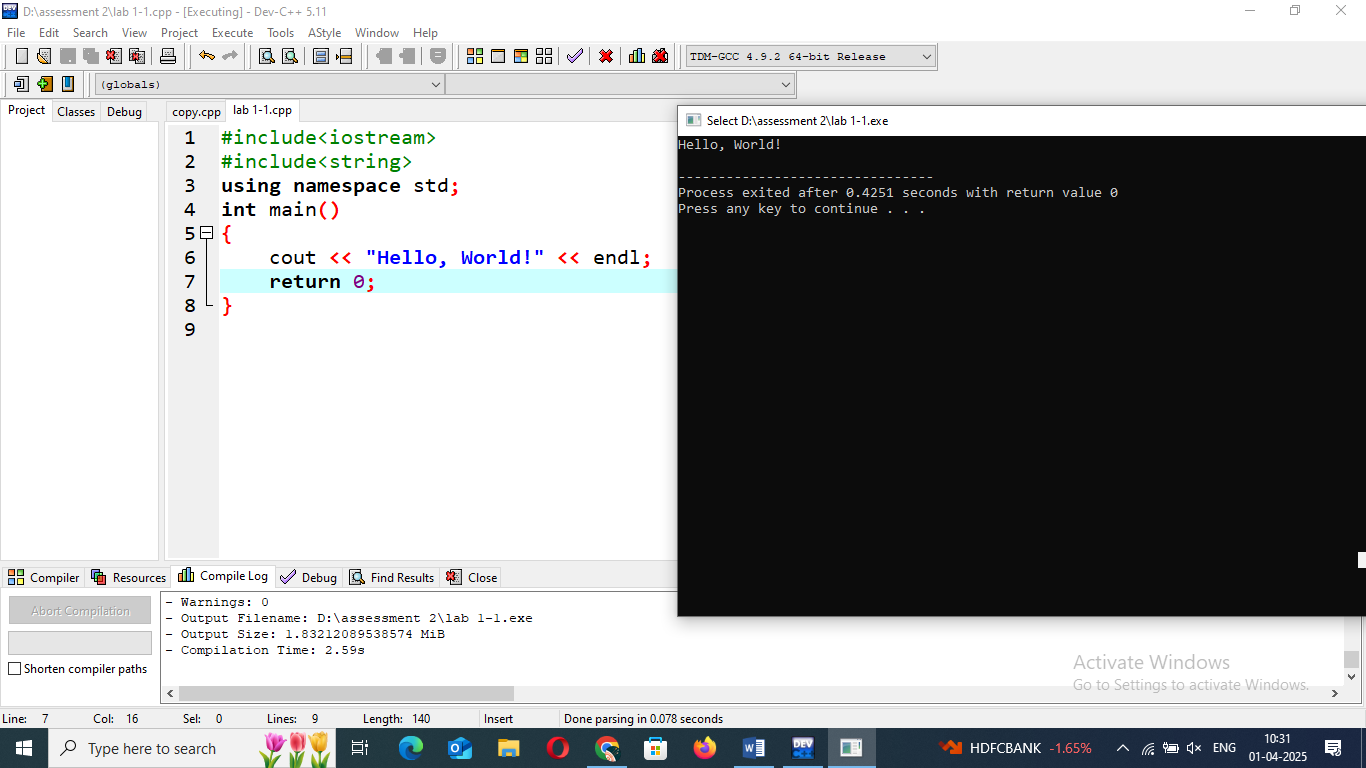
🡪LAB EXERCISES:

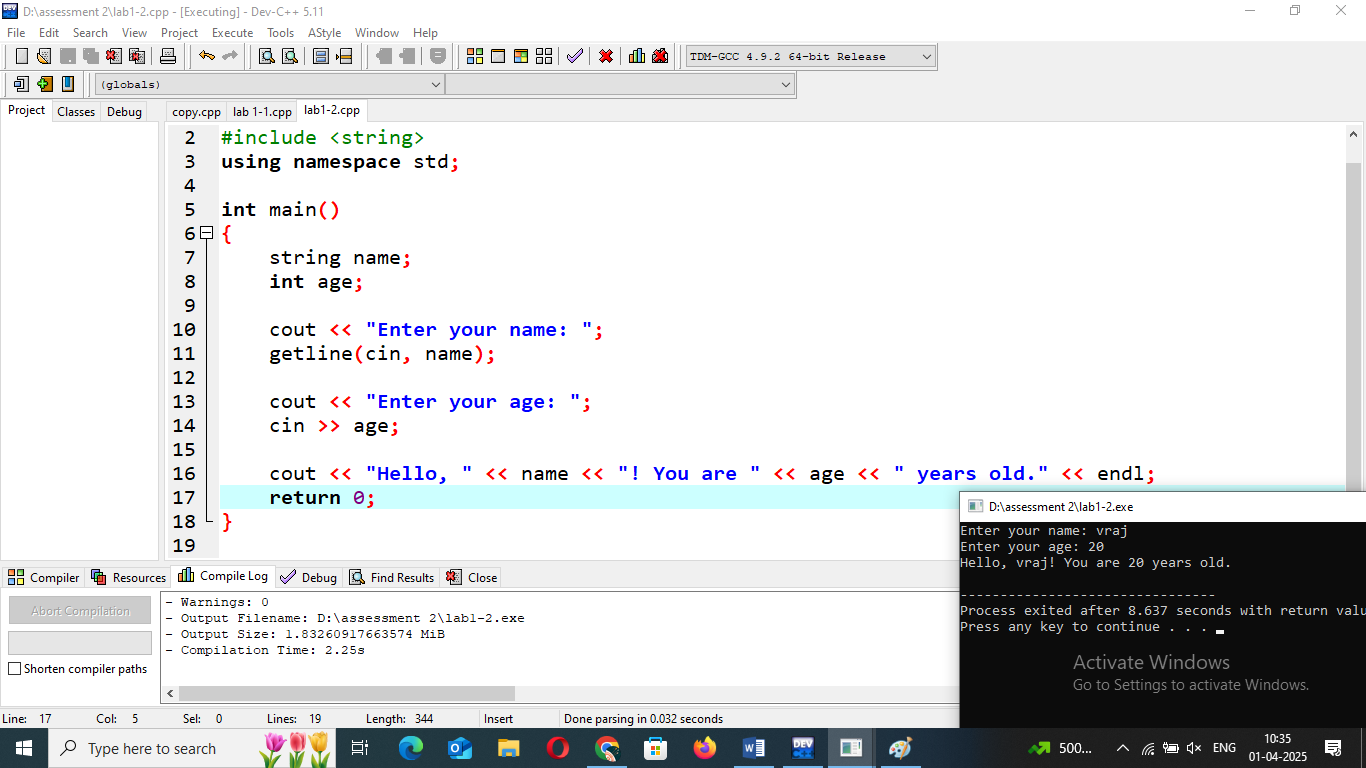
1. Write a simple C++ program to display "Hello, World!".

Objective: Understand the basic structure of a C++ program, including #include, main(), and cout.

=>

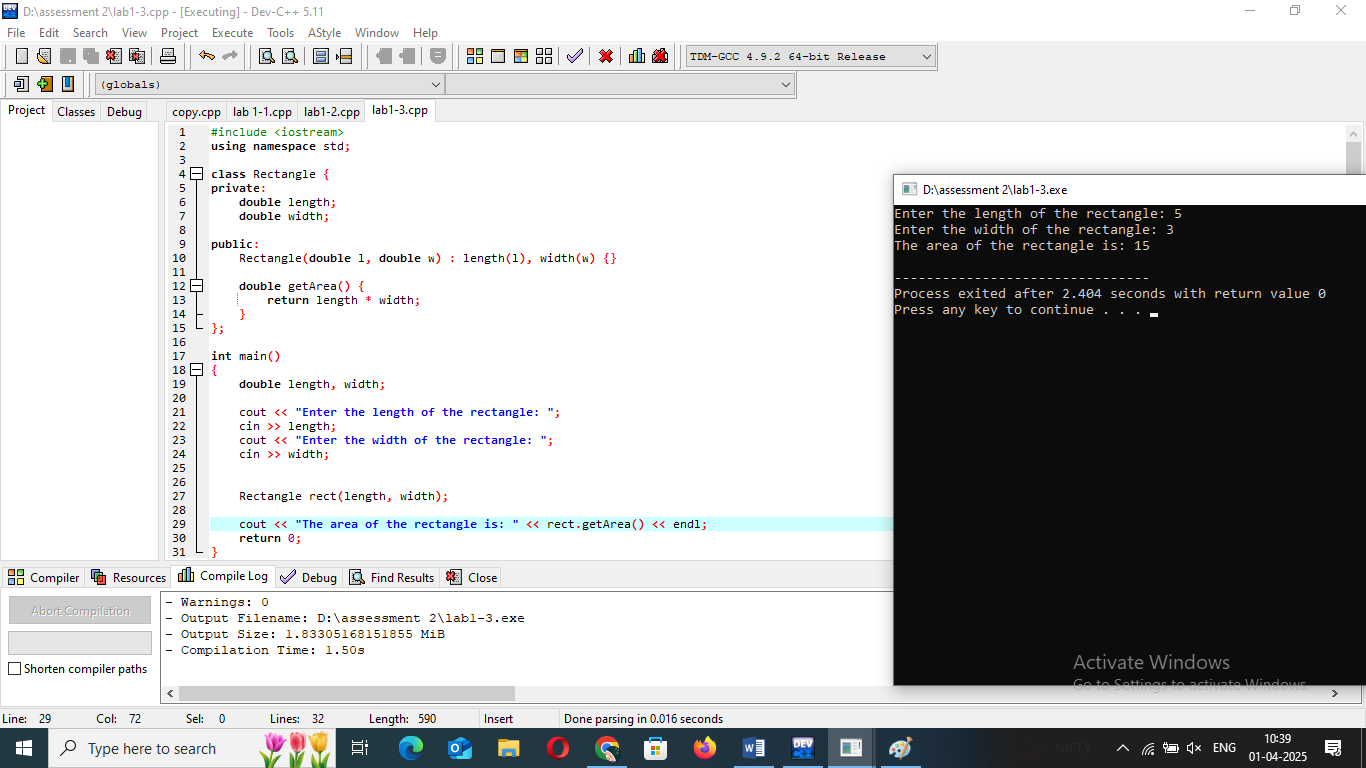
2.Write a C++ program that accepts user input for their name and age and then displays a personalized greeting.

Objective: Practice input/output operations using cin and cout.

=>

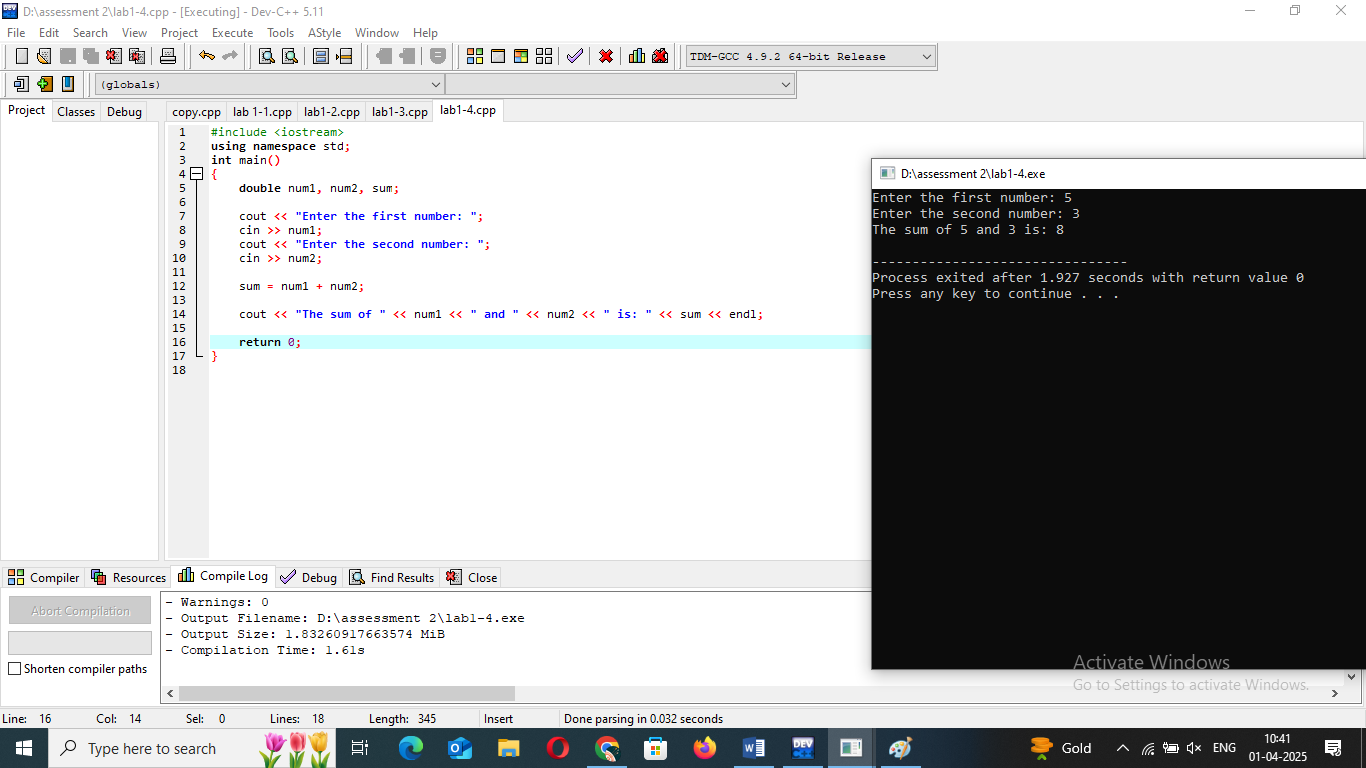
3. Write two small programs: one using Procedural Programming (POP) to calculate the area of a rectangle, and another using Object-Oriented Programming (OOP) with a class and object for the same task.

Objective: Highlight the difference between POP and OOP approaches.

=>

4. Write a program that asks for two numbers and displays their sum. Ensure this is done after setting up the IDE

Objective: Help students understand how to install, configure, and run programs inan IDE.

=>

THEORY EXERCISE:

1. What are the key differences between Procedural Programming and ObjectOrientedProgramming (OOP)?

=> **1. Core Concept**

* **Procedural Programming (PP)**: Focuses on writing a sequence of instructions to perform a task. It revolves around the concept of procedures (or functions), which are blocks of code that carry out specific tasks. The main focus is on how tasks are performed.
* **Object-Oriented Programming (OOP)**: Focuses on creating objects, which are instances of classes. OOP organizes code around real-world entities, bundling data and the methods that operate on that data into a single unit called a class. The main focus is on modeling real-world behavior.

**2. Data and Functions**

* **PP**: Data and functions are separate. Functions operate on data, and the program's state is generally managed through variables.
* **OOP**: Data (attributes) and functions (methods) are bundled together in objects. The state and behavior are tightly coupled within the object, which makes the code more modular and easier to manage.

**4. Approach to Problem Solving**

* **PP**: Focuses on breaking down tasks into a series of actions and solving problems by dividing the problem into smaller sub-procedures.
* **OOP**: Focuses on modeling entities in the problem space as objects, organizing them in classes, and solving problems through interactions between objects.

**5. Data Access and Modification**

* **PP**: Data is generally global or passed around between functions, which can lead to unintended side effects and difficulties in tracking changes to the data.
* **OOP**: Data is encapsulated within objects, and access to it is controlled through methods. This **encapsulation** helps protect the integrity of data and reduces the likelihood of errors.

**8. Examples**

* **PP**: C, Fortran, Pascal
* **OOP**: Java, C++, Python, C#

2. List and explain the main advantages of OOP over POP.

=>This modularity makes it easier to manage, modify, and understand large codebases. You can work on individual objects without affecting the entire program.

This reduces code duplication and increases efficiency. **Polymorphism** also enables the use of the same method name to perform different tasks depending on the context, making code more flexible.

This improves security and prevents accidental interference with the internal state of an object, making the code more reliable and easier to debug. It also allows for easier maintenance since changes to the internal implementation do not affect how the object is used.

When the code needs to be updated or modified, it's simpler to make changes to specific objects or methods without risking changes in the rest of the system.

This promotes **code reuse**, reduces redundancy, and leads to a cleaner, more organized codebase. It also allows for the development of more specialized classes based on a common base.

3. Explain the steps involved in setting up a C++ development environment.

=>**1. Choose an Integrated Development Environment (IDE) or Text Editor**

**2. Install a C++ Compiler**

**3. Install Necessary Libraries and Tools**

**4. Configure the Development Environment**

**5. Set Up Environment Variables**

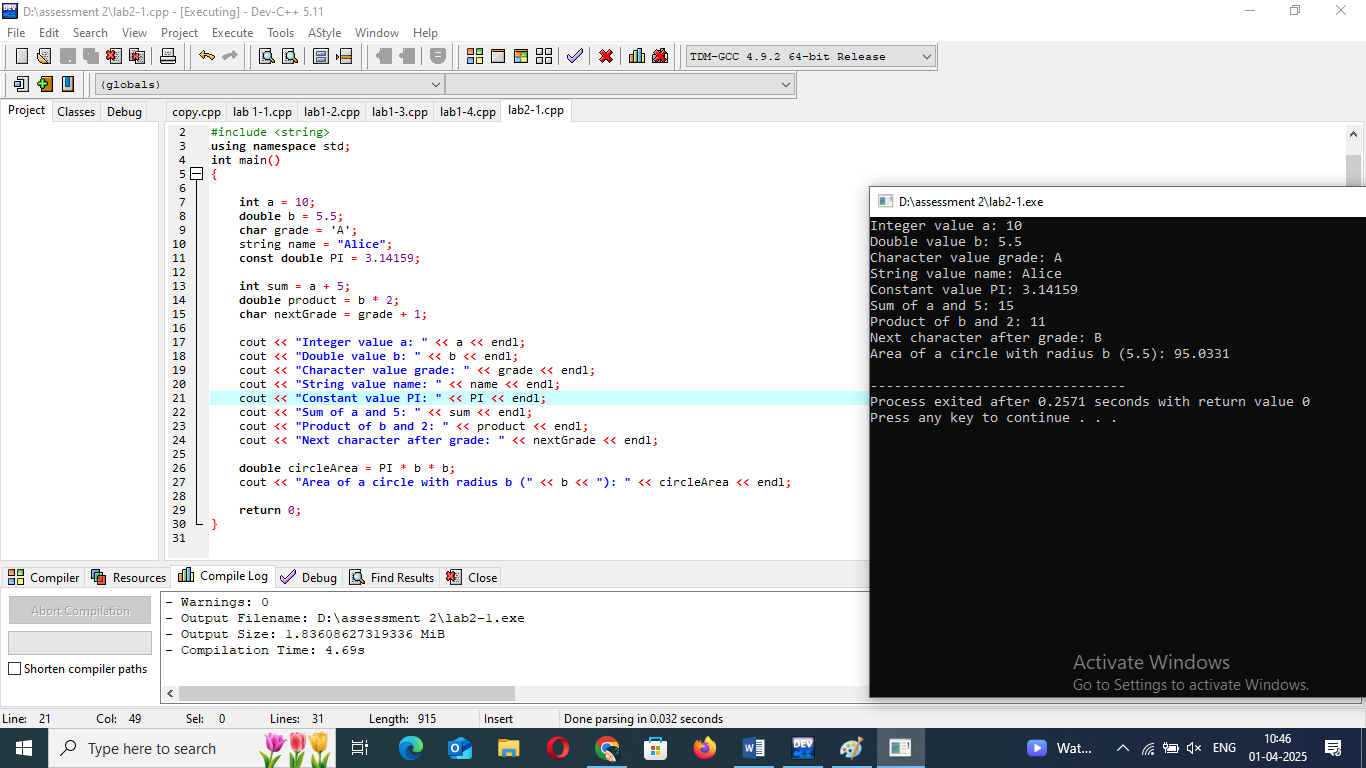
**6. Test Your Development Setup**

2. Variables, Data Types, and Operators

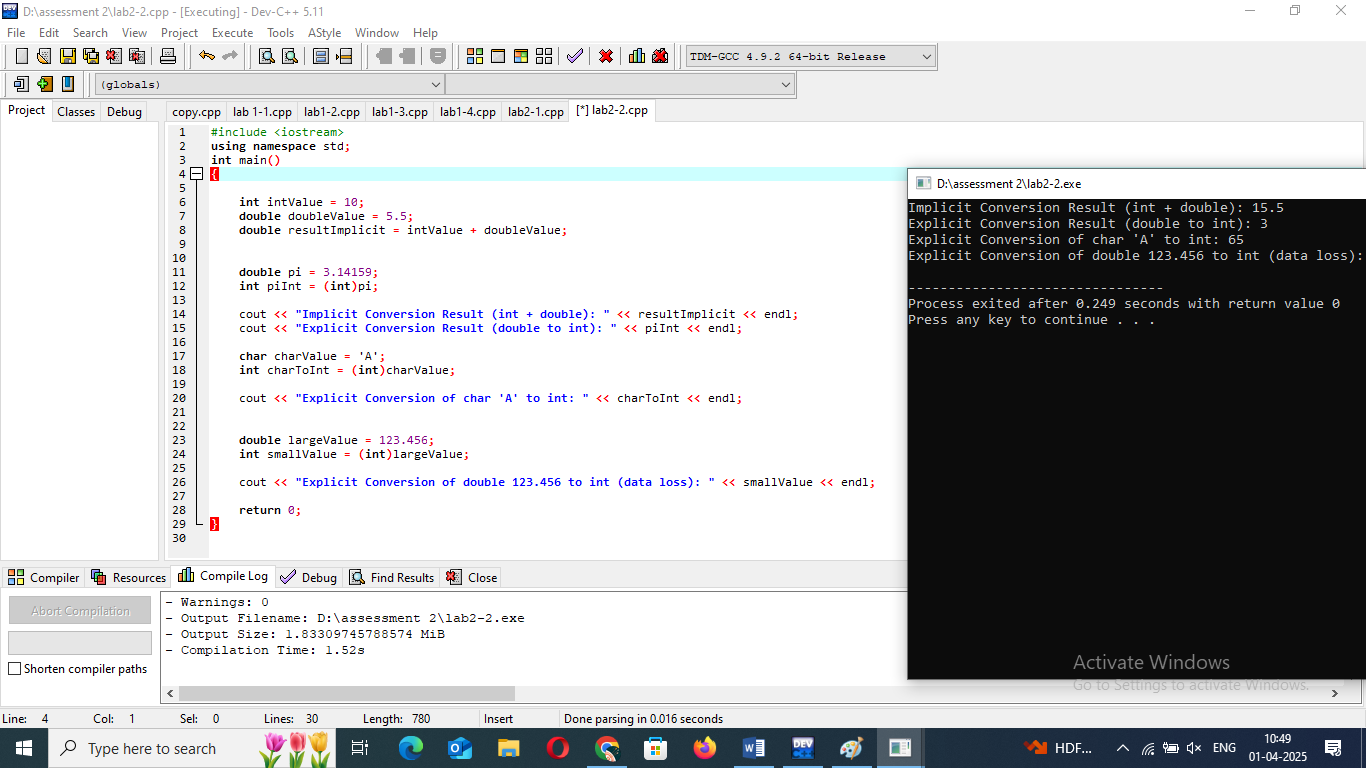
LAB EXERCISES:

1. Write a C++ program that demonstrates the use of variables and constants. Create variables of different data types and perform operations on them.

Objective: Understand the difference between variables and constants.

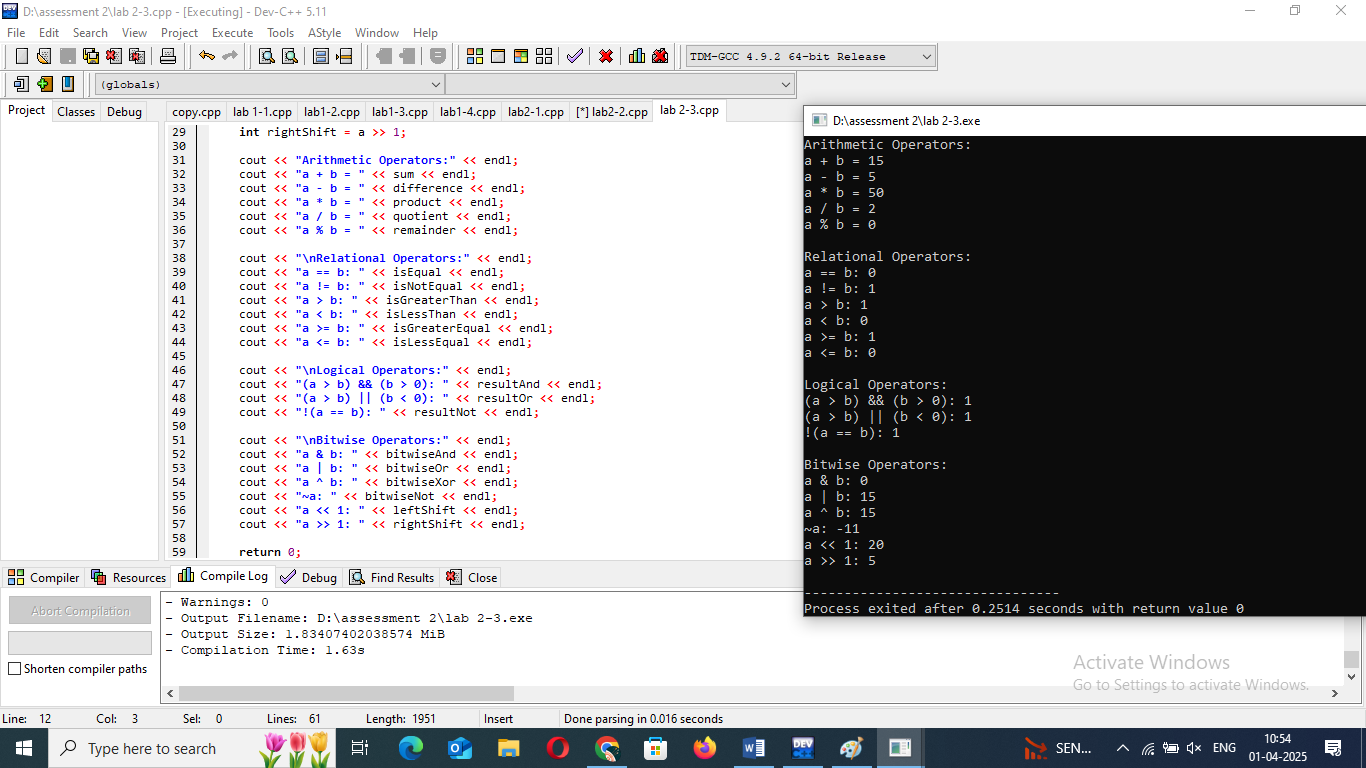
=>

2. Write a C++ program that performs both implicit and explicit type conversions and prints the results.

=>

3. Write a C++ program that demonstrates arithmetic, relational, logical, and bitwise operators. Perform operations using each type of operator and display the results.

Objective: Reinforce understanding of different types of operatorsin C++.

=>

THEORY EXERCISE:

1. What are the different data types available in C++? Explain with examples.

=> a. **Integer Types**

These data types are used to represent whole numbers (both positive and negative).

* **int**: Represents a basic integer.
  + Size: Typically 4 bytes
  + Example:

int age = 25;

b. ***Floating-Point Types***

* **double**: Represents double-precision floating-point numbers
  + Example:

double precisePi= 3.141592653;

c. ***Character Type***

These are used to store individual characters.

* **char**: Represents a single
  + Example:

char grade = 'A';

2. Explain the difference between implicit and explicit type conversion in C++.

### =>1. ****Implicit Type Conversion (Type Casting)****

Implicit type conversion, also known as **automatic type conversion** or **type promotion,** is performed by the **compiler** automatically. It happens when a value of one type is assigned to a variable of a different, usually larger, type. The compiler automatically handles this conversion without requiring any special syntax or explicit direction from the programmer.

### 2. **Explicit Type Conversion (Type Casting)**

Explicit type conversion, also known as **manual type casting** or **type coercion,** is when the programmer explicitly instructs the compiler to convert a variable of one type to another type. In this case, the programmer needs to use a specific syntax to convert the value.

3. What are the different types of operators in C++? Provide examples of each.

### =>1. Arithmetic Operators

These operators are used to perform basic arithmetic operations.

* **Addition (+)**: Adds two operands.

int a = 10, b = 20;

int sum = a + b;

* **Subtraction (-)**: Subtracts the second operand from the first.

int difference = b - a;

* **Multiplication (\*)**: Multiplies two operands.

int product = a \* b;

* **Division (/)**: Divides the first operand by the second.

int quotient = b / a;

* **Modulus (%)**: Returns the remainder of the division of two integers.

int remainder = b % a;

**2. Relational (Comparison) Operators**

These operators are used to compare two values. They return a boolean result

* **Equal to (==)**: Checks if two operands are equal.

bool isEqual = (a == b);

* **Not equal to (!=)**: Checks if two operands are not equal.

bool isNotEqual = (a != b); // true

* **Greater than (>)**: Checks if the left operand is greater than the right operand.

bool isGreater = (a > b); // false

* **Less than (<)**: Checks if the left operand is less than the right operand.

bool isLess = (a < b); // true

* **Greater than or equal to (>=)**: Checks if the left operand is greater than or equal to the right operand.

bool isGreaterEqual = (a >= b); // false

* **Less than or equal to (<=)**: Checks if the left operand is less than or equal to the right operand.

bool isLessEqual = (a <= b); // true

**3. Logical Operators**

These operators are used to perform logical operations, typically with boolean values.

* **Logical AND (&&)**: Returns true if both operands are true.

bool result = (a > 5 && b < 30); // true

* **Logical OR (||)**: Returns true if at least one operand is true.

bool result = (a > 5 || b > 30); // true

* **Logical NOT (!)**: Reverses the logical state of its operand.

bool result = !(a == b); // true (because a != b)

4. Explain the purpose and use of constants and literals in C++.

### =>**Advantages of Using Constants:**

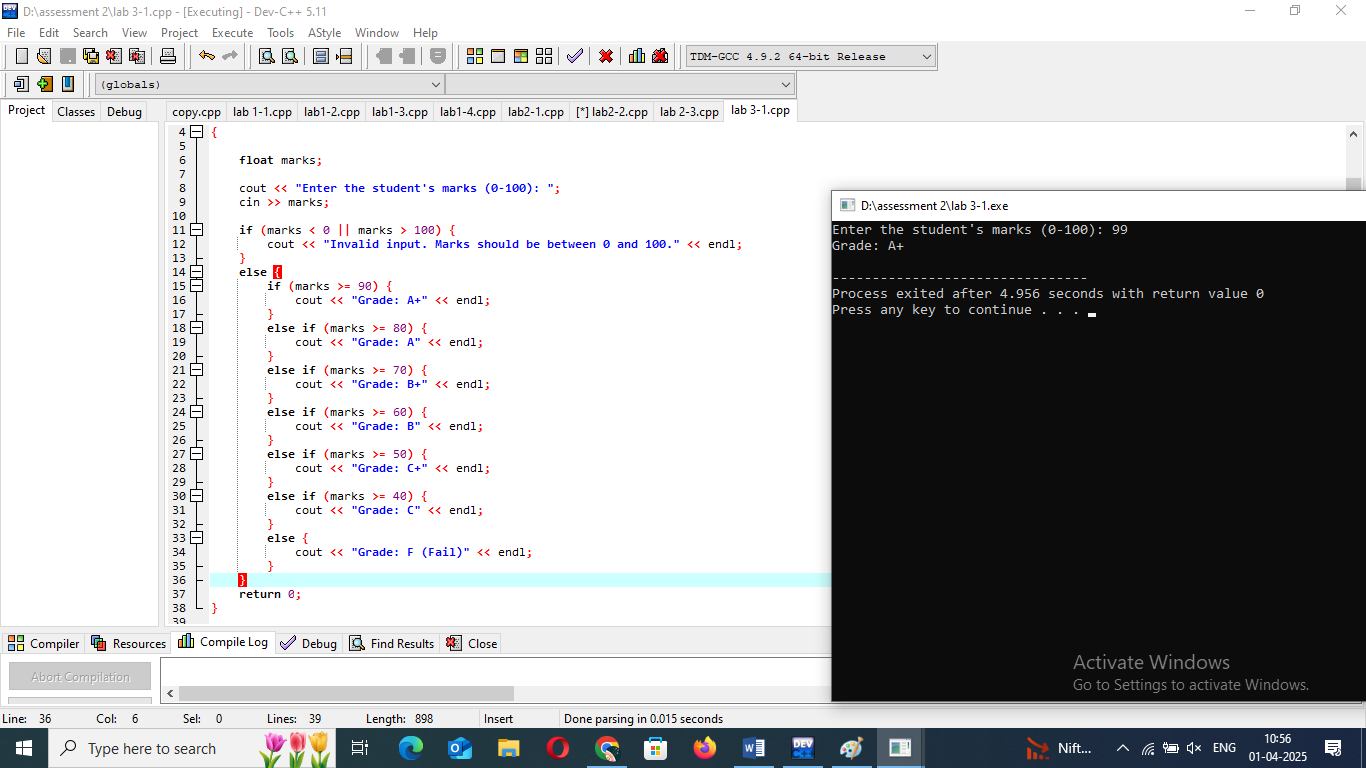
1. **Prevent Accidental Modification:** Constants prevent accidental changes to values, making the program more robust.
2. **Improved Readability:** Using descriptive constant names enhances code readability and makes the purpose of the value clear.
3. **Code Maintenance:** If you need to update a constant's value, you only need to do it in one place, making the code easier to maintain.
4. **Optimization:** Constants are often evaluated at compile time, allowing for optimizations that improve performance.

3. Control Flow Statements

LAB EXERCISES:

1. Write a C++ program that takes a student’s marks as input and calculates the grade based on if-else conditions.

Objective: Practice conditional statements (if-else).

=>

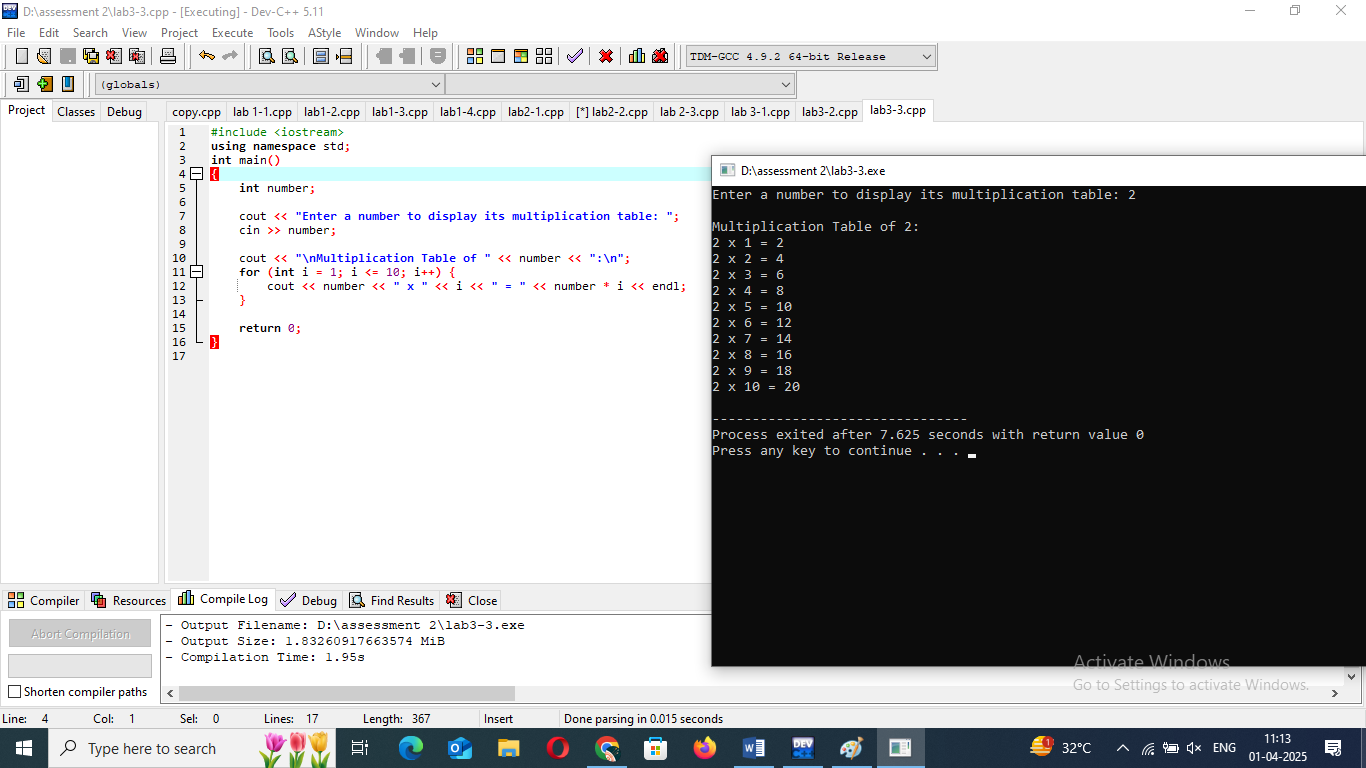
2.Write a C++ program that asks the user to guess a number between 1 and 100. The program should provide hints if the guess is too high or too low. Use loops to allow the user multiple attempts.

Objective: Understand while loops and conditional logic.

---------------------------------------------------------------------

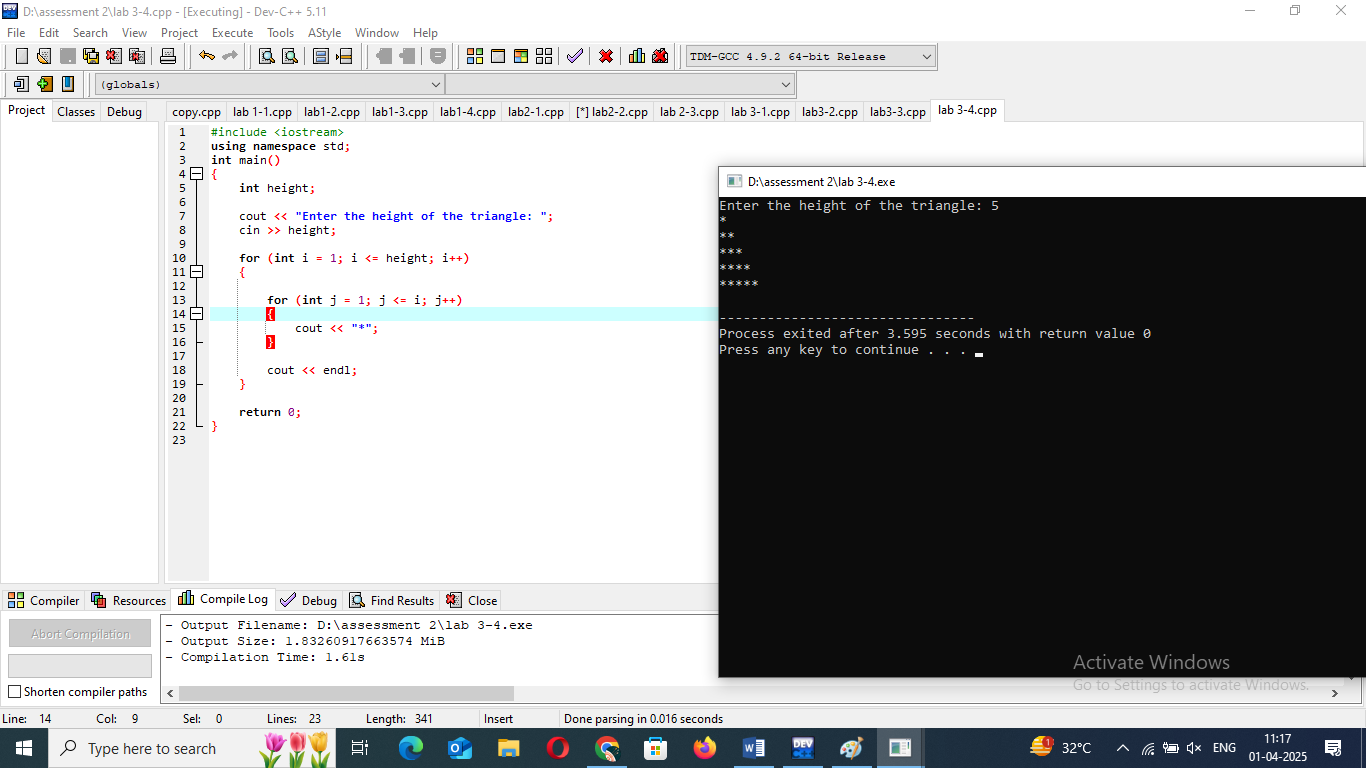
3. Write a C++ program to display the multiplication table of a given number using a for loop.

Objective: Practice using loops.



4. Write a program that prints a right-angled triangle using stars(\*) with a nested loop.

Objective: Learn nested control structures.



THEORY EXERCISE:

1. What are conditional statements in C++? Explain the if-else and switch statements.

### =>****1.**** if-else ****Statement****

The if-else statement is the most basic conditional statement used to execute a block of code based on whether a condition evaluates to true or false. It can have multiple else if conditions to test more than one condition.

***Syntax of*** if-else ***Statement:***

if (condition)

{ // Block of code to be executed if condition is true

}

else

{ // Block of code to be executed if condition is false

}

**2.** switch **Statement**

The switch statement is an alternative to a series of if-else statements when you need to compare a variable to several possible values. It is typically used when there are multiple possible outcomes based on the value of a single variable.

***Syntax of*** *switch* ***Statement:***

cpp

Copy

switch (variable) {

case value1:

// Block of code to be executed if variable == value1

break;

case value2:

// Block of code to be executed if variable == value2

break;

default:

// Block of code to be executed if variable doesn't match any case

}

2. What is the difference between for, while, and do-while loops in C++?

### => 1. ****For Loop****

The for loop is generally used when the number of iterations is known beforehand.

### 2. ****While Loop****

The while loop is used when the number of iterations is not necessarily known in advance. The loop continues to execute as long as the condition is true. The condition is checked **before** each iteration, similar to the for loop.

### 3. ****Do-While Loop****

The do-while loop is similar to the while loop, but the key difference is that the condition is checked **after** the loop body has executed. This guarantees that the loop body will run at least once, even if the condition is false.

3. How are break and continue statements used in loops?

### =>1. ****Break Statement****

The break statement is used to **exit** a loop prematurely, meaning it terminates the loop completely, regardless of the loop's condition. When the break statement is encountered, the program control jumps to the first statement after the loop.

### 2. ****Continue Statement****

The continue statement is used to **skip** the current iteration of the loop and immediately proceed to the next iteration. When the continue statement is encountered, the rest of the loop body is skipped, and the loop condition is re-evaluated for the next iteration.

4. Explain nested control structures with an example.

### => ****Nested**** if ****Statements****

A **nested if statement** occurs when an if statement is inside another if statement. This is useful when you need to check for multiple conditions at different levels.

### ****Nested Loops****

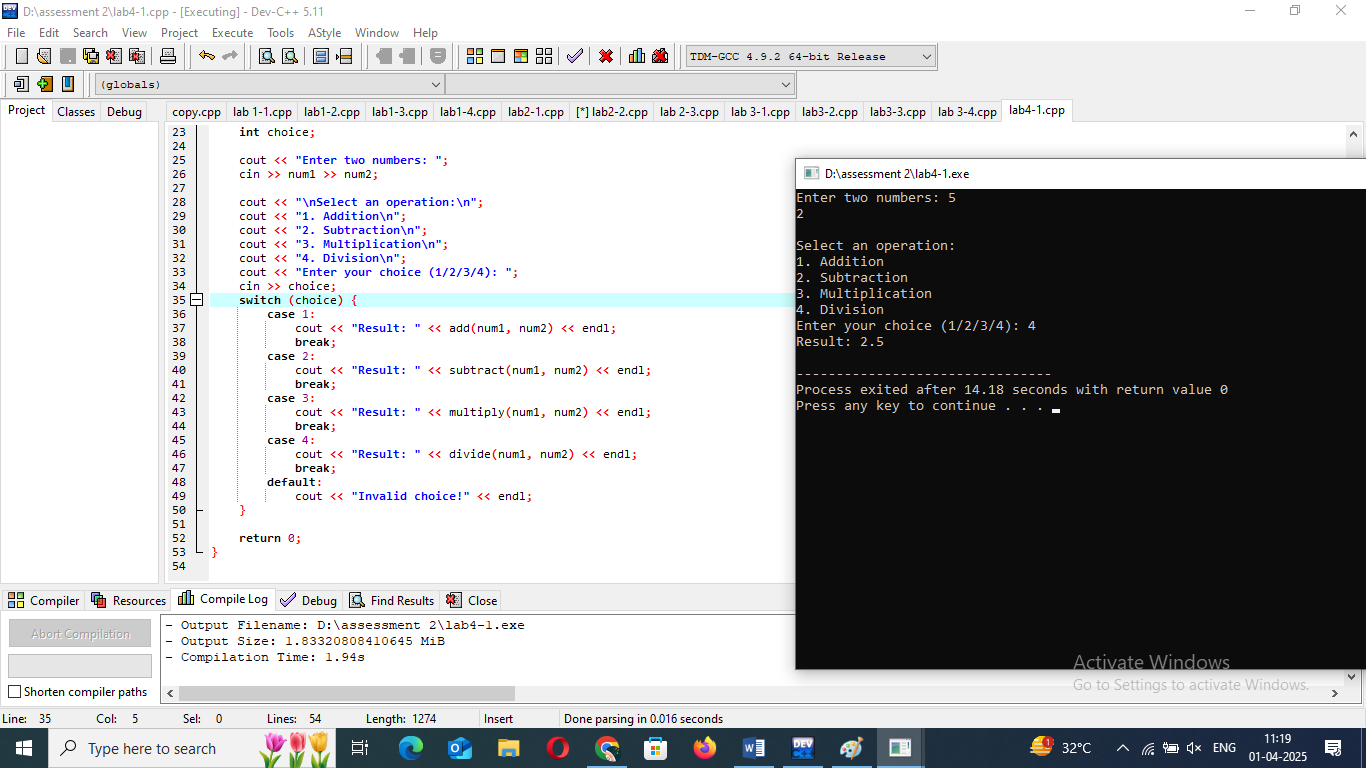
A **nested loop** occurs when one loop is placed inside another loop. This is common when working with multi-dimensional data, such as 2D arrays, or when you need to perform a task repeatedly for each element of a collection.

4. Functions and Scope

LAB EXERCISES:

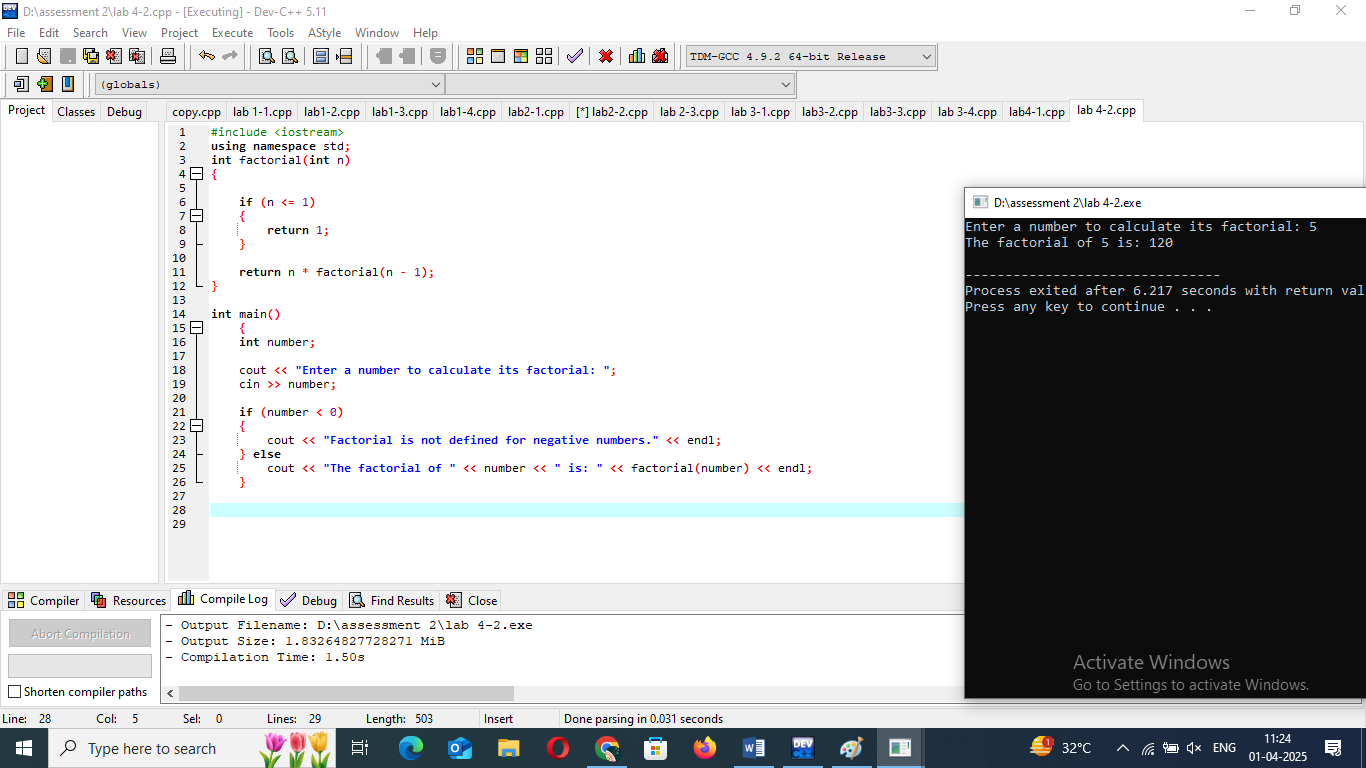
1. Write a C++ program that defines functions for basic arithmetic operations .The main function should call these based on user input.

Objective: Practice defining and using functions in C++.



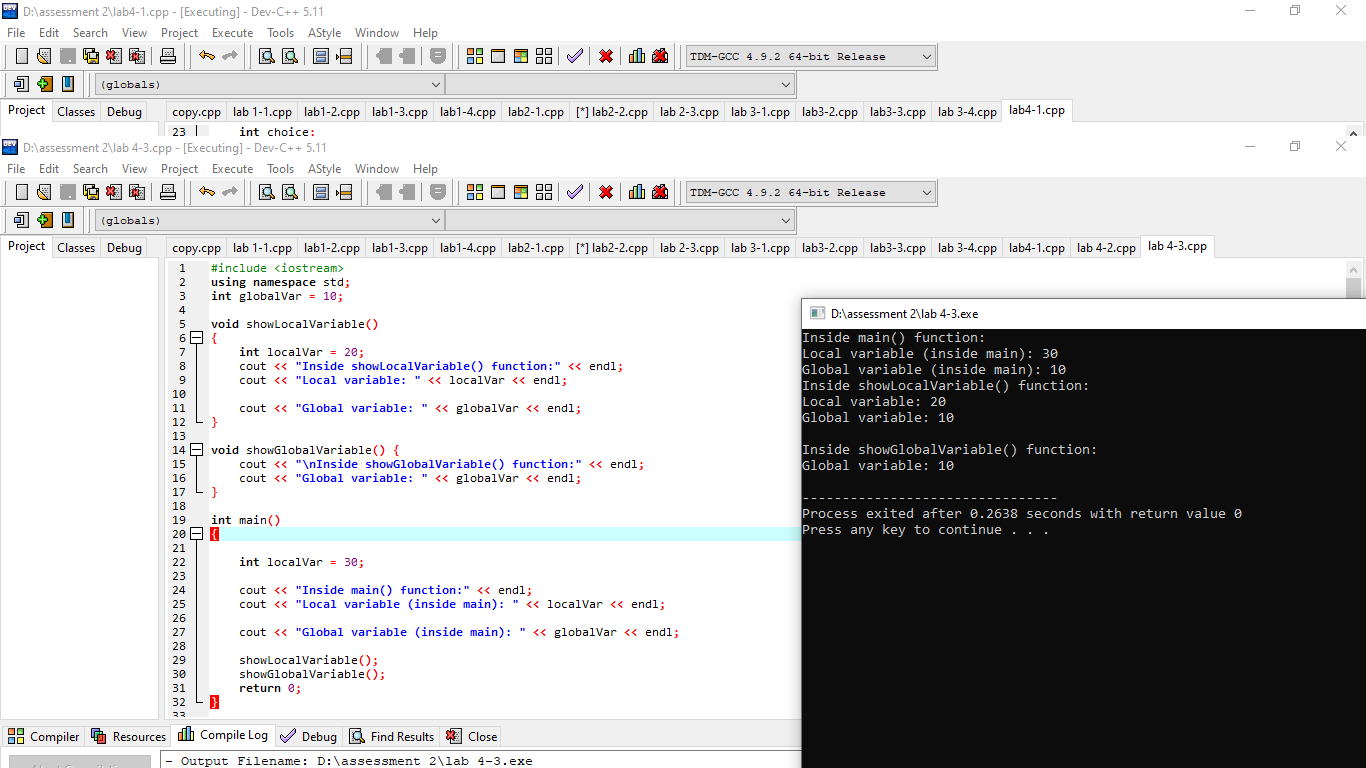
2. Write a C++ program that calculates the factorial of a number using recursion.

Objective: Understand recursion in functions.



3. Write a program that demonstrates the difference between local and global variables in C++. Use functions to show scope.

Objective: Reinforce the concept of variable scope.



THEORY EXERCISE:

1. What is a function in C++? Explain the concept of function declaration, definition, and calling.

=> In C++, a **function** is a block of code that performs a specific task.

### 1. ****Function Declaration****

A **function declaration** is a statement that specifies the function's name, return type, and parameters (if any). It tells the compiler about the function's existence and how to call it. A function declaration doesn't define the actual behavior of the function, just its signature.

### 2. ****Function Definition****

A **function definition** is where the actual code for the function is written. It defines what the function does when it is called.

### 3. ****Function Calling****

A **function call** is how you execute a function that has been declared and defined. When calling a function, you use its name followed by parentheses containing any necessary arguments

2. What is the scope of variables in C++? Differentiate between local and global scope.

=>In C++, **scope** refers to the region of a program where a variable is accessible or visible.

**1. Local Scope**

A variable has local scope if it is declared inside a function, block, or method. It is accessible only within the function or block in which it is declared. Once the function or block finishes execution, the local variable is destroyed and its memory is freed.

* **Access**: Can only be accessed within the function or block where it is declared.
* **Lifetime**: Exists only during the execution of the function or block.
* **Storage**: Local variables are stored on the stack.

**2. Global Scope**

A variable has globalscope if it is declared outside all functions, typically at the top of the program. It is accessible from any function in the program, as long as it's not shadowed by a local variable with the same name. A global variable exists for the entire duration of the program and is available throughout the entire program.

* **Access**: Can be accessed from any function or block in the program.
* **Lifetime**: Exists for the entire duration of the program.
* **Storage**: Global variables are typically stored in a fixed memory location (usually in the data segment).

3. Explain recursion in C++ with an example.

=> **Recursion** in C++ (or any programming language) is a technique where a function calls itself in order to solve a problem. The recursive function typically breaks down the problem into smaller subproblems, and the base case defines when the recursion stops, preventing infinite recursion.

return\_type function\_name(parameters) {

if (base\_condition) {

return base\_value;

} else {

// Recursive case: call the function itself with modified parameters

return function\_name(modified\_parameters);

}

}

4. What are function prototypes in C++? Why are they used?

=>A **function prototype** is a declaration of a function that specifies the function's **name, return type**, and **parameters** (if any) before the function is actually defined.

 **Informing the Compiler About the Function:** A function prototype tells the compiler what the function's signature is — including the return type and the parameters — so the compiler can perform type checking when the function is called in the program. It ensures that functions are called correctly in terms of arguments and return types, even if the actual function definition is located further down in the code.

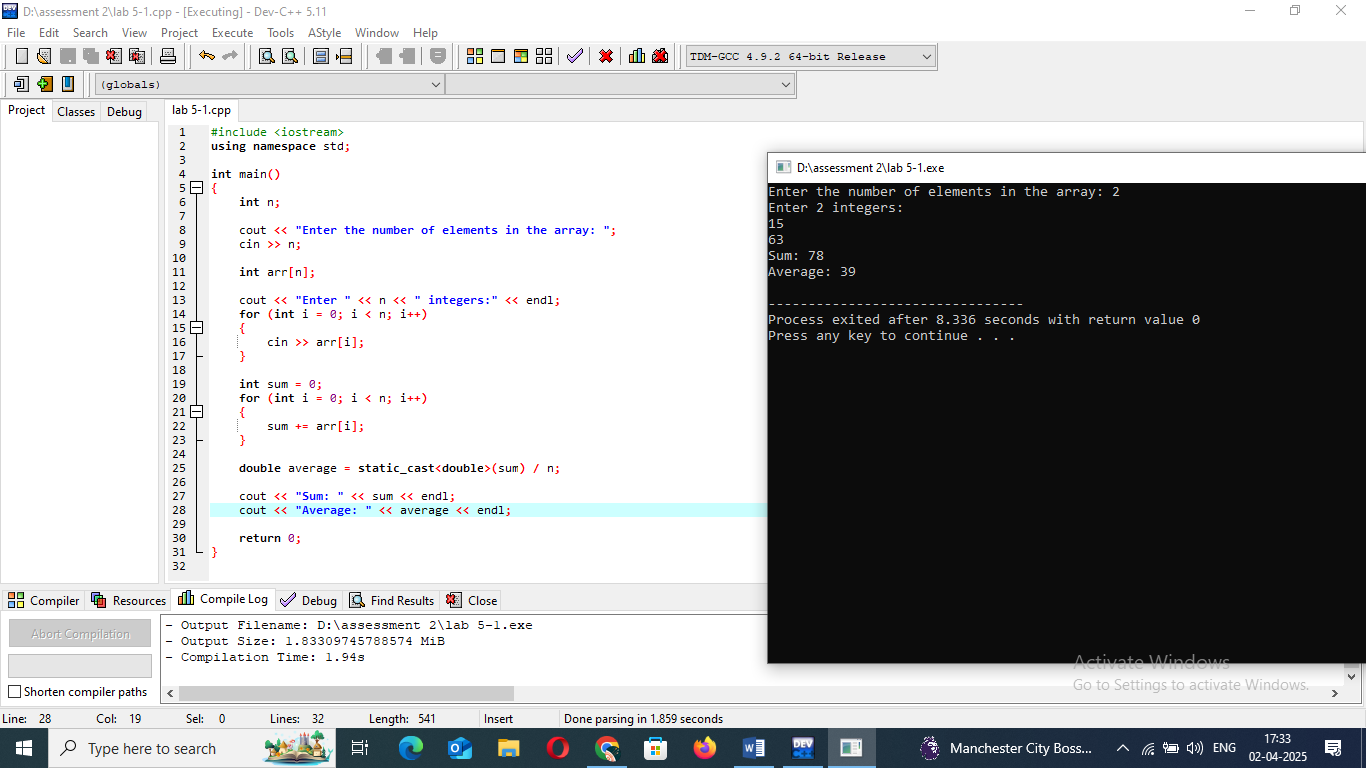
 **Allowing Functions to Be Used Before They Are Defined:** Function prototypes allow you to declare and use functions before they are defined. This is particularly useful when:

* You want to organize code with multiple functions but prefer to place the function implementations after the main() function or in a separate file.
* You need to call a function before its actual definition appears in the code.

5. Arrays and Strings LAB EXERCISES:

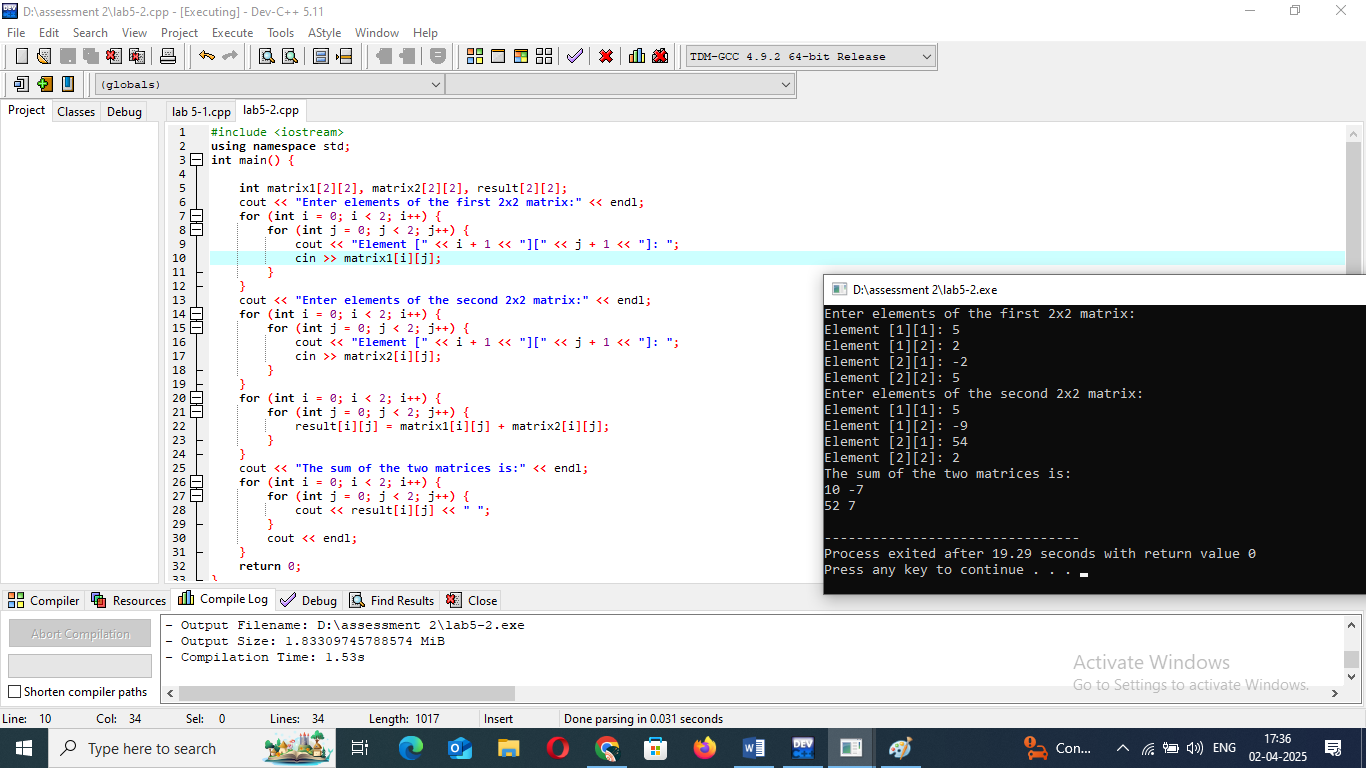
1. write a C++ program that accepts an array of integers, calculates the sum and average, and displays the results.

Objective: Understand basic array manipulation.



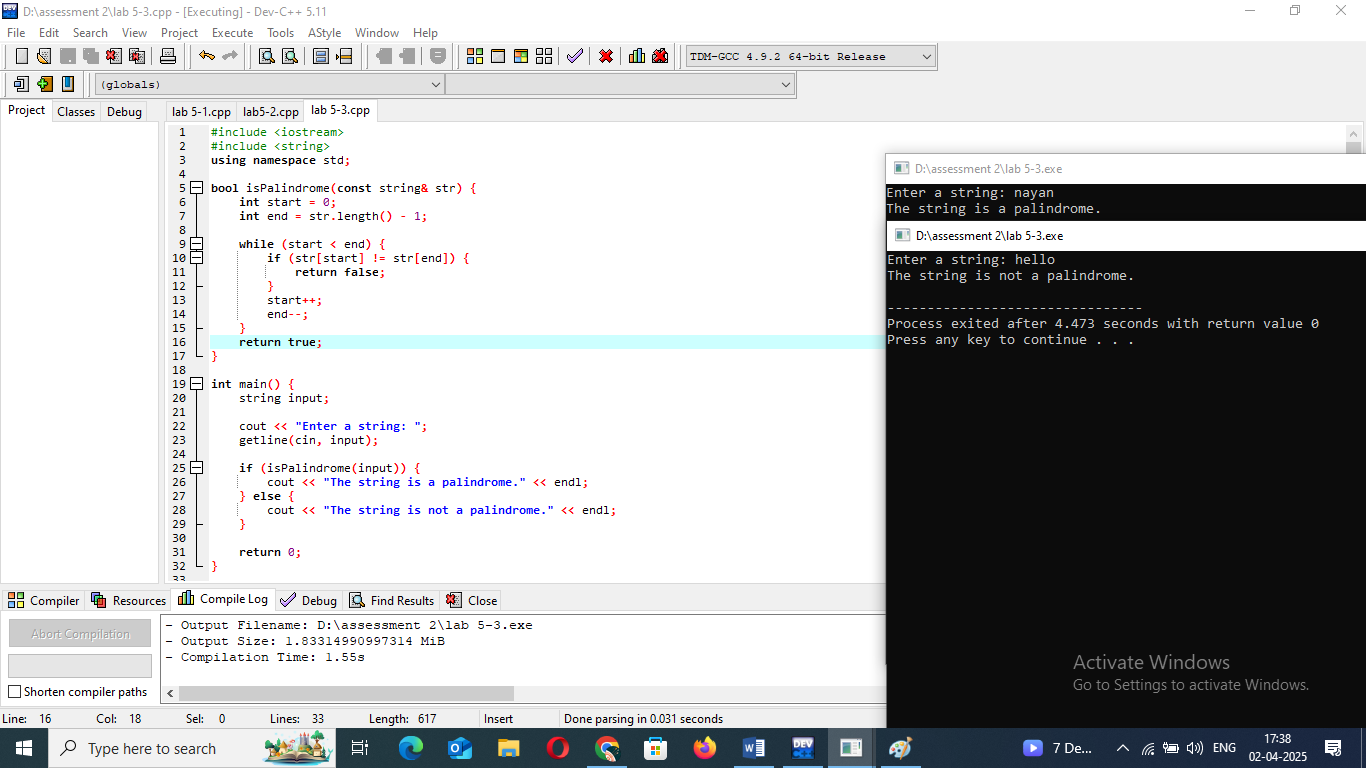
2. Write a C++ program to perform matrix addition on two 2x2 matrices.

Objective: Practice multi-dimensional arrays.



3. Write a C++ program to check if a given string is a palindrome.

Objective: Practice string operations.



THEORY EXERCISE:

1. What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays.

=> In C++, an **array** is a collection of elements of the same data type stored in contiguous memory locations.

**Single-Dimensional Arrays**  
A **single-dimensional array** is a simple list of elements, where each element can be accessed using one index.

**Multi-Dimensional Arrays**  
A **multi-dimensional array** is an array of arrays. The most common type of multi-dimensional array is a **two-dimensional array**, which can be thought of as a matrix (like a table with rows and columns). You can also have arrays with more than two dimensions

2. Explain string handling in C++ with examples.

=> The std::string class provides a higher-level, more flexible way to handle strings in C++. It is part of the C++ Standard Library and provides a rich set of functions for string manipulation, making string handling much easier.

#include <iostream>

#include <string> // Required for std::string

using namespace std;

int main() {

string str = "Hello";

cout << str << endl; // Output: Hello

return 0;

}

3. How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.

=> 1D Array Initialization

int arr[5] = {1, 2, 3, 4, 5};

2D Array Initialization

int arr[2][3] = {

{1, 2, 3},

{4, 5, 6}

};

4. Explain string operations and functions in C++.

=>You can get the number of characters in a string using the size() or length() function.

You can concatenate strings using the + operator or the append() function.

Strings can be compared using the standard relational operators (==, !=, <, >, etc.).

You can extract a part of a string using the substr() function. The syntax is substr

To find a substring within a string, you can use the find() function. It returns the index of the first occurrence of the substring, or std::string::npos if not found.

You can replace part of a string using the replace() function, which takes the starting index, the number of characters to replace, and the new substring.