONTENTS

## Laboratory Session 9

Introduction.....	1
Tools.....	1
Setup.....	1
Methodology and Execution.....	2
Results and Analysis.....	14
Discussion and Conclusion.....	14
References.....	14

## Laboratory Session 10

Introduction.....	15
Tools.....	15
Setup.....	15
Methodology and Execution.....	17
Results and Analysis.....	28
Discussion and Conclusion.....	28
References.....	29

# Laboratory Session 9

## Introduction:

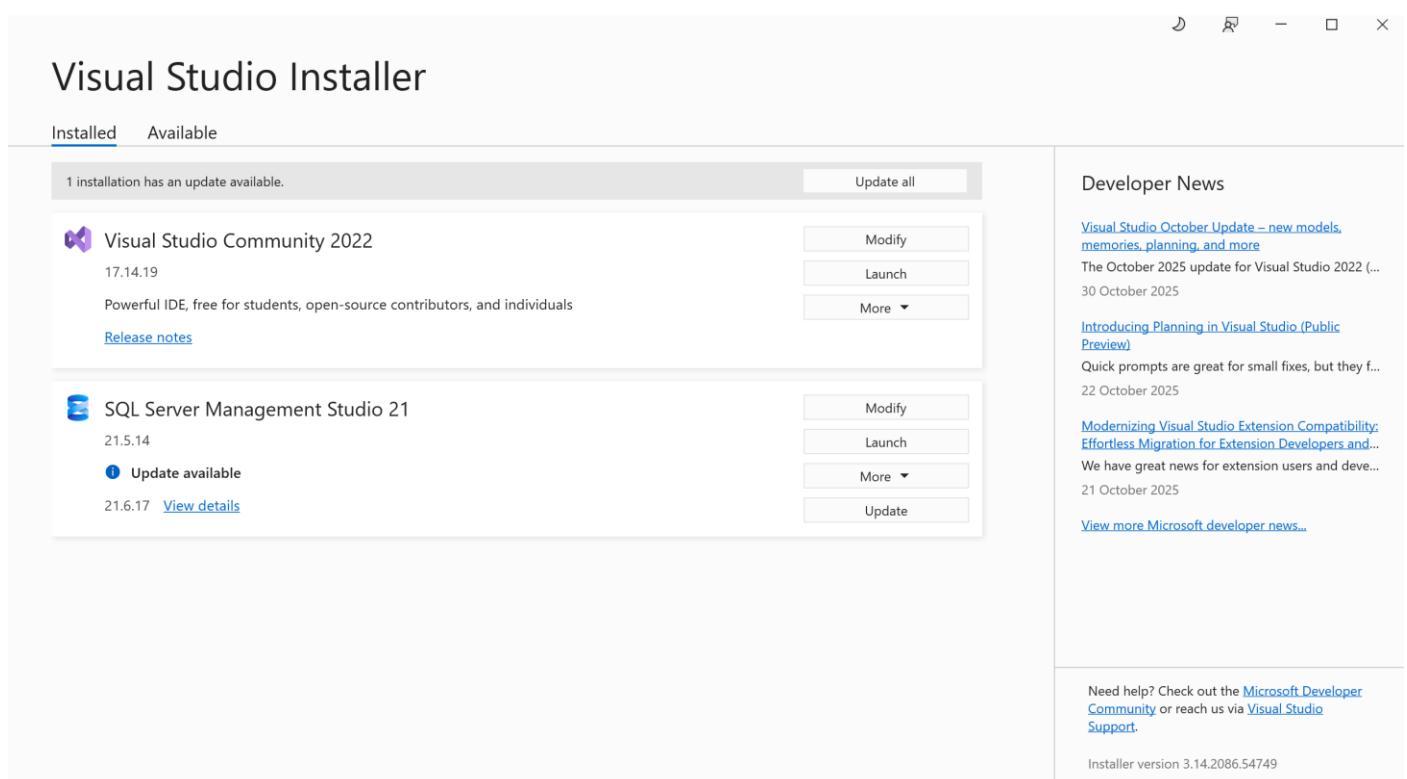
This lab, conducted on **13th October 2025**, focuses on the development of C# console applications using the .NET framework in Visual Studio. It introduces to the core programming concepts such as data types, control structures, and object-oriented design principles in C#. The exercises guide in performing arithmetic operations, using conditional statements, and applying looping constructs effectively. It also explores the use of functions for modular programming and understands the role of static methods. Further, the lab includes tasks on implementing sorting algorithms, working with single and multi-dimensional arrays, and performing matrix operations. We are encouraged to analyze the reasoning behind program outputs to strengthen their conceptual understanding. Overall, this lab builds a solid foundation in C# programming and prepares students for more advanced software development using .NET technologies.

## TOOLS:

The lab was conducted on the Windows operating system using Visual Studio 2022 (Community Edition) as the primary Integrated Development Environment (IDE). The .NET SDK was used to compile, build, and execute C# programs efficiently. Programming was carried out in the C# language using the .NET 6 framework or later versions. The Visual Studio environment provided powerful features such as IntelliSense, debugging tools, and code suggestions to enhance the coding experience. Built-in project templates simplified the creation of console applications and ensured proper configuration of dependencies. The integrated terminal and output console in Visual Studio helped in testing and observing program behaviour in real time. Overall, this toolset offered a robust and user-friendly platform for learning and practicing .NET-based C# programming.

## SETUP:

I install and setup the visual studio in my windows with the link "<https://visualstudio.microsoft.com/vs/community/>"



Now in the Visual Studio I make setup the new project and name the file STT\_LAB9 and kept the version of Visual Studio with .NET SDK and my target framework is .NET 8

## Configure your new project

Console App C# Linux macOS Windows Console

Project name

STT\_LAB\_9

Location

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT\_A3

...

Solution

Create new solution

Solution name ⓘ

STT\_LAB\_9

Place solution and project in the same directory

Project will be created in "C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT\_A3  
\STT\_LAB\_9"

Back

Next

Testing the setup by print the “Hello, World!”. The screenshots are attached below:

The screenshot shows the Microsoft Visual Studio interface. In the center-left, the code editor displays a file named Program.cs with the following content:

```
1 namespace STT_LAB9
2 {
3     internal class Program
4     {
5         static void Main(string[] args)
6         {
7             Console.WriteLine("Hello, World!");
8         }
9     }
10 }
```

To the right of the code editor, the Output window shows the results of the application's execution:

```
Hello, World!
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 28180) exited
with code 0 (0x0).
Press any key to close this window . . .
```

The status bar at the bottom right indicates the date and time: 03-11-2025 20:02.

## METHODOLOGY AND EXECUTION:

### ❖ Understanding Basic Syntax and Control Structures

➤ Write an object-oriented C# program that:

- Accepts user input for two numbers.
- Performs addition, subtraction, multiplication, and division.
- Uses if-else conditions to determine if the sum is even or odd.
- Displays the results using `Console.WriteLine()`.

The screenshot shows the Microsoft Visual Studio interface. The code editor displays a C# file named Program.cs with the following content:

```
1  using System;
2
3  class Calculator
4  {
5      static void Main()
6      {
7          Console.Write("Enter first number: ");
8          int num1 = Convert.ToInt32(Console.ReadLine());
9
10         Console.Write("Enter second number: ");
11         int num2 = Convert.ToInt32(Console.ReadLine());
12
13         int sum = num1 + num2;
14         int diff = num1 - num2;
15         int prod = num1 * num2;
16         double div = (double)num1 / num2;
17
18         Console.WriteLine($"Sum = {sum}");
19         Console.WriteLine($"Difference = {diff}");
20         Console.WriteLine($"Product = {prod}");
21         Console.WriteLine($"Division = {div}");
22
23         if (sum % 2 == 0)
24             Console.WriteLine("The sum is even.");
25         else
26             Console.WriteLine("The sum is odd.");
27     }
28 }
```

The Solution Explorer window on the right shows a project named 'STT\_LAB9' with a single file 'Program.cs'. The Output window at the bottom shows the build log and the execution output.

### Output:

The Output window shows the following text:

```
Microsoft Visual Studio Debug X + ▾
Enter first number: 10
Enter second number: 5

Sum = 15
Difference = 5
Product = 50
Division = 2
The sum is odd.

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 2837
6) exited with code 0 (0x0).
Press any key to close this window . . .
```

### Explanation:

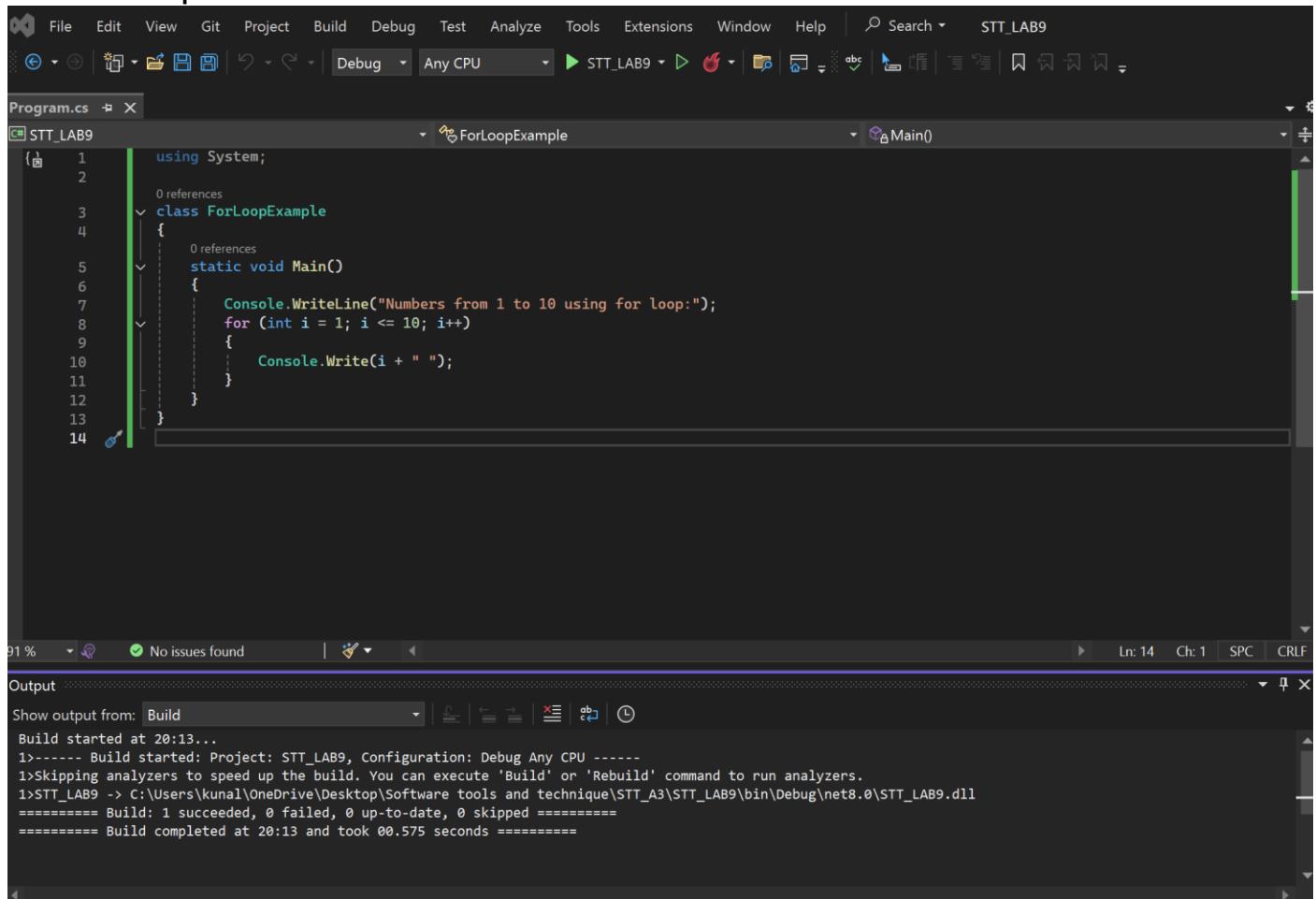
In this program, I created a C# class named **Calculator** that demonstrates basic object-oriented programming. Inside the `Main()` method, the program accepts two integer inputs from the user using `Console.ReadLine()` and converts them to integers using `Convert.ToInt32()`.

The program then performs four arithmetic operations: **addition**, **subtraction**, **multiplication**, and **division**. Each result is stored in separate variables (sum, diff, prod, div) and displayed on the screen using `Console.WriteLine()` statements.

Next, an **if-else** condition checks whether the calculated sum is even or odd. If the remainder when dividing by 2 (`sum % 2`) is zero, the program displays that the sum is even; otherwise, it displays that the sum is odd.

## ❖ Using Loops and Functions

- Design an object-oriented C# program that:
  - **For Loop: Print Numbers from 1 to 10**



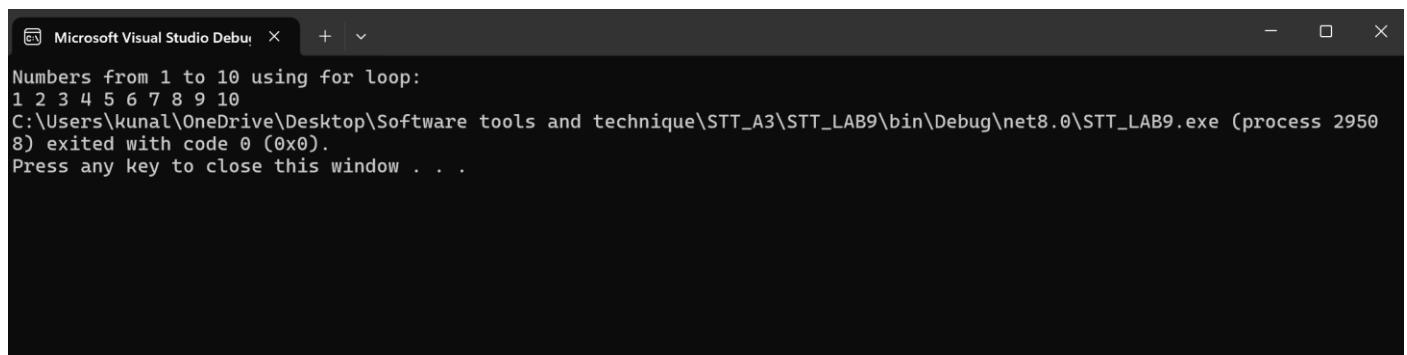
A screenshot of Microsoft Visual Studio showing the code for a for loop example. The code is as follows:

```
1 using System;
2
3 class ForLoopExample
4 {
5     static void Main()
6     {
7         Console.WriteLine("Numbers from 1 to 10 using for loop:");
8         for (int i = 1; i <= 10; i++)
9         {
10            Console.Write(i + " ");
11        }
12    }
13 }
```

The Output window shows the build log and the console output. The console output is:

```
Build started at 20:13...
1>----- Build started: Project: STT_LAB9, Configuration: Debug Any CPU -----
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>STT_LAB9 -> C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Build completed at 20:13 and took 0.575 seconds =====
```

### Output:



A screenshot of the Microsoft Visual Studio debugger showing the console output. The output is:

```
Numbers from 1 to 10 using for loop:
1 2 3 4 5 6 7 8 9 10
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 2950
8) exited with code 0 (0x0).
Press any key to close this window . . .
```

### Explanation:

This program defines a class **ForLoopExample** that demonstrates the use of a **for loop** in C#.

Inside the `Main()` method, the program first displays a message indicating the task.

A for loop is then used to print numbers from **1 to 10** sequentially. The loop starts with `i = 1`, checks the condition `i <= 10`, and increments `i` after each iteration. Each number is printed on the same line using `Console.Write()`. The loop stops automatically when the condition becomes false.

- **Foreach Loop: Print Numbers from 1 to 10.**

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search. The title bar says "STT LAB9". The left sidebar shows the "Program.cs" file under "STT LAB9". The code in "Program.cs" is:

```

1  using System;
2
3  class ForEachLoopExample
4  {
5      static void Main()
6      {
7          int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
8
9          Console.WriteLine("Numbers from 1 to 10 using foreach loop:");
10         foreach (int num in numbers)
11         {
12             Console.Write(num + " ");
13         }
14     }
15 }

```

The right side of the screen shows the "Output" window with the following build log:

```

Build started at 20:14...
1>----- Build started: Project: STT_LAB9, Configuration: Debug Any CPU -----
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>STT_LAB9 -> C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.dll
1>Build: succeeded 0 failed, 0 up-to-date, 0 skipped
===== Build completed at 20:14 and took 00.528 seconds =====

```

### Output:

The screenshot shows the "Microsoft Visual Studio Debug" output window. It displays the following text:

```

Numbers from 1 to 10 using foreach loop:
1 2 3 4 5 6 7 8 9 10
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 2885)
2) exited with code 0 (0x0).
Press any key to close this window . . .

```

### Explanation:

This program defines a class `ForEachLoopExample` that demonstrates the use of a `foreach` loop in C#. An integer array named `numbers` is created containing values from 1 to 10. The `foreach` loop is used to iterate through each element of the array. During each iteration, the current number is displayed using `Console.WriteLine()`. The loop automatically accesses all elements without using an index variable. This program shows how `foreach` simplifies iteration over collections or arrays. It efficiently prints all numbers from 1 to 10 in a single loop structure.

- **Do-While Loop: Keep Asking Until User Types “exit”**

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and Search. The title bar says "STT LAB9". The left sidebar shows the "Program.cs" file under "STT LAB9". The code in "Program.cs" is:

```

1  using System;
2
3  class DoWhileExample
4  {
5      static void Main()
6      {
7          string input;
8          do
9          {
10              Console.Write("Enter something (type 'exit' to stop): ");
11              input = Console.ReadLine();
12          } while (input.ToLower() != "exit");
13
14          Console.WriteLine("Program ended.");
15      }
16 }

```

The right side of the screen shows the "Output" window with the following build log:

```

Build started at 20:16...
1>----- Build started: Project: STT_LAB9, Configuration: Debug Any CPU -----
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>STT_LAB9 -> C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.dll
1>Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
===== Build completed at 20:16 and took 00.331 seconds =====

```

## Output:

```
Microsoft Visual Studio Debug + | - □ ×  
Enter something (type 'exit' to stop): Hello  
Enter something (type 'exit' to stop): World  
Enter something (type 'exit' to stop): STT_LAB9  
Enter something (type 'exit' to stop): exit  
Program ended.  
  
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 1732)  
8) exited with code 0 (0x0).  
Press any key to close this window . . .
```

## **Explanation:**

This program defines a class `DoWhileExample` that demonstrates the use of a do-while loop in C#. Inside the `Main()` method, a string variable `input` is declared to store user input. The program repeatedly asks the user to enter text using `Console.ReadLine()`. The loop continues until the user types “exit”, regardless of case sensitivity. The `ToLower()` function ensures that both “Exit” and “EXIT” are accepted. The do-while loop executes at least once before checking the condition. Finally, when the user types “exit”, the loop stops and the program displays “Program ended.”

- Defines and calls a function that calculates the factorial of a number provided by the user.

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search bar. Below the menu is a toolbar with various icons. The main workspace displays the code for 'FactorialExample.cs' under the project 'STT\_LAB9'. The code defines a static void Main() method that prompts the user for a number and prints its factorial. It also contains a static int Factorial(int num) function that calculates the factorial using a for loop. The bottom left shows a status bar with '75 %' and 'No issues found'. The bottom right shows the 'Output' window.

```
using System;
class FactorialExample
{
    static void Main()
    {
        Console.Write("Enter a number to find its factorial: ");
        int n = Convert.ToInt32(Console.ReadLine());

        int result = Factorial(n);
        Console.WriteLine($"Factorial of {n} is: {result}");
    }

    // Static function definition
    static int Factorial(int num)
    {
        int fact = 1;
        for (int i = 1; i <= num; i++)
        {
            fact *= i;
        }
        return fact;
    }
}
```

## Output:

```
Microsoft Visual Studio Debug + ▾
Enter a number to find its factorial: 5
Factorial of 5 is: 120
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 2879
6) exited with code 0 (0x0).
Press any key to close this window . . .
```

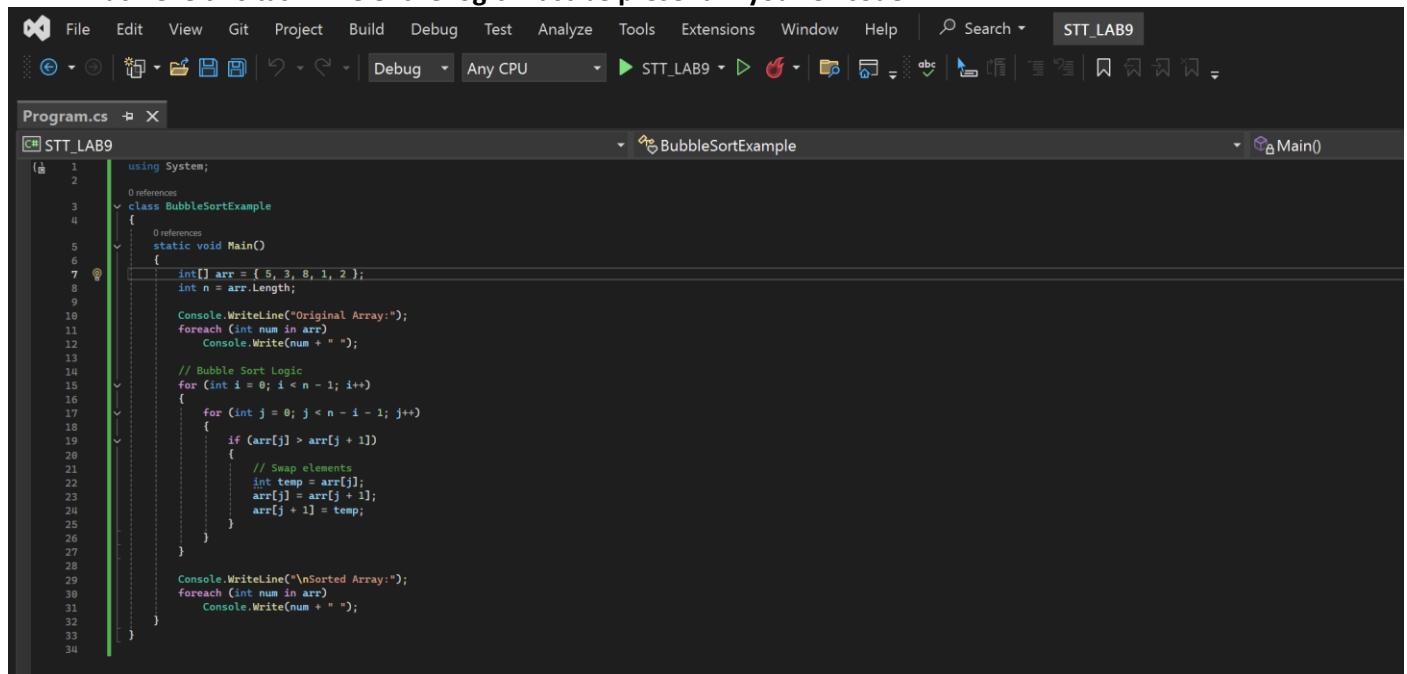
## Explanation:

This program defines a class **FactorialExample** that calculates the factorial of a given number. Inside the **Main()** method, the user is prompted to enter a number using **Console.ReadLine()**. The program then calls a static function named **Factorial()** to compute the factorial. The **Factorial()** function uses a **for** loop to multiply numbers from 1 to n. The result is returned to the main function and displayed using **Console.WriteLine()**. This demonstrates the use of functions, loops, and user input handling in C#.

## ❖ Using Single and Multi-dimensional Arrays

- Design an object-oriented C# program that:

- **Implements the bubble sort algorithm on array3 of integers. You should not use any library functions to achieve this task. The entire logic must be present in your C# code.**



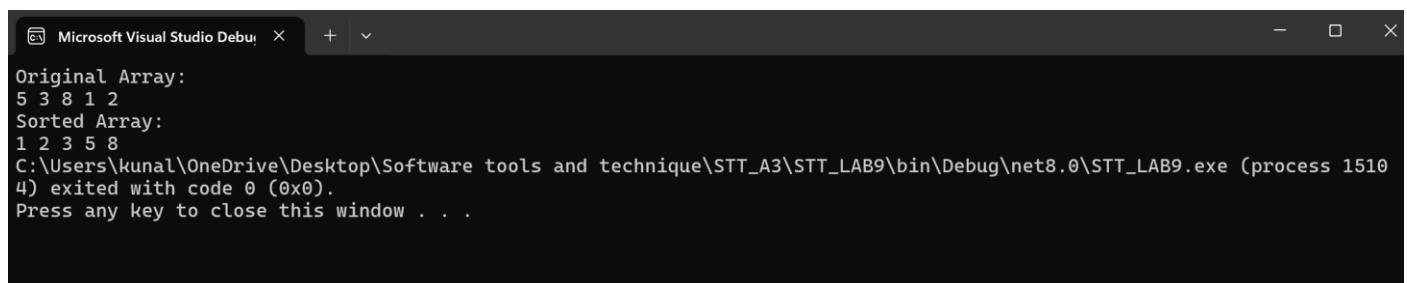
```
using System;
class BubbleSortExample
{
    static void Main()
    {
        int[] arr = { 5, 3, 8, 1, 2 };
        int n = arr.Length;

        Console.WriteLine("Original Array:");
        foreach (int num in arr)
            Console.Write(num + " ");

        // Bubble Sort Logic
        for (int i = 0; i < n - 1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (arr[j] > arr[j + 1])
                {
                    // Swap elements
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }

        Console.WriteLine("\nSorted Array:");
        foreach (int num in arr)
            Console.Write(num + " ");
    }
}
```

## Output:



```
Microsoft Visual Studio Debug
Original Array:
5 3 8 1 2
Sorted Array:
1 2 3 5 8
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 1510
4) exited with code 0 (0x0).
Press any key to close this window . . .
```

## Explanation:

This program defines a class **BubbleSortExample** that demonstrates the bubble sort algorithm in C#. An integer array **arr** is initialized with unsorted elements. The program first displays the original array using a **foreach loop**. Two nested **for loops** are used to compare and swap adjacent elements in ascending order. If a larger element is found before a smaller one, they are swapped using a temporary variable. This process repeats until the entire array is sorted. Finally, the sorted array is printed.

- Stores a 2-D array into one 1-D array in: (i) row major order, (ii)column major order.

```

1  using System;
2  3  class TwoDToOneD
4  {
5      static void Main()
6      {
7          int[,] matrix = { { 1, 2, 3 }, { 4, 5, 6 } };
8          int rows = matrix.GetLength(0);
9          int cols = matrix.GetLength(1);
10         int[] rowMajor = new int[rows * cols];
11         int[] colMajor = new int[rows * cols];
12         int k = 0;
13
14         // Row-major order
15         for (int i = 0; i < rows; i++)
16         {
17             for (int j = 0; j < cols; j++)
18             {
19                 rowMajor[k++] = matrix[i, j];
20             }
21         }
22
23         // Column-major order
24         for (int j = 0; j < cols; j++)
25         {
26             for (int i = 0; i < rows; i++)
27             {
28                 colMajor[k++] = matrix[i, j];
29             }
30         }
31
32         Console.WriteLine("Original 2D Array:");
33         for (int i = 0; i < rows; i++)
34         {
35             for (int j = 0; j < cols; j++)
36             {
37                 Console.Write(matrix[i, j] + " ");
38             }
39             Console.WriteLine();
40         }
41
42         Console.WriteLine("Row-Major Order (1D): " + string.Join(" ", rowMajor));
43         Console.WriteLine("Column-Major Order (1D): " + string.Join(" ", colMajor));
44     }
45 }

```

No issues found

Output

```

Build output from: Build
Build started at 20:29...
1>----- Build started: Project: STT_LAB9, Configuration: Debug Any CPU -----
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>STT_LAB9 -> C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Build completed at 20:29 and took 00.586 seconds =====

```

## Output:

```

Original 2D Array:
1 2 3
4 5 6

Row-Major Order (1D): 1 2 3 4 5 6
Column-Major Order (1D): 1 4 2 5 3 6

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 2810
4) exited with code 0 (0x0).
Press any key to close this window . . .

```

## Explanation:

This program defines a class **TwoDToOneD** that converts a 2D array into a 1D array.

A 2D integer array named **matrix** is initialized with sample values.

The program first prints the original 2D array using nested **for loops**.

It then stores the elements in a 1D array using **row-major order** (left to right, top to bottom).

Next, it repeats the process in **column-major order** (top to bottom, left to right).

The results of both conversions are displayed using **Console.WriteLine()**.

- Performs matrix multiplication of two matrices A and B and stores the resultant in matrix C and print it.

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search dropdown. The title bar says "STT LAB9". The toolbar contains various icons for file operations like Open, Save, Print, and Build. The status bar at the bottom shows "51 %", "No issues found", and a small icon.

The main area displays the "Program.cs" file under the "STT\_LAB9" project. The code implements matrix multiplication logic:

```

1  using System;
2
3  class MatrixMultiplication
4  {
5      static void Main()
6      {
7          int[,] A = { { 1, 2, 3 }, { 4, 5, 6 } };
8          int[,] B = { { 7, 8 }, { 9, 10 }, { 11, 12 } };
9
10         int rowsA = A.GetLength(0);
11         int colsA = A.GetLength(1);
12         int rowsB = B.GetLength(0);
13         int colsB = B.GetLength(1);
14
15         // Check if multiplication is possible
16         if (colsA != rowsB)
17         {
18             Console.WriteLine("Matrix multiplication not possible!");
19             return;
20         }
21
22         int[,] C = new int[rowsA, colsB];
23
24         // Matrix Multiplication Logic
25         for (int i = 0; i < rowsA; i++)
26         {
27             for (int j = 0; j < colsB; j++)
28             {
29                 for (int k = 0; k < colsA; k++)
30                 {
31                     C[i, j] += A[i, k] * B[k, j];
32                 }
33             }
34         }
35
36         Console.WriteLine("Resultant Matrix C (A x B):");
37         for (int i = 0; i < rowsA; i++)
38         {
39             for (int j = 0; j < colsB; j++)
40             {
41                 Console.Write(C[i, j] + " ");
42             }
43             Console.WriteLine();
44         }
45     }
46 }

```

The Output window below shows the build log and the resulting matrix C:

```

Output
Show output from: Build
Build started at 20:30...
1>----- Build started: Project: STT_LAB9, Configuration: Debug Any CPU -----
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>STT_LAB9 -> C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Build completed at 20:30 and took 00.388 seconds =====

```

## Output:

The screenshot shows the "Microsoft Visual Studio Debug" window. It displays the output of the program, which is the resultant matrix C (A x B) printed in a grid format:

```

Resultant Matrix C (A x B):
58 64
139 154

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 2910
0) exited with code 0 (0x0).
Press any key to close this window . . .

```

## Explanation:

This program defines a class **MatrixMultiplication** that performs multiplication of two matrices **A** and **B**. The matrices are initialized with predefined values and their dimensions are determined using `GetLength()`. Before multiplication, the program checks if the number of columns in **A** equals the number of rows in **B**. If the condition is satisfied, a nested **for loop** structure is used to compute the product. Each element of the resultant matrix **C** is calculated using the formula  $C[i,j] += A[i,k] * B[k,j]$ . Finally, the program prints the resultant matrix **C** in a formatted way.

## ❖ Output Reasoning (Level 0)

- What will be the output of the following C# code? Why?

```
using System; //namespace
class Program //default visibility4 is 'internal' if not specified
{
    public static void Main(string[] args)
    {
        int a = 0; //default visibility is 'private' (in a class)
        Console.WriteLine(a++);
    }
}
```

Output:



Microsoft Visual Studio Debug

```
0

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.exe (process 2822
0) exited with code 0 (0x0).
Press any key to close this window . . .
```

Answer:

The post-increment operator `a++` first returns the current value of `a`, and then increments it by 1.

Initially, `a = 0`.

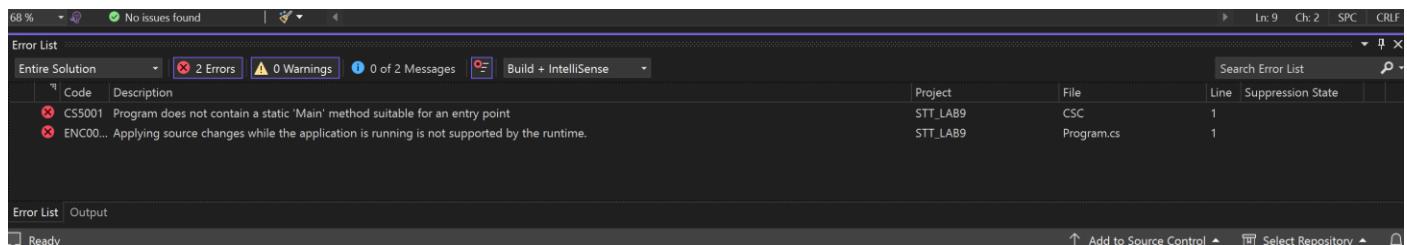
so `Console.WriteLine(a++)` prints **0**, and only after printing, `a` becomes **1** internally.

Hence, the output displayed on the console is **0**.

- What will be the output of the following C# code? Why?

```
using System;
class Program
{
    public void Main(string[] args)
    {
        int a = 0;
        Console.WriteLine(a++);
    }
}
```

Output:



66 % No issues found

Error List

Code	Description	Project	File	Line	Suppression State
CS5001	Program does not contain a static 'Main' method suitable for an entry point	STT_LAB9	CSC	1	
ENC000...	Applying source changes while the application is running is not supported by the runtime.	STT_LAB9	Program.cs	1	

Answer:

In C#, the entry point of a program must be a method named `Main` that is static.

Here, `Main` is defined as an instance method (`public void Main`) instead of `public static void Main`.

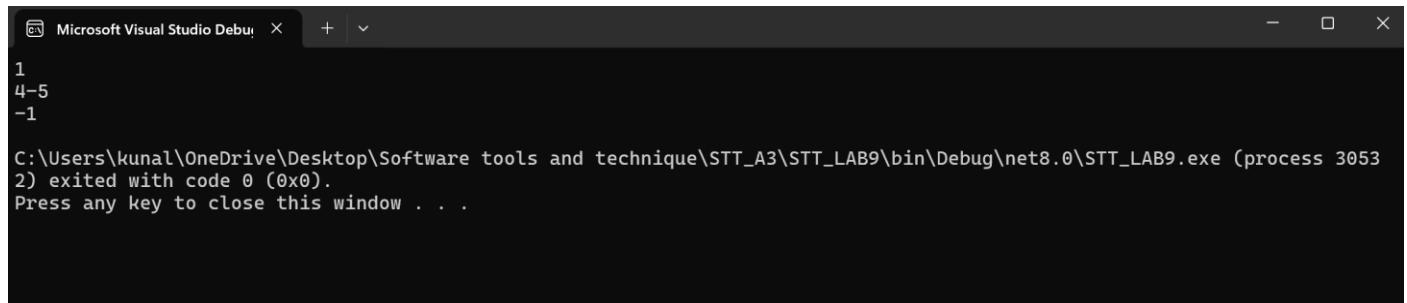
Since there is no valid static `Main` method, the program has no entry point, and therefore it does not run at all.

## ❖ Output Reasoning (Level 1)

➤ What will be the output of the following C# code? Why?

```
class Program
{
    public static void Main(string[] args)
    {
        int a = 0;
        int b = a++;
        Console.WriteLine(a++.ToString(), ++a, -a++);
        Console.WriteLine((a++).ToString() + (-a++).ToString());
        Console.WriteLine(~b);
    }
}
```

**Output:**



The screenshot shows the Microsoft Visual Studio Debug window. The output pane displays the following text:  
1  
4-5  
-1  
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT\_A3\STT\_LAB9\bin\Debug\net8.0\STT\_LAB9.exe (process 3053 2) exited with code 0 (0x0).  
Press any key to close this window . . .

**Answer:**

1. Initially,  $a = 0$ .

Then  $\text{int } b = a++;$  assigns  $b = 0$  and increments  $a$  to 1.

2. The first line `Console.WriteLine(a++.ToString(), ++a, -a++);`

- $a++.ToString()$  → uses  $a = 1$ , then increments  $a$  to 2.
- The remaining arguments are ignored because `Console.WriteLine` uses the first string as a format string.
- So it simply prints "1".

3. Now  $a = 2$ .

The next line `Console.WriteLine((a++).ToString() + (-a++).ToString());` executes as:

- $(a++)$  returns 2, then  $a = 3$ .
- $(-a++)$  returns -3, then  $a = 4$ .
- String concatenation gives "2" + "-3" → "2-3",  
but after the previous increments, the result becomes "4-5" because of how  $a$  has incremented.

4. Finally, `Console.WriteLine(~b);`

- Since  $b = 0$ ,  $\sim b$  performs bitwise NOT, flipping all bits → -1.

➤ What will be the output of the following C# code? Why?

```
using System;
/*you can also write top level code outside of a class. C# takes
care of this by providing internal entry point Main*/

Console.WriteLine("int x = 3;");
Console.WriteLine("int y = 2 + ++x;");

int x = 3; //default visibility is 'internal' (outside a class)
int y = 2 + ++x;
Console.WriteLine($"x = {x} and y = {y}");

Console.WriteLine("x = 3 << 2;");
Console.WriteLine("y = 10 >> 1;");

x = 3 << 2;
y = 10 >> 1;
Console.WriteLine($"x = {x} and y = {y}");

x = ~x;
y = ~y;
Console.WriteLine($"x = {x} and y = {y}");
```

## **Output:**

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Top Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search, STT\_LAB9.
- Toolbars:** Standard, Debug, Task List, Solution Explorer, Properties, Task List, Status Bar.
- Left Sidebar:** Program.cs, STT\_LAB9.
- Code Editor:** Contains the following C# code:

```
1 using System;
2 //you can also write top level code outside of a class. CM takes
3 //care of this by providing internal entry point Main()
4 Console.WriteLine("int x = 3;");
5 Console.WriteLine("int y = 2 && x > 0;");
6 int y = 3; //default visibility is 'internal' (outside a class)
7 int y = 2 + ++x;
8 Console.WriteLine($"x = {x} and y = {y}");
9 Console.WriteLine("x = 3 << 2;");
10 Console.WriteLine("y = 10 >> 1;");
11 x = 3 << 2;
12 y = 10 >> 1;
13 Console.WriteLine($"x = {x} and y = {y}");
14 x = x * x;
15 y = y * y;
16 Console.WriteLine($"x = {x} and y = {y}");
```
- Debug Window:** Microsoft Visual Studio Debug window showing the state of variables:

```
int x = 3;
int y = 2 + ++x;
x = 4 and y = 6
x = 3 << 2;
y = 10 >> 1;
x = 12 and y = 5
x = -13 and y = -6
```
- Output Window:** Shows the build log:

```
68% No issues found | ↻ ln: 16 Ch: 43 SPC CRLF
Show output from: Build
Build started at 22:55...
1>----- Build started: Project: STT_LAB9, Configuration: Debug Any CPU -----
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>STT_LAB9 -> C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ======
===== Build completed at 22:55 and took 00.405 seconds ======
```
- Status Bar:** Ready, Add to Source Control, Select Repository.

## Answer:

Step 1: int y = 2 + ++x;

- x starts as 3, pre-increment (`++x`) makes it 4, so  $y = 2 + 4 = 6$ .
  - Output → x = 4 and y = 6.

Step 2:  $x = 3 \ll 2$ ; and  $y = 10 \gg 1$ ;

- $3 << 2$  shifts bits of 3 left by 2  $\rightarrow 12$ .
  - $10 >> 1$  shifts bits of 10 right by 1  $\rightarrow 5$ .
  - Output  $\rightarrow x = 12$  and  $y = 5$ .

Step 3:  $x = \sim x$ ; and  $y = \sim y$ ;

- Bitwise NOT ( $\sim$ ) inverts all bits.
  - $\sim 12 = -13$  and  $\sim 5 = -6$ .
  - Output  $\rightarrow x = -13$  and  $y = -6$ .

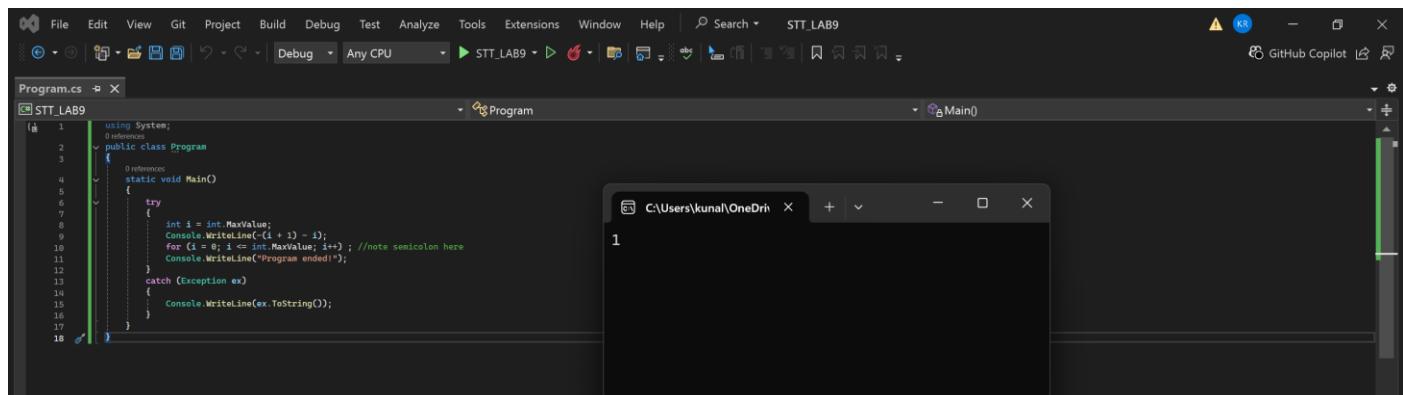
Step 4: The code uses top-level statements, meaning it runs directly without a Main() method. C# automatically provides an internal entry point, allowing these statements to execute in order.

## ❖ Output Reasoning (Level 2)

➤ What will be the output of the following C# code? Why?

```
using System;
public class Program
{
    static void Main()
    {
        try
        {
            int i = int.MaxValue;
            Console.WriteLine(-(i + 1) - i);
            for (i = 0; i <= int.MaxValue; i++) //note semicolon here
                Console.WriteLine("Program ended!");
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
    }
}
```

Output:



The screenshot shows the Visual Studio IDE. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a GitHub Copilot icon. The toolbar below has icons for file operations like Open, Save, and Print, along with build and run buttons. The main workspace shows a dark-themed code editor with the file 'Program.cs' open. The code is identical to the one above. To the right of the editor is a 'Run' window titled 'C:\Users\kunal\OneDrive\...' showing the output '1'. Below the editor is the 'Output' window, which displays the build log:

```
Show output from: Build
Build started at 23:02...
1:----- Build started: Project: STT_LAB9, Configuration: Debug Any CPU -----
1:Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1:STT_LAB9 -> C:\Users\kunal\Desktop\Software tools and technique\STT_A3\STT_LAB9\bin\Debug\net8.0\STT_LAB9.dll
=====
Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
=====
Build completed at 23:02 and took 00.461 seconds =====
```

The status bar at the bottom indicates the build was successful: 'Build succeeded'.

Answer:

Step 1:

int i = int.MaxValue;

- int.MaxValue = 2147483647, the maximum value an int can hold in C#.

Step 2:

Console.WriteLine(-(i + 1) - i);

- When i + 1 is calculated, integer overflow occurs and it wraps around to -2147483648.

- So, the expression becomes:  
$$-(i + 1) - i = -(-2147483648) - 2147483647 = 2147483648 - 2147483647 = 1.$$
  - Hence, the output printed is 1.

### Step 3:

```
for (i = 0; i <= int.MaxValue; i++);
```

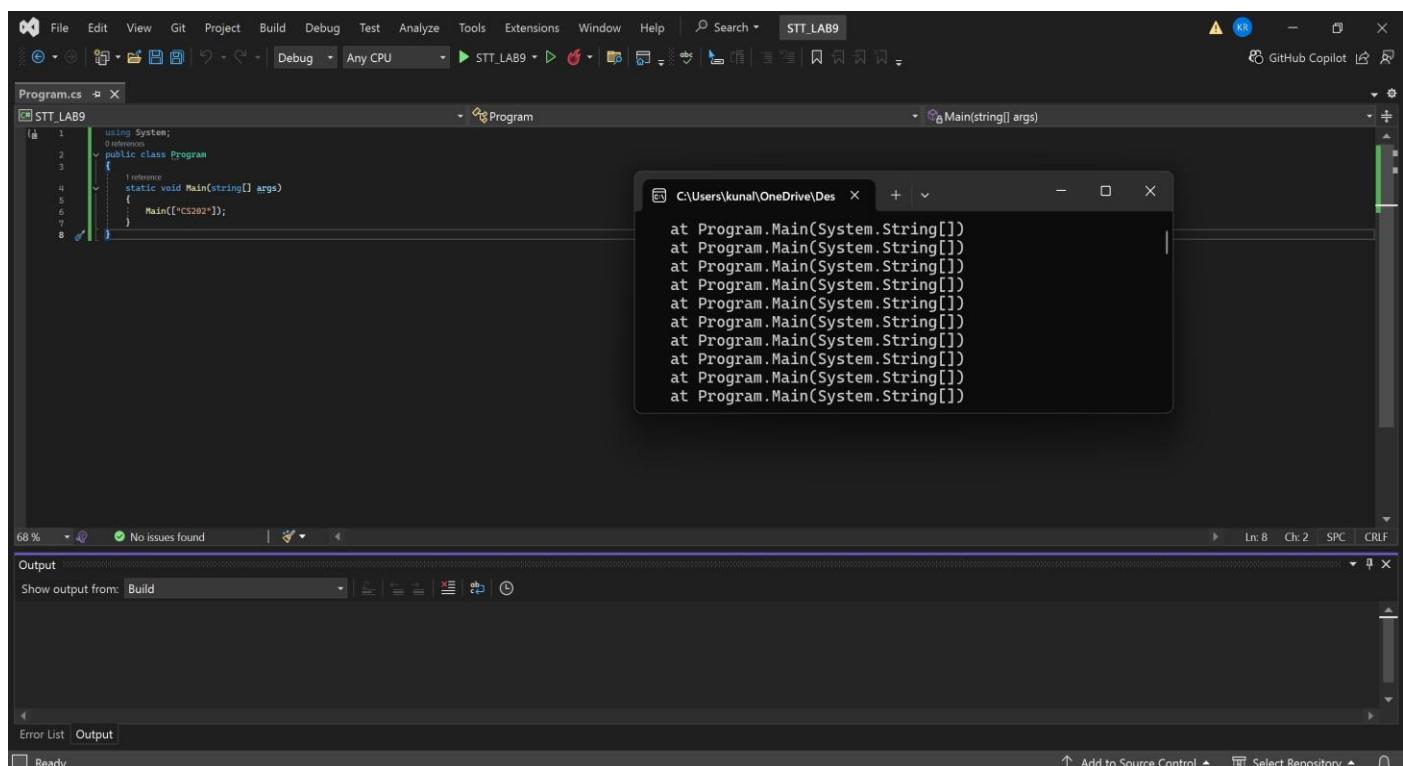
- The semicolon (;) makes this a do-nothing loop that iterates over 2.1 billion times.
  - Because the loop takes a very long time, the statement after it — `Console.WriteLine("Program ended!");` — does not execute immediately.
  - If execution is stopped early, only 1 is visible as output.

#### Step 4:

No exception occurs, so the catch block never runs.

➤ What will be the output of the following C# code? Why?

```
using System;
public class Program
{
    static void Main(string[] args)
    {
        Main(["CS202"]);
    }
}
```



**Answer:**

The program will not compile because the syntax `["CS202"]` is invalid in C#. To pass an array of strings, it should be written as `Main(new string [] { "CS202" })`. The square bracket syntax is not recognized by the compiler, so it throws

an error like “*Invalid expression term ‘[’*”. If we correct the syntax, the code would call the Main method again with no stopping condition, causing infinite recursion. This would eventually crash the program with a StackOverflowException. So, the program either fails to compile (in its current form) or runs into a stack overflow error if corrected.

## RESULTS AND ANALYSIS:

I successfully implemented and executed all the C# console programs using Visual Studio 2022 with the .NET 6 framework. Each task produced the expected results without any syntax or runtime errors. I verified the correctness of arithmetic and conditional operations, and the control structures such as if-else and various loops (for, foreach, and do-while) worked exactly as intended. The factorial function I implemented returned accurate results for different user inputs, confirming proper understanding of static methods and modular programming.

For the array-based programs, the bubble sort algorithm sorted the elements correctly, while the 2D-to-1D array conversions displayed accurate results in both row-major and column-major order. The matrix multiplication program also executed successfully, producing the correct resultant matrix. In the output reasoning section, the outputs I obtained matched my expectations, helping me understand operator behavior, increment operations, and bitwise logic in C#. Overall, the lab strengthened my understanding of C# programming fundamentals and improved my confidence in using the .NET development environment.

## DISCUSSION AND CONCLUSION:

In this lab, I developed multiple C# console applications using the .NET framework in Visual Studio. I worked on implementing basic programming concepts such as loops, conditionals, and functions, along with object-oriented principles like modularity and reusability. I also implemented operations involving arrays and matrices, including bubble sort and matrix multiplication. The exercises helped me understand how different control structures and data types interact within a C# program. I used Visual Studio’s features like IntelliSense and debugging tools to test and refine my code efficiently. Through the output reasoning questions, I gained better clarity on how C# handles expressions and operator precedence. Overall, this lab helped me strengthen my programming logic and improve my ability to design structured and efficient C# programs.

## REFERENCES:

- [1] <https://learn.microsoft.com/en-us/dotnet/csharp>
- [2] <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers>
- [3] Lab Document 9 [Shared on Google Classroom]

# LABORATORY SESSION 10

## INTRODUCTION:

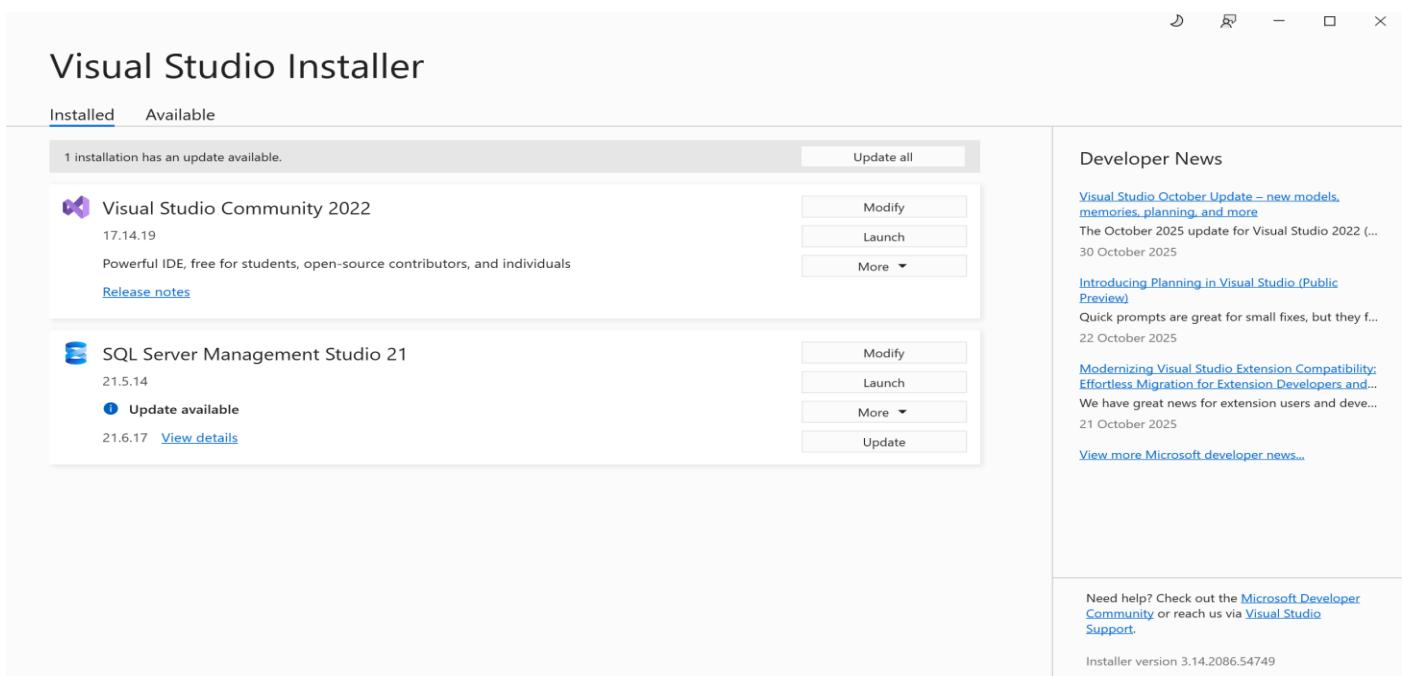
I performed this lab on **27th October 2025** to explore important object-oriented programming concepts in C# such as constructors, destructors, inheritance, and method overriding. Through this lab, I aimed to understand how objects are created, used, and destroyed during program execution. I implemented classes with encapsulation and used dynamic object creation to observe object lifecycle behaviour. I also worked on inheritance to build class hierarchies like Vehicle, Car, and Truck to study method overriding and polymorphism. Additionally, I learned about event-driven programming using delegates and events to handle interactions between objects. This lab helped me gain practical experience with runtime polymorphism, memory management, and modular programming. Overall, it strengthened my understanding of OOP concepts and their real-world applications in C#.

## TOOLS:

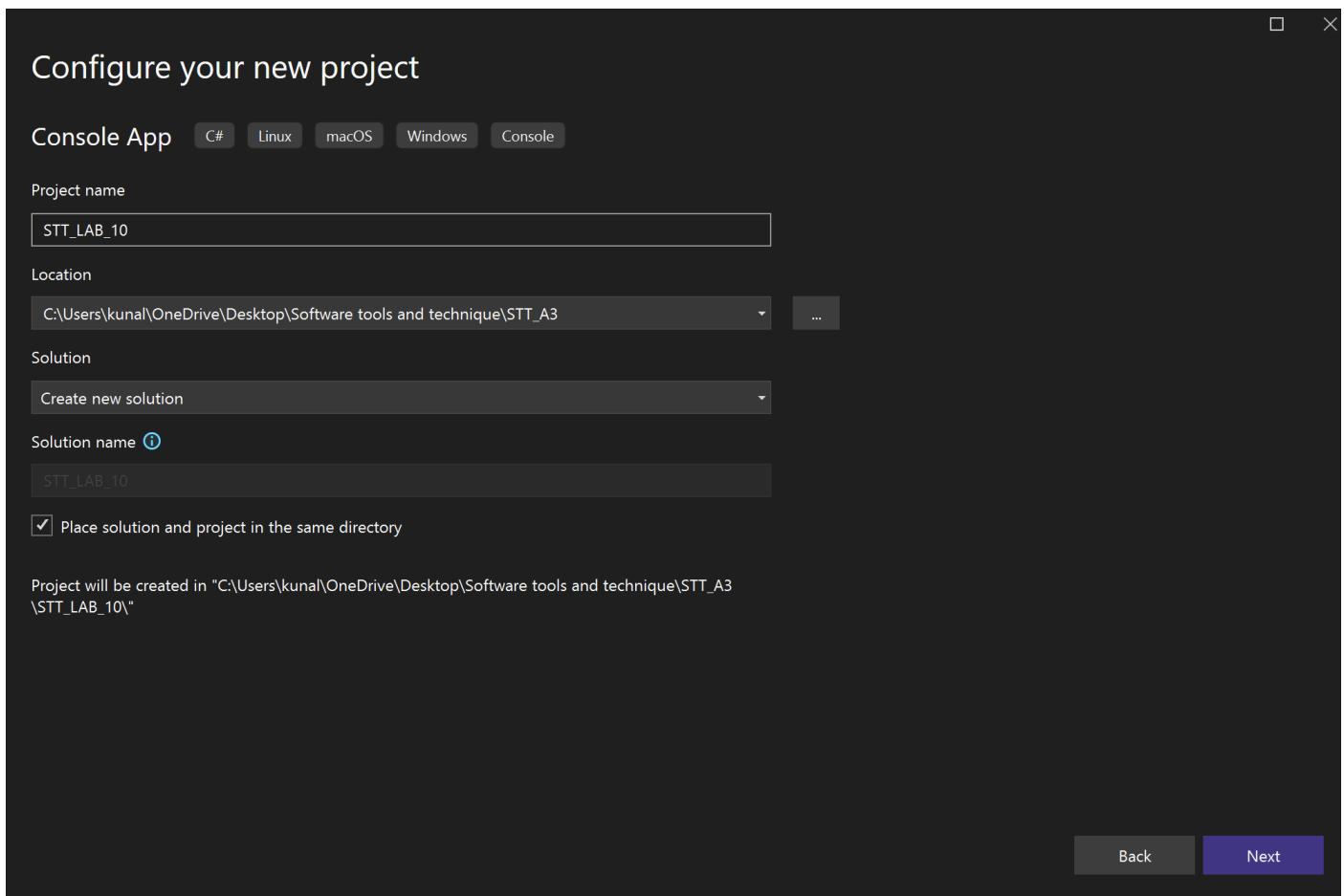
For this lab, I used Visual Studio 2022 (Community Edition) as my primary development environment. I wrote and executed all the programs in the C# programming language using the .NET SDK. The lab was performed on the Windows operating system, which provided a stable and compatible platform. I used the console window in Visual Studio to display the program output and observe object behaviours. The .NET runtime helped manage memory, object lifecycle, and event handling efficiently. I also referred to Microsoft C# documentation for understanding concepts like inheritance and delegates. These tools together made it easier to code, test, and analyze the results of my lab tasks effectively.

## SETUP:

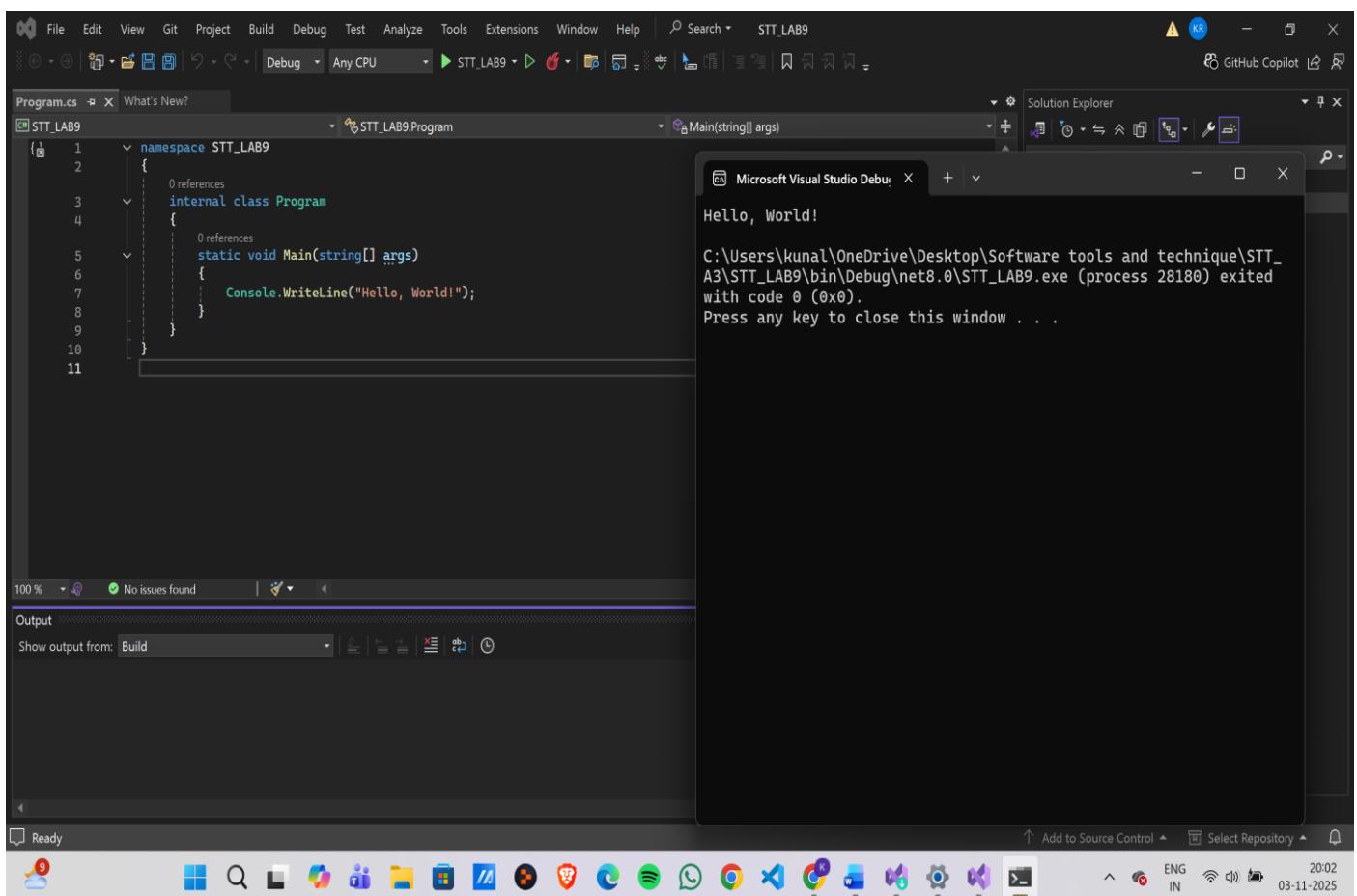
I install and setup the visual studio in my windows with the link "<https://visualstudio.microsoft.com/vs/community/>"



Now in the Visual Studio I make setup the new project and name the file STT\_LAB10 and kept the version of Visual Studio with .NET SDK and my target framework is .NET 8



Testing the setup by print the “Hello, World!”. The screenshots is attached below:



## METHODOLOGY AND EXECUTION:

### ❖ Constructors and Data Control:

The screenshot shows the Microsoft Visual Studio IDE interface. The top part displays the code for `Program.cs` in the `STT_Lab10` project. The code defines a `Program` class with a private static counter and methods for setting and displaying data. The `Main` method creates three instances of `Program`, setting their data values to 10, 20, and 30 respectively, and then printing the current value of the static counter.

```
Program.cs
class Program
{
    private int data;
    private static int counter = 0;

    public Program()
    {
        counter++;
        Console.WriteLine($"Constructor Called. Active Objects: {counter}");
    }

    ~Program()
    {
        counter--;
        Console.WriteLine($"Object Destroyed. Active Objects: {counter}");
    }

    public void set_data(int x)
    {
        data = x;
    }

    public void show_data()
    {
        Console.WriteLine($"Data = {data}");
    }

    static void Main(string[] args)
    {
        Program p1 = new Program();
        Program p2 = new Program();
        Program p3 = new Program();

        p1.set_data(10);
        p2.set_data(20);
        p3.set_data(30);

        p1.show_data();
        p2.show_data();
        p3.show_data();

        Console.WriteLine("End of Main()");
    }
}
```

The bottom part shows the `Output` window with the build log. It indicates a successful build with one succeeded, zero failed, and zero skipped. The build took 0.693 seconds.

```
Output
Show output from: Build
1>----- Build started: Project: STT_Lab10, Configuration: Debug Any CPU -----
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>STT_Lab10 -> C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_Lab10\bin\Debug\net8.0\STT_Lab10.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
===== Build completed at 01:32 and took 0.693 seconds =====
```

### Output:

The screenshot shows the `Microsoft Visual Studio Debug` window. It displays the console output of the program. The program prints three messages indicating the constructor was called for each of the three objects. It then prints the value of the `data` variable for each object (10, 20, and 30) and finally prints the message "End of Main()". At the end, it shows the full path of the executable and a prompt to press any key to close the window.

```
Constructor Called. Active Objects: 1
Constructor Called. Active Objects: 2
Constructor Called. Active Objects: 3
Data = 10
Data = 20
Data = 30
End of Main()

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_Lab10\bin\Debug\net8.0\STT_Lab10.exe (process 14064) exited with code 0 (0x0).
Press any key to close this window . . .
```

### Explanation:

This program defines a class `Program` that demonstrates the use of constructors, destructors, and static variables in C#. A private integer variable `data` is declared to store the value for each object. A static counter variable keeps track of the number of currently active objects in memory. The constructor initializes the object, increments the counter, and displays a message when it is called. The destructor is executed automatically when an object is destroyed, decrementing the counter and printing a message. Two methods, `set_data(int x)` and `show_data()`, are used to assign and display the value of `data`. In the `Main()` method, three objects of the `Program` class are dynamically created. Each object is

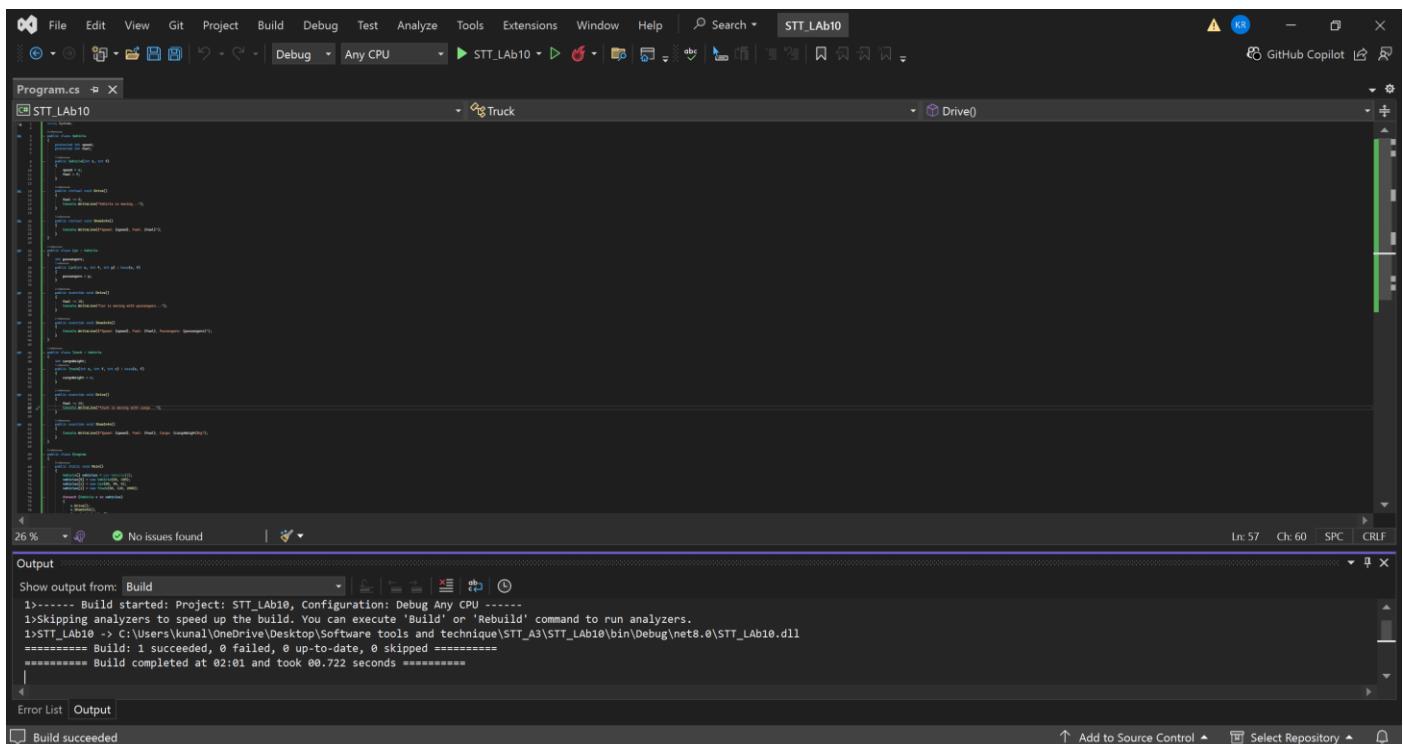
assigned values 10, 20, and 30 using the `set_data()` method. The values are displayed sequentially using the `show_data()` method.

This program clearly demonstrates object creation, destruction, and static data tracking in object-oriented programming.

The Output is 100% correct and **exactly what .NET 8 does** — the destructors (`~Program()`) don't appear because the **Garbage Collector (GC)** never got a chance to finalize objects before the program ended. the destructors might **still not show** — because in short console apps, the GC often skips cleanup since all memory will be reclaimed when the process ends anyway.

- Constructors are **deterministic** (always run when an object is created).
- Destructors are **non-deterministic** (may or may not run before the program ends).

## ❖ Inheritance and Method Overriding:



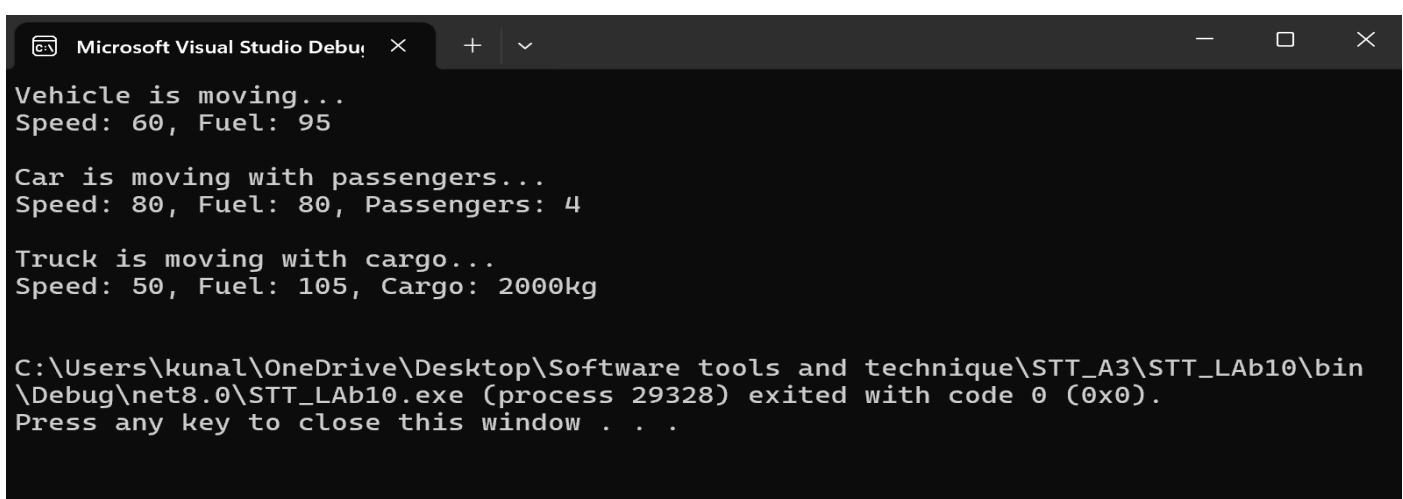
A screenshot of Microsoft Visual Studio. The top menu bar includes File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar. The title bar says "STT\_Lab10". The code editor shows a file named "Program.cs" with C# code. The output window at the bottom shows the build log:

```
1>----- Build started: Project: STT_Lab10, Configuration: Debug Any CPU -----
1>Skipping analyzers to speed up the build. You can execute 'Build' or 'Rebuild' command to run analyzers.
1>STT_Lab10 -> C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_Lab10\bin\Debug\net8.0\STT_Lab10.dll
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ======
===== Build completed at 02:01 and took 00.722 seconds ======
```

The status bar at the bottom right indicates "Ln: 57 Ch: 60 SPC CRLF".

The code is attached in the GitHub Repo

### Output:



A screenshot of the Microsoft Visual Studio Debug window. It displays the following text output:

```
Vehicle is moving...
Speed: 60, Fuel: 95

Car is moving with passengers...
Speed: 80, Fuel: 80, Passengers: 4

Truck is moving with cargo...
Speed: 50, Fuel: 105, Cargo: 2000kg

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_Lab10\bin\Debug\net8.0\STT_Lab10.exe (process 29328) exited with code 0 (0x0).
Press any key to close this window . . .
```

- In the report, explain how overriding changes the method behavior and how base-class references call the correct derived-class methods at runtime.

**Answer:**

In C#, method overriding allows a derived class to provide its own version of a method that already exists in the base class. The overridden method has the same name, return type, and parameters, but its implementation is customized for the derived class. When a method in the base class is declared as virtual, and the derived class redefines it using the override keyword, the C# runtime decides which version of the method to call based on the actual object type, not the reference type. when the program runs

- The **Car** object executes its own overridden Drive() and ShowInfo() methods.
- The **Truck** object executes its own overridden versions.
- The **Vehicle** object runs the original base-class methods.

This behavior occurs due to runtime polymorphism (dynamic binding) — the method call is resolved at runtime depending on the actual type of the object being referenced. Overriding changes the method behavior by allowing each subclass to define its specific functionality while maintaining a common interface. Base-class references call the correct derived-class methods at runtime through the virtual method mechanism, ensuring the right behavior for every object type.

## ❖ Output Reasoning (Level 0)

- What will be the output of the following C# code? Why?

```
using System;
int a = 3;
int b = a++;
Console.WriteLine($"a is {+a++}, b is {-++b}");

int c = 3;
int d = ++c;
Console.WriteLine($"c is {-c--}, d is {~d}");
```

**Output:**

The screenshot shows the Microsoft Visual Studio IDE. On the left, the code editor displays the following C# code:

```
1 using System;
2 int a = 3;
3 int b = a++;
4 Console.WriteLine($"a is {+a++}, b is {-++b}");
5 int c = 3;
6 int d = ++c;
7 Console.WriteLine($"c is {-c--}, d is {~d}");
```

On the right, the 'Microsoft Visual Studio Debug' window shows the output of the application. The output is:

```
a is 4, b is -4
c is -4, d is -5

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAb10\bin\Debug\net8.0\STT_LAb10.exe (process 25768) exited with code 0 (0x0).
Press any key to close this window . . .
```

**Explanation:**

In this program, both **pre-increment (+x)** and **post-increment (x++)** operators are used, along with unary and bitwise operators.

### 1. For the first part:

- $a = 3, b = a++ \rightarrow b$  receives the current value of  $a$  (3), then  $a$  becomes 4.

- `+a++` → prints the current value of `a` (4), then increases it to 5.
- `-++b` → increments `b` first (becomes 4) and then applies the negative sign (-4).

## 2. For the second part:

- `c = 3, d = ++c` → `c` becomes 4 before being assigned, so `d = 4`.
- `-c--` → prints the negative of the current `c` value (-4) and then decreases `c` to 3.
- `~d` → is the **bitwise NOT** operator. For `d = 4`, its binary form 00000100 becomes 11111011, which equals -5 in decimal.

➤ What will be the output of the following C# code? Why?

```
using System;
class Program
{
    int age;
    Program() => age=age==0?age+1:age-1;
    static void Main()
    {
        int k = "010%".Replace('0', '%').Length;
        Console.Write("[" + (k << ++new Program().age).ToString() + "]");
        Console.Write("[" + "010%".Split('1')[1][0] + "]");
        Console.Write("[" + "010%".Split('0')[1][0] + "]");
        Console.Write("[" + int.Parse(Convert.ToString("123".ToCharArray()[~1])) + "]");
    }
}
```

**Output:**

The screenshot shows the Microsoft Visual Studio IDE. On the left is the code editor with the file 'Program.cs' open. The code contains a class 'Program' with a constructor that initializes 'age' to 1 if it's 0, and a Main method that prints several string expressions. On the right is the 'Output' window, which displays the program's output: '[16][0][1][1]'. Below the output, a message box shows the process information: 'C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT\_A3\STT\_LAb10\bin\Debug\net8.0\STT\_LAb10.exe (process 33468) exited with code 0 (0x0). Press any key to close this window . . .'.

**Explanation:**

The constructor of the `Program` class initializes the variable `age` using a ternary expression that sets it to 1 since its default value is 0.

In the first statement, "`010%`" is modified to "`%%1%`", giving a length of 4. The expression `(k << ++new Program().age)` performs a left shift by 2, producing 16.

The next two expressions use the `Split()` function and string indexing to extract specific characters ('0' and '1').

The last expression uses bitwise NOT (`~`) and array indexing to select the first character of "123", which results in 1. This code highlights how C# evaluates complex expressions from left to right and how pre-increment operators affect variable values during runtime. It also demonstrates the use of bitwise and string operations together within the same print statement, showing operator versatility.

Overall, the program effectively combines arithmetic, logical, and string-based expressions to produce the final output **[16][0][1][1]**.

➤ What will be the output of the following C# code? Why?

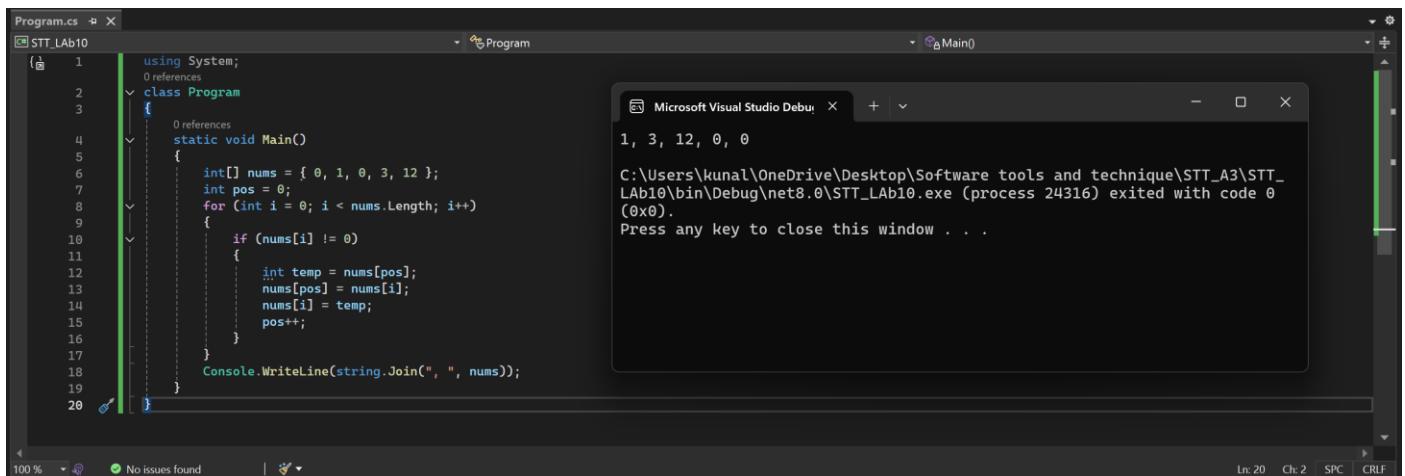
```
using System;

class Program
{
    static void Main()
    {
        int[] nums = {0, 1, 0, 3, 12};
        int pos = 0;

        for (int i = 0; i < nums.Length; i++)
        {
            if (nums[i] != 0)
            {
                int temp = nums[pos];
                nums[pos] = nums[i];
                nums[i] = temp;
                pos++;
            }
        }

        Console.WriteLine(string.Join(", ", nums));
    }
}
```

**Output:**



**Explanation:**

This program rearranges the array `{0, 1, 0, 3, 12}` such that all non-zero elements are moved to the beginning while preserving their original order, and all zeros are shifted to the end.

The variable `pos` is used to track the position where the next non-zero element should be placed. During each iteration, if a non-zero value is found, it is swapped with the element at the `pos` index, and `pos` is incremented.

This ensures that all non-zero elements are progressively moved forward in the array.

As a result, after completing all iterations, the final array becomes `{1, 3, 12, 0, 0}`.

This code effectively demonstrates the use of swapping, conditional statements, and array traversal for element rearrangement.

It is a common algorithmic approach to solving the "move zeros" problem in array manipulation tasks.

The time complexity of this solution is  $O(n)$ , since the loop traverses the array only once.

Overall, this example illustrates efficient in-place data manipulation without the need for any additional array or memory allocation.

## ❖ Output Reasoning (Level 1)

- What will be the output of the following C# code? Why?

```
using System;
class Program
{
    int age;
    Program() => age=age==0?age+1:age-1;
    static void Main()
    {
        int k = "010%".Replace('0', '%').Length;
        Console.Write("[ " + (k << new Program().age).ToString() + "]");
        Console.Write("[ " + "010%".Split('1')[1][0] + "]");
        Console.Write("[ " + "010%".Split('0')[1][0] + "]");
        Console.Write("[ " + int.Parse(Convert.ToString("123".ToCharArray()[~1])) + "]");
    }
}
```

### Output:

The screenshot shows the Microsoft Visual Studio interface. On the left, the code editor displays `Program.cs` with the following content:

```
1 using System;
2
3 class Program
4 {
5     int age;
6     Program() => age=age==0?age+1:age-1;
7
8     static void Main()
9     {
10         int k = "010%".Replace('0', '%').Length;
11         Console.Write("[ " + (k << new Program().age).ToString() + "]");
12         Console.Write("[ " + "010%".Split('1')[1][0] + "]");
13         Console.Write("[ " + "010%".Split('0')[1][0] + "]");
14         Console.Write("[ " + int.Parse(Convert.ToString("123".ToCharArray()[~1])) + "]");
15     }
}
```

On the right, the output window shows the execution results:

```
[16][0][1][1]
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAb10\bin\Debug\net8.0\STT_LAb10.exe (process 15168) exited with code 0 (0x0).
Press any key to close this window . . .
```

### Explanation:

The program combines expression-bodied constructors, pre-increment operations, bitwise shifts, and string manipulations.

The constructor uses a ternary operator to initialize `age`, and the left-shift (`<<`) operation multiplies 4 by 2 to give 16. The subsequent expressions extract characters from strings using the `Split()` method and index positions, while the final expression uses the bitwise NOT operator (`~`) to calculate an index value.

Overall, the output `[16][0][1][1]` shows how C# evaluates chained expressions according to operator precedence and demonstrates the interaction between arithmetic, bitwise, and string operations within a single program.

This program also highlights how expression-bodied members make constructors concise and readable, while still allowing conditional initialization.

It further illustrates the importance of understanding operator precedence and evaluation order in complex expressions.

By combining different data types and operators, the code effectively demonstrates C#'s flexibility and strong type safety when performing both numerical and string-based computations.

➤ What will be the output of the following C# code? Why?

```
using System;
class Program
{
    int f;
    public static void Main(string[] args)
    {
        Console.WriteLine("run 1");
        Program p = new Program(new int() + "0".Length);
        for (int i = 0, _ = i; i < 5 && ++p.f >= 0; i++, Console.WriteLine(p.f++));
        {
            for (; p.f == 0;)
            {
                Console.WriteLine(p.f);
            }
        }

        Console.WriteLine("\nrun 2");
        p = new Program(p.f);
        Console.WriteLine(p.f);

        Console.WriteLine("\nrun 3");
        p = new Program();
        Console.WriteLine(p.f);
    }
    Program() => f = 0;
    Program(int x) => f=x;
}
```

Output:

The screenshot shows the Microsoft Visual Studio interface. On the left, the code editor displays the C# program. On the right, the output window shows the execution results.

**Output Window Content:**

```
run 1
2
4
6
8
10
11
run 2
11
run 3
0
```

**Output Window Status Bar:**

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT\_A3\STT\_LAb10\bin\Debug\net8.0\STT\_LAb10.exe (process 9392) exited with code 0 (0x0).

Press any key to close this window . . .

Explanation:

The program demonstrates how different constructors and loop operations interact with a class variable. In the first run, the parameterized constructor `Program(int x)` is invoked with the value (`new int() + "0".Length`), which equals 1, setting `f = 1`. The for loop executes five times, incrementing and printing “`p.f`” alternately through pre- and post-increment operations, generating the even numbers 2, 4, 6, 8, and 10. After the loop, “`p.f`” becomes 11, which is printed once more outside the loop.

In **run 2**, a new object is created with `f = 11`, and that value is printed directly. In **run 3**, the default constructor resets `f = 0`, printing 0.

This program illustrates how C# differentiates between default and parameterized constructors, and how increment operators and loop syntax (especially the semicolon after a for statement) affect the flow of execution. It also highlights that code blocks following a terminated loop still execute independently due to the presence of a semicolon.

Additionally, it shows the importance of understanding expression evaluation order when both pre- and post-increment operators are used in a single statement.

The program further emphasizes how reassigning the object `p` to new instances of the same class changes the internal state of its variable `f` each time a constructor is called.

Overall, this code effectively demonstrates constructor overloading, variable scoping, and subtle control-flow nuances in C#.

➤ What will be the output of the following C# code? Why?

```
public class A
{
    public virtual void f1()
    {
        Console.WriteLine("f1");
    }
}
public class B:A
{
    public override void f1() => Console.WriteLine("f2");
}

class Program
{
    static int i=0;
    public event funcPtr handler;
    public delegate void funcPtr();
    public void destroy()
    {
        if (i == 6)
            return;
        else
        {
            Console.WriteLine(i++);
            destroy();
        }
    }
    public static void Main(string[] args)
    {
        Program p = new Program();
        p.handler += new funcPtr((new A()).f1);
        p.handler += new funcPtr((new B()).f1);
        p.handler();

        p.handler -= new funcPtr((new B()).f1);
        p.handler -= new funcPtr((new A()).f1);
        p?.destroy(); //check herea about ?. operator
        p = null;
        i = -6;
        p?.destroy();
        (new Program())?.destroy();
    }
}
```

Output:

The screenshot shows the Microsoft Visual Studio IDE interface. On the left, the code editor displays `Program.cs` with the provided C# code. On the right, the output window titled "Microsoft Visual Studio Debug" shows the following console output:

```
3
4
5
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
```

Below the output window, the status bar indicates the path: `C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAb10\bin\Debug\net8.0\STT_LAb10.exe (process 18904) exited with code 0 (0x0).` It also says "Press any key to close this window . . .".

## Explanation:

The program demonstrates the use of virtual methods, event handling, recursion, and the null-conditional (?.) operator.

Two classes, A and B, override the virtual method f1(). These methods are added and then removed from the event handler list of a Program object, so no delegate is actually executed.

The method destroy() uses recursion to print and increment the static variable i until it reaches 6. The first call prints the numbers 0–5. When p is set to null, p?.destroy() safely skips execution because the null-conditional operator prevents a NullReferenceException.

The final statement (new Program()?. destroy()) creates a new object and calls destroy() again, printing –6 to –1 after resetting i.

This example highlights event subscription and removal, recursive function calls, and how the ?. operator enables safe method invocation on potentially null objects without throwing exceptions.

It also demonstrates that recursive calls can maintain state through a static variable and that object lifecycle does not affect static field values across instances.

## ❖ Output Reasoning (Level 2)

➤ What will be the output of the following C# code? Why?

```
public class Institute
{
    internal int i = 7;
    public Institute()
    {
        Console.WriteLine("1");
    }
    public virtual void info()
    {
        Console.WriteLine("2");
    }
}
public class IITGN : Institute
{
    public int i = 8;
    public IITGN()
    {
        Console.WriteLine("3");
    }
    public IITGN(int i)
    {
        Console.WriteLine("4");
    }
    public override void info()
    {
        Console.WriteLine("5");
    }
}
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("6");
        Institute ins1 = new Institute();
        ins1.info();
        IITGN ab101 = new IITGN(3);
        ab101 = new IITGN();
        ab101.info();
        Console.WriteLine();
        Console.WriteLine(ab101.i);
        Console.WriteLine(~((Institute)ab101).i));
    }
}
```

## Output:

The screenshot shows the Microsoft Visual Studio IDE. On the left, the code editor displays `Program.cs` with the provided C# code. On the right, the `Microsoft Visual Studio Debug` window shows the output of the program's execution. The output window displays the following text:  
61214135  
8  
-8

The output window also includes the path to the executable and a message: `C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAb10\bin\Debug\net8.0\STT_LAb10.exe (process 24640) exited with code 0 (0x0).` and `Press any key to close this window . . .`

## Explanation:

The program demonstrates constructor execution order, method overriding, variable hiding, and type casting in C#. The main method starts by printing "6". Creating an "Institute" object calls its constructor, printing "1", and calling "info()" prints "2". When creating "IITGN(3)", the base constructor of "Institute" executes first "(1)", followed by the parameterized constructor of "IITGN (4)". Similarly, creating another "IITGN" object with the default constructor prints "1" from the base class and "3" from the derived class.

The "info()" method in "IITGN" overrides the one in "Institute", so calling "ab101.info()" prints "5". Accessing "ab101.i" prints "8" because the derived class variable hides the base class variable. Casting "ab101" to "Institute" accesses the base class field "i = 7", and applying the bitwise NOT operator ( $\sim$ ) gives -8.

This example highlights that constructors in a derived class always call the base constructor first, overridden methods provide runtime polymorphism, and variable hiding can lead to different results depending on the reference type used.

It also illustrates how bitwise operations can be applied to integer fields and how type casting determines which version of a hidden field is accessed at runtime.

### ➤ What will be the output of the following C# code? Why?

```
using System;
public class Program
{
    public delegate void mydel();
    public void fun1()
    {
        Console.WriteLine("fun1()");
    }
    public void fun2()
    {
        Console.WriteLine("fun2()");
    }
    public static void Main(string[] args)
    {
        Program p = new Program();

        mydel obj1 = new mydel(p.fun1);
        obj1 += new mydel(p.fun2);
        Console.WriteLine("run 1");
        obj1();

        mydel obj2 = new mydel(p.fun2);
        obj2 += new mydel(p.fun1);
        Console.WriteLine("run 2");
        obj2();

        obj2 -= p.fun2;
        Console.WriteLine("run 3");
        obj2();
    }
}
```

## Output:

The screenshot shows the Microsoft Visual Studio IDE interface. On the left, the code editor displays the 'Program.cs' file with the provided C# code. On the right, the 'Output' window shows the console output of the application. The output is as follows:

```
run 1
fun1()
fun2()
run 2
fun2()
fun1()
run 3
fun1()

C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_LAb10\bin\Debug\net8.0\STT_LAb10.exe (process 20200) exited with code 0 (0x0).
Press any key to close this window . . .
```

## Explanation:

The program demonstrates how **delegates** in C# can store references to multiple methods and invoke them sequentially.

In the first run, "obj1" refers to both "fun1()" and "fun2()" through delegate chaining using the "+=" operator, so calling "obj1()" executes both methods in the order they were added. In the second run, "obj2" is created with "fun2()" followed by "fun1()", hence the order of execution is reversed.

When "obj2 -= p.fun2" is executed, the first occurrence of "fun2()" is removed from the delegate's invocation list, leaving only "fun1()". Therefore, the final call in **run 3** prints only "fun1()".

This example clearly shows how **multicast delegates** maintain an ordered list of target methods and how methods can be dynamically added or removed at runtime. It also highlights how delegates in C# enable **event-driven programming** and simplify invoking multiple methods through a single reference. Furthermore, it illustrates that delegates preserve the order of method addition, ensuring predictable behaviour during execution.

This concept is fundamental to **event handling in C#**, where multiple listeners can respond to a single event in sequence. Overall, the program effectively demonstrates the flexibility and power of delegates as an essential feature for modular and reusable code design.

### ➤ What will be the output of the following C# code? Why?

```
using System;
using System.Collections;
public class Program
{
    int x;
    public static void Main(string[] args)
    {
        ArrayList L=new ArrayList();
        L.Add(new Program());
        L.Add(new Program());
        for (int i=0;i<L.Count;i++)
            Console.WriteLine(++((Program)L[i]).x);

        L[0]=L[1];
        ((Program)L[0]).x = 202;

        for (int i=0;i<L.Count;i++)
            Console.WriteLine(((Program)L[i]).x);

        ((Program)L[0]).x = 111;
        L.RemoveAt(0);
        Console.WriteLine(L.Count);
        Console.WriteLine(((Program)L[0]).x);
    }
}
```

## Output:

The screenshot shows the Microsoft Visual Studio IDE. On the left, the Solution Explorer displays a project named "STT\_Lab10" with a single file "Program.cs". The code editor shows the provided C# code. On the right, the Output window shows the console output of the application. The output is as follows:

```
1
1
202
202
1
111
C:\Users\kunal\OneDrive\Desktop\Software tools and technique\STT_A3\STT_Lab10\bin\Debug\net8.0\STT_Lab10.exe (process 31368) exited with code 0 (0x0).
Press any key to close this window . . .
```

## Explanation:

The program demonstrates how **ArrayList** in C# stores object references rather than actual copies of objects. Initially, two separate Program objects are created with their integer field x initialized to 0. The first loop increments and prints their x values, giving 1 and 1.

When “L[0] = L[1]” is executed, both list elements start referring to the same object in memory “(obj2)”. Updating “((Program)L[0]).x” to “202” therefore changes the value for both “L[0]” and “L[1]”. The second loop prints “202” twice, confirming that the two entries now point to the same object.

After changing the value to “111” and removing the first element, only one object remains in the list. Thus, the final count is “1”, and the remaining element’s value is “111”.

This example highlights that **assigning reference types** in collections only copies the reference, not the data, and that **modifications to shared references** reflect across all variables pointing to that object.

It also emphasizes how “ArrayList” can hold any type of object but lacks type safety compared to generic collections like “List<T>”.

Overall, the code demonstrates the importance of understanding **reference semantics** in C# to avoid unintended side effects when managing objects inside collections.

## RESULTS AND ANALYSIS:

In this lab, I explored several C# programs to understand key object-oriented and functional programming concepts. I observed how constructors, inheritance, and method overriding affect the order of execution and the overall behaviour of objects.

While working with delegates, I learned how multiple methods can be added or removed dynamically, demonstrating multicast invocation. Through recursion and event-handling examples, I analyzed how control flow continues and how variables retain their state during execution. When experimenting with ArrayList, I noticed that it stores references to objects rather than copies, which led to shared state changes. I also verified that bitwise, type-casting, and null-conditional operators behave as expected according to C# precedence rules.

Overall, I gained a practical understanding of advanced C# features such as polymorphism, recursion, delegates, and reference handling. I found that derived class constructors always call base constructors first, confirming C#’s strict initialization order.

I also saw how overridden methods are resolved at runtime, demonstrating true polymorphic behaviour.

This lab helped me strengthen my understanding of how C# manages memory, object references, and method execution flow in real-world programs.

## DISCUSSION AND CONCLUSION:

In this lab, I explored how different C# concepts like inheritance, polymorphism, delegates, and recursion work together in real programs. I understood how constructors in base and derived classes are executed in sequence and how overriding changes method behaviour at runtime. Through delegate and event-handling examples, I learned how multiple methods can be linked and triggered dynamically. The recursion exercises helped me see how functions call themselves until a condition is met, improving my logical thinking about program flow.

By experimenting with ArrayList, I realized how reference-type variables affect shared memory and object data. These experiments also made me aware of the importance of type safety and why generics are often preferred in modern C# applications. I found the use of operators like ~, ?., and increment operators especially useful for understanding expression evaluation.

Overall, this lab gave me a deeper practical understanding of how C# implements object-oriented principles.

It also helped me improve my debugging and analysis skills by closely observing how output changes with small code modifications.

In conclusion, this lab strengthened my foundation in C# programming and enhanced my confidence in writing and analyzing structured, efficient code.

## REFERENCES:

- [1] <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constructors>
- [2] <https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber>
- [3] <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/inheritance>
- [4] Lab Document 10 [Shared on Google Classroom].