**MODULE 2:- Introduction to Programming**

1. **THEORY EXERCISE: - Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.**

=>The history of C dates back to the late 1960s when computer scientists were seeking a more efficient and flexible programming language. C was developed by Dennis Ritchie at Bell Labs in 1972.

The first official documentation of C, known as The C Programming Language by Brian Kernighan and Dennis Ritchie, established key principles and syntax in 1978.

The American National Standards Institute (ANSI) standardized C to ensure consistency across different compilers in 1989.

This revision introduced new features such as variable-length arrays, inline functions, and improved support for floating-point arithmetic in 1999.

As technology advances, C continues to play a crucial role in software development and system programming, proving that its relevance is far from fading.

**LAB EXERCISE: -** **Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.**

=> Embedded Systems – C is widely used in microcontrollers, automotive systems, medical devices, and IoT applications due to its efficiency and direct hardware access.

Operating Systems – Most modern operating systems, including Linux, Windows, and macOS, are either written in C or heavily rely on it.

Game Development – Many game engines, including Unreal Engine, incorporate C or C++ to ensure high performance and low-level memory management.

1. **THEORY EXERCISE: - Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like DevC++, VS Code, or CodeBlocks.**

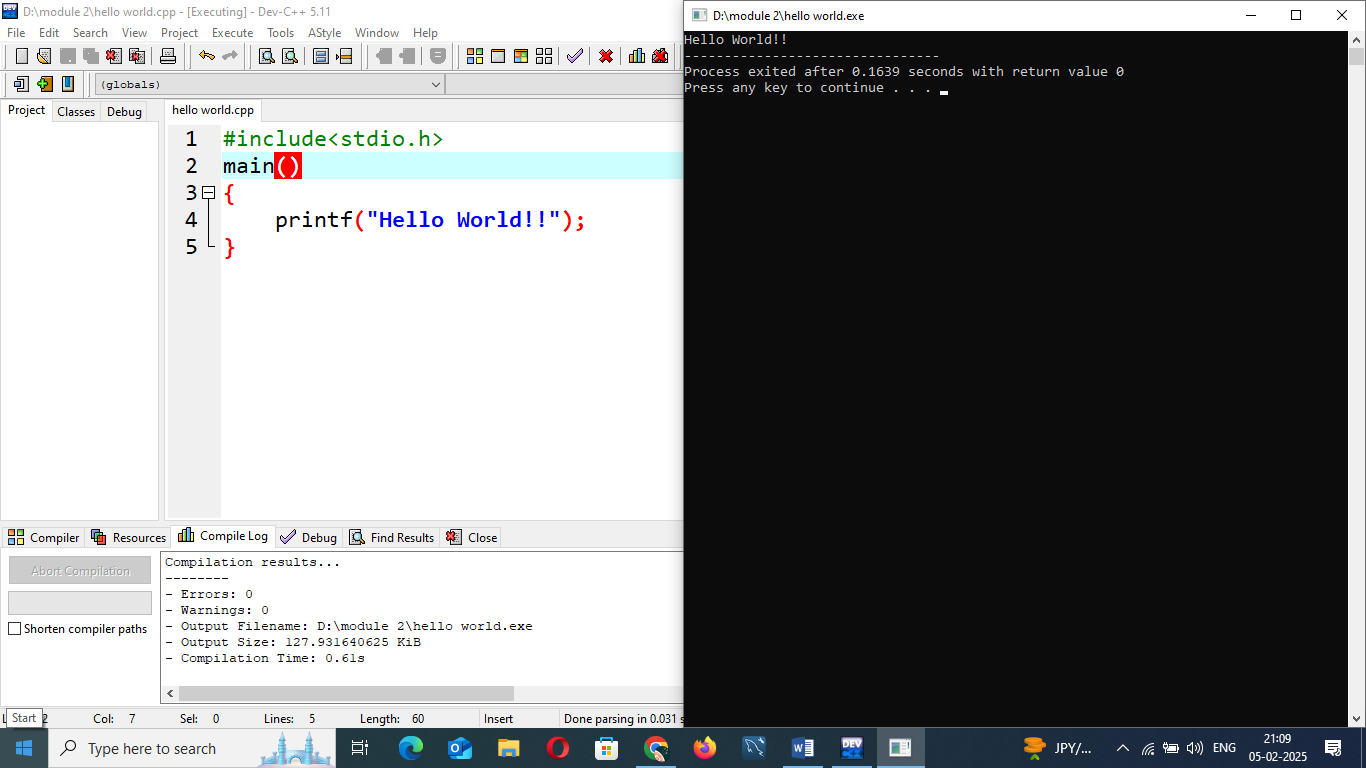
#### => Installing GCC Compiler:

Windows: Install MinGW by downloading it from MinGW-w64 and following the installation instructions.

#### Setting Up an IDE:

* **DevC++**
  1. Download and install DevC++ from a trusted source.
  2. Configure the installed compiler (e.g., MinGW).
* **VS Code**
  1. Install the C/C++ extension from the VS Code marketplace.
  2. Configure tasks and launch settings to compile and run C programs.
* **CodeBlocks**
  1. Download and install CodeBlocks with MinGW.
  2. Open CodeBlocks and set up a new C project.
  3. Write, compile, and run your C programs within the IDE.

**LAB EXERCISE: -Install a C compiler on your system and configure the IDE. Write your first program to print "Hello, World!" and run it.**

=> 

**3. THEORY EXERCISE: -Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.**

=> Headers contain pre-processor directives that include necessary libraries for the program. The most commonly used header is <stdio.h>which provides input and output functions.

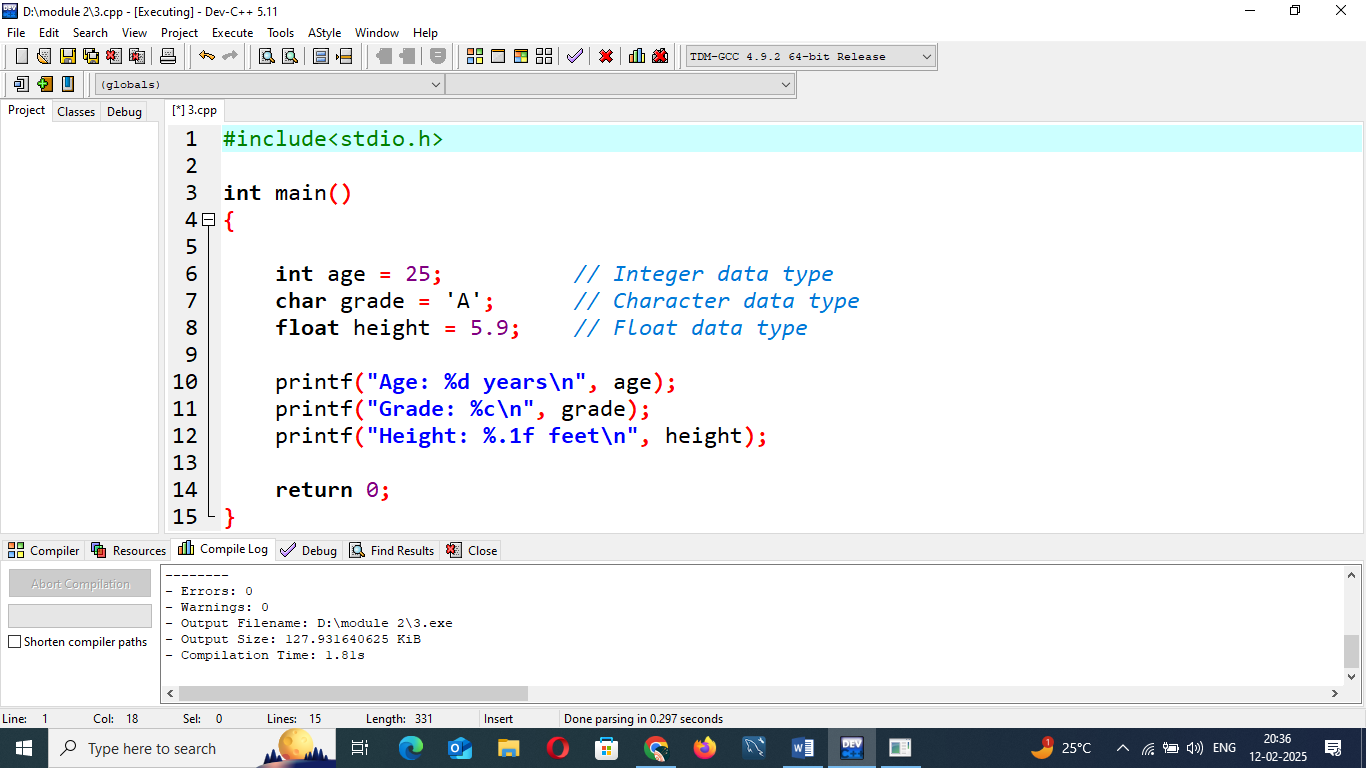
Comments are used to explain code and make it more readable. They are ignored by the compiler.

Every C program must have a main() function. This is the entry point where execution starts.

Variables store data in memory, and each variable has a specific data type. Some common data types in C are:

* int
* float
* char
* double

**LAB EXERCISE: -Write a C program that includes variables, constants, and comments. Declare and use different data types (int, char, float) and display their values.**

=>

**4. THEORY EXERCISE: -Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.**

**=>** 1. **Arithmetic Operators**

Used for basic mathematical operations.

2. **Relational Operators**

Used to compare two values. The result is either 1 (true) or 0 (false).

3. **Logical Operators**

Used to combine multiple relational expressions.

4. **Assignment Operators**

Used to assign values to variables.

5. **Increment and Decrement Operators**

Used to increase or decrease a variable's value by 1.

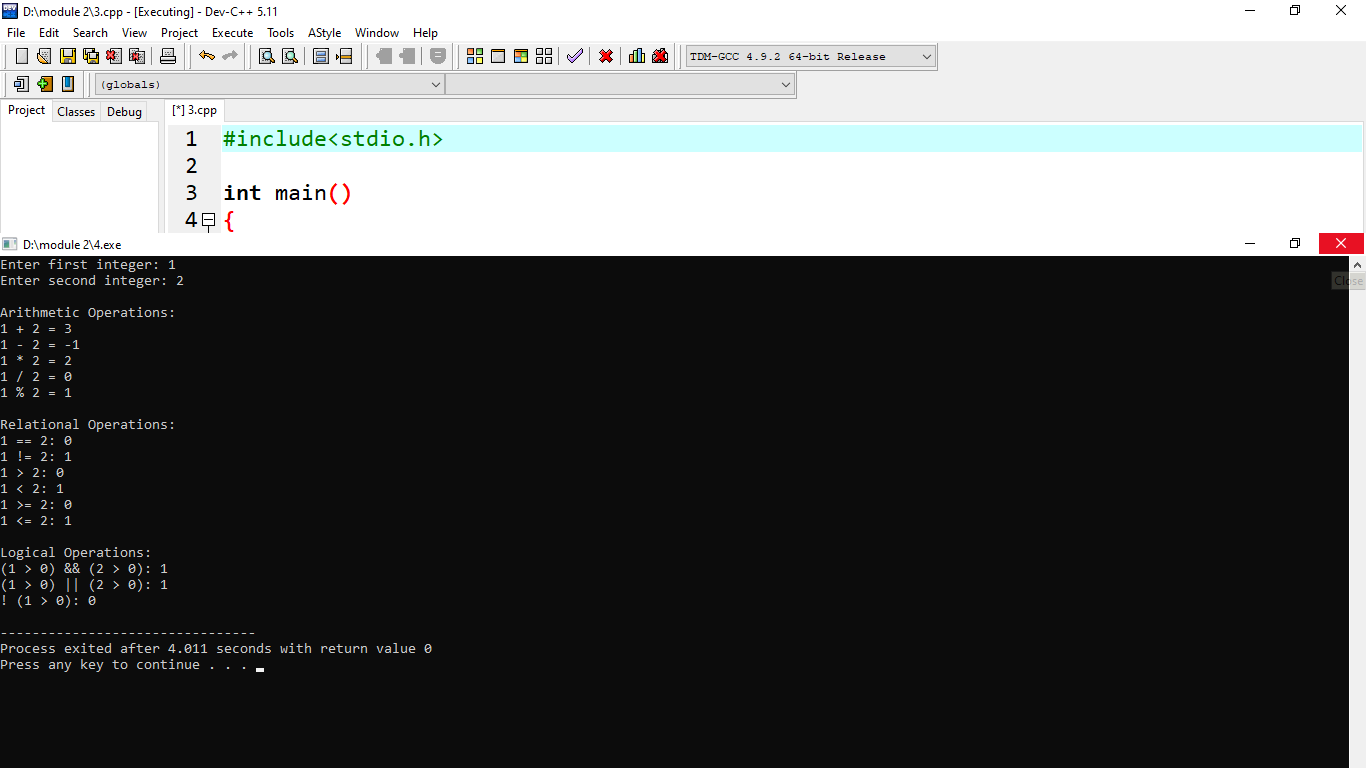
6. **Bitwise Operators**

Operate on binary digits (bits) of integers.

7. **Conditional (Ternary) Operator**

A shorthand for an if-else statement.

**LAB EXERCISE: -Write a C program that accepts two integers from the user and performs arithmetic, relational, and logical operations on them. Display the results.**

**=>**

5. **THEORY EXERCISE: -Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.**

=> 1. If Statement

The if statement evaluates a condition and executes a block of code only if the condition is true.

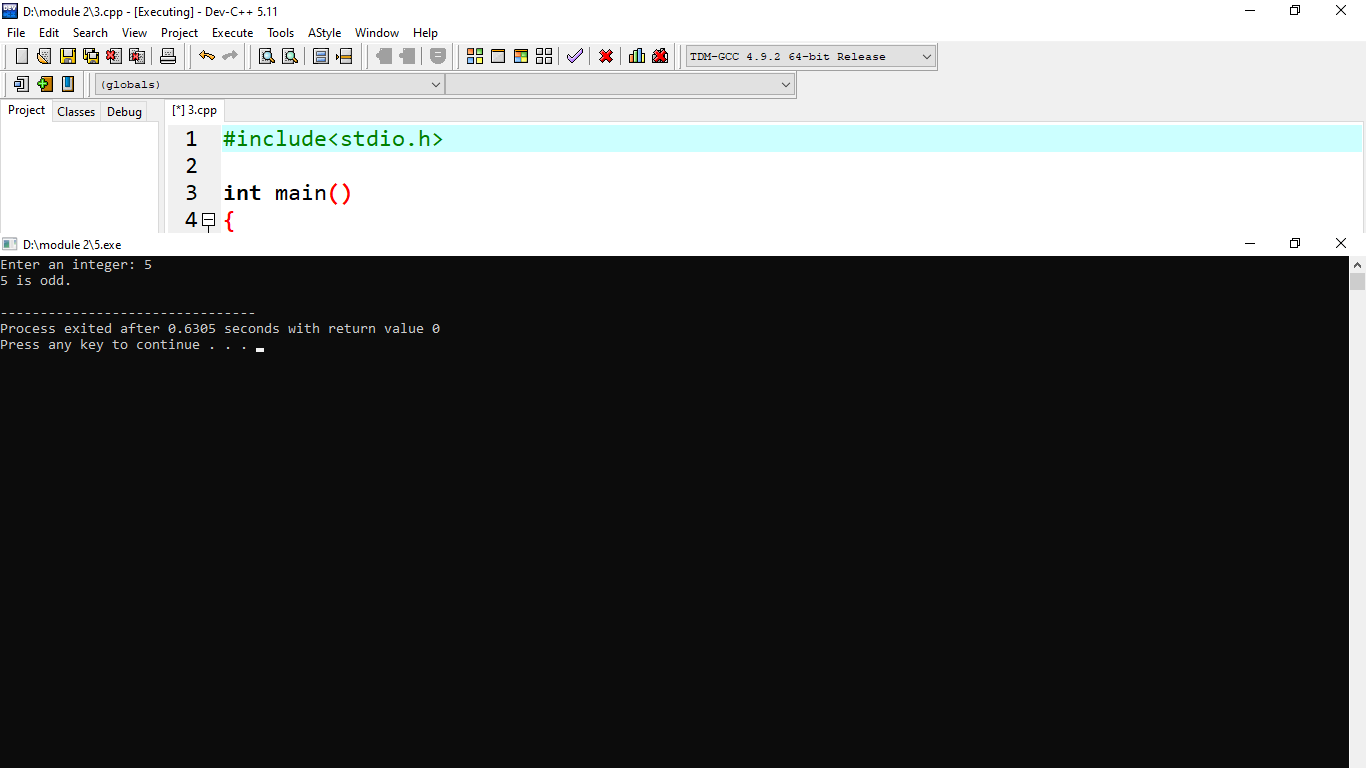
2. if-else Statement

The if-else statement allows the execution of one block of code if the condition is true and another block if the condition is false.

3. Nested if-else Statement

Nested if-else means using one or more if or if-else statements inside another if or else block.

**LAB EXERCISE: - Write a C program to check if a number is even or odd using an if-else statement.**

=>

**6.** **THEORY EXERCISE: -Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.**

**=>** while L**oop**

* Use when the number of iterations is unknown and depends on a condition.
* Example: Reading user input until a valid response is provided.
* Example:

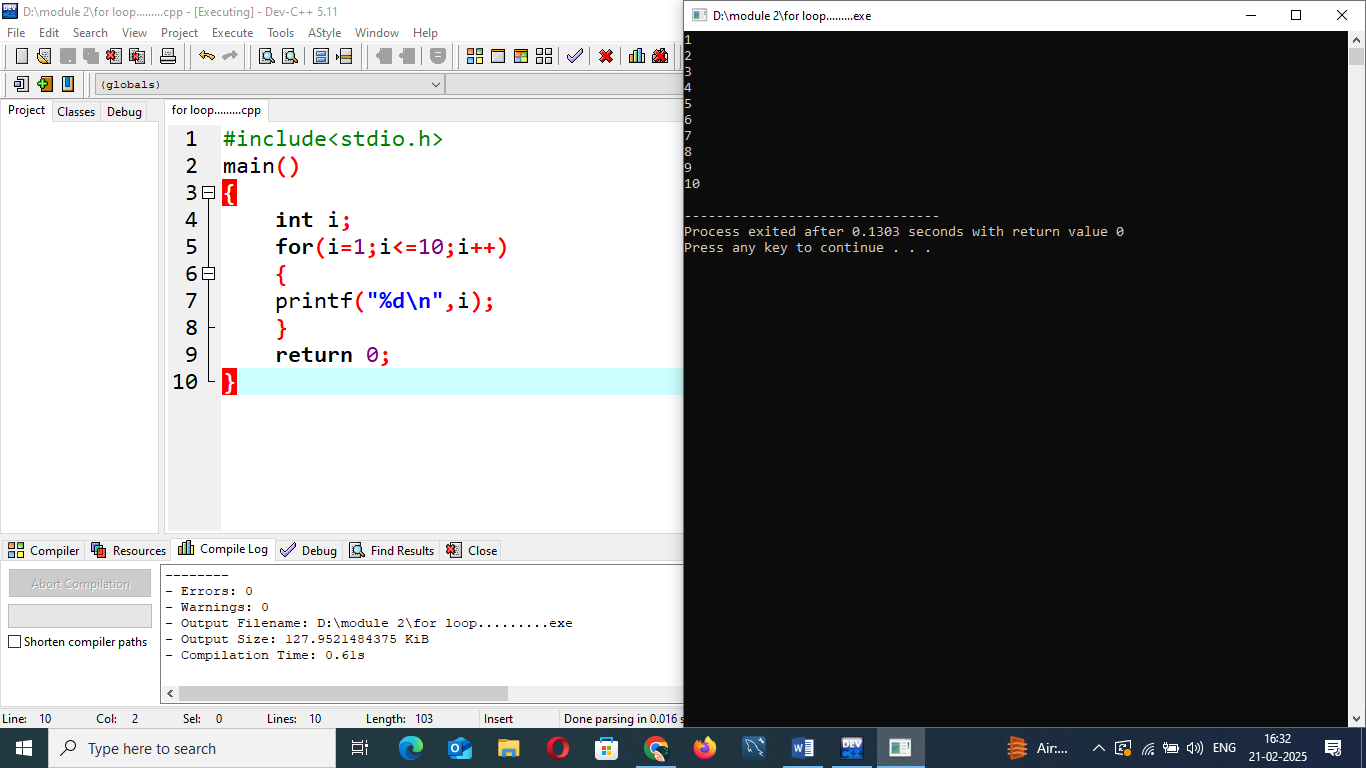
 forLoop

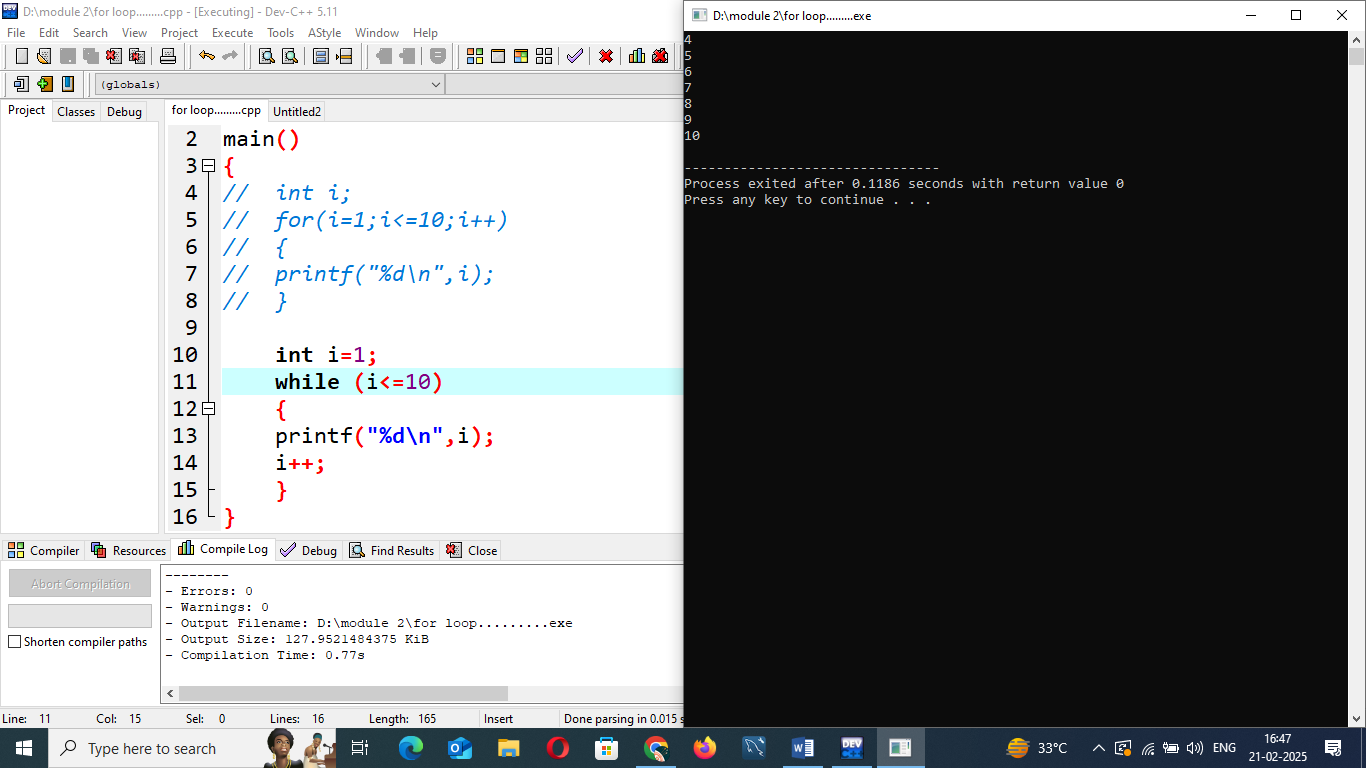
* Use when the number of iterations is predetermined or easy to track.
* Example: Iterating over an array or counting a fixed number of times.
* Example:

 do-while Loop

* Use when the loop must execute at least once, regardless of the condition.
* Example: Displaying a menu at least once and allowing the user to choose to repeat.

**LAB EXERCISE: -Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).**

****

****

**7.** **THEORY EXERCISE: -Explain the use of break, continue, and goto statements in C. Provide examples of each.**

**=>** Break **Statement**

* **Use Case:** Exits a loop or switch statement immediately.
* **Best Used When:** You need to terminate a loop based on a condition.

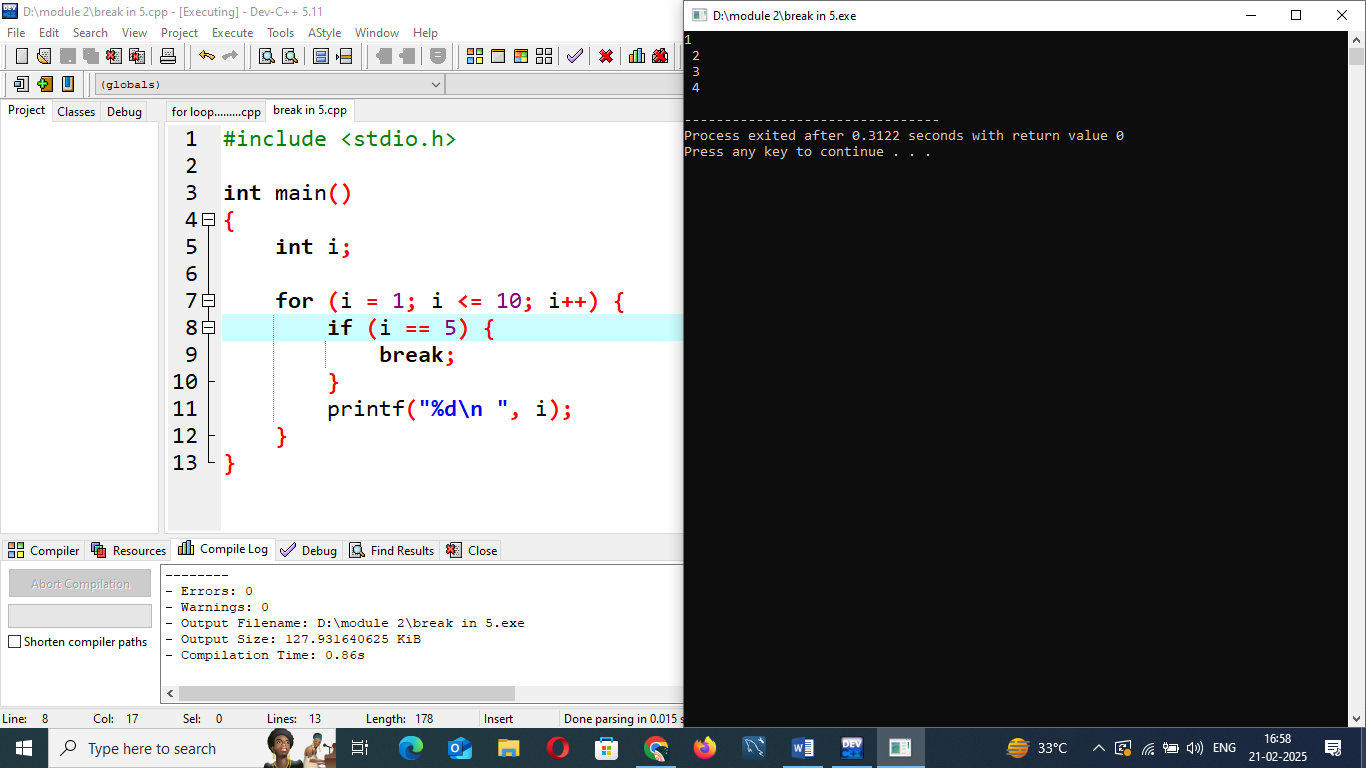
**2.** Continue **Statement**

* **Use Case:** Skips the current iteration and moves to the next one.
* **Best Used When:** You need to ignore specific values but continue the loop.

**3.** goto **Statement**

* **Use Case:** Transfers control to a labeled statement.
* **Best Used When:** You need an emergency jump in specific

**LAB EXERCISE: -Write a C program that uses the break statement to stop printing numbers when it reaches 5.**

****

**8.** **THEORY EXERCISE: -What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.**

**=>** A function in C is a block of code that performs a specific task. Functions help in modular programming, code reusability, and better maintainability.

**Function Declaration (Prototype)**

* Declares a function before its actual definition.
* Specifies the function name, return type, and parameters.
* Allows the compiler to ensure correct function usage.

**Function Definition**

* Contains the actual implementation of the function.
* Must match the declaration in terms of return type and parameters.

**Function Call**

* Invokes the function to execute.
* Must match the function’s parameters.

**9.** **THEORY EXERCISE: -Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.**

=> An **array** in C is a collection of elements of the same data type stored in contiguous memory locations. Arrays allow efficient storage and manipulation of multiple values using a single variable name. Each element in the array is accessed using an **index**, which starts from **0**.

**1. One-Dimensional Arrays (1D Arrays)**

A **one-dimensional array** is a linear collection of elements of the same data type, accessible using a single index.

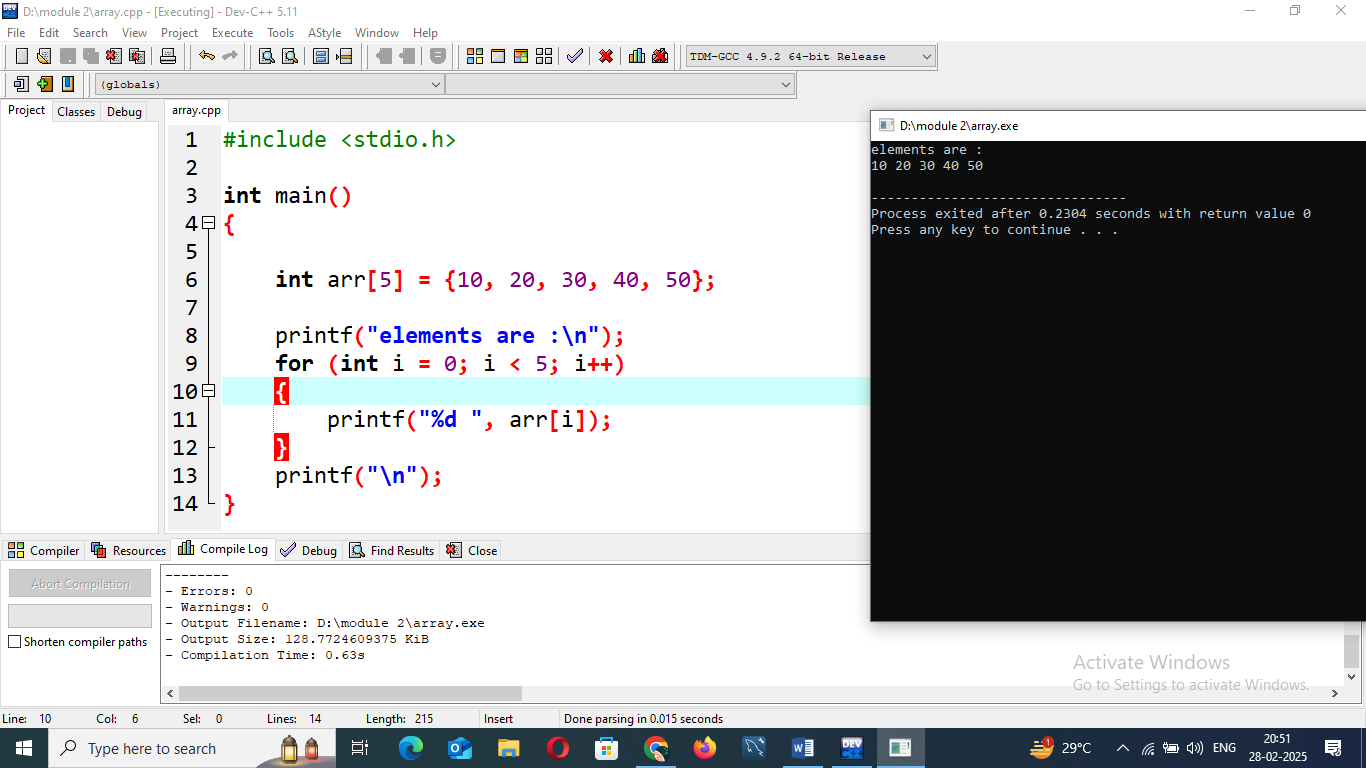
**2. Multi-Dimensional Arrays**

Multi-dimensional arrays store data in a tabular format and require **multiple indices** to access elements.

#### **2D Arrays (Matrix)**

A **two-dimensional array** is an array of arrays, often used to store matrices.

**LAB EXERCISE: -Write a C program that stores 5 integers in a one-dimensional array and prints them.**

**=>**

**10. THEORY EXERCISE: -Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?**

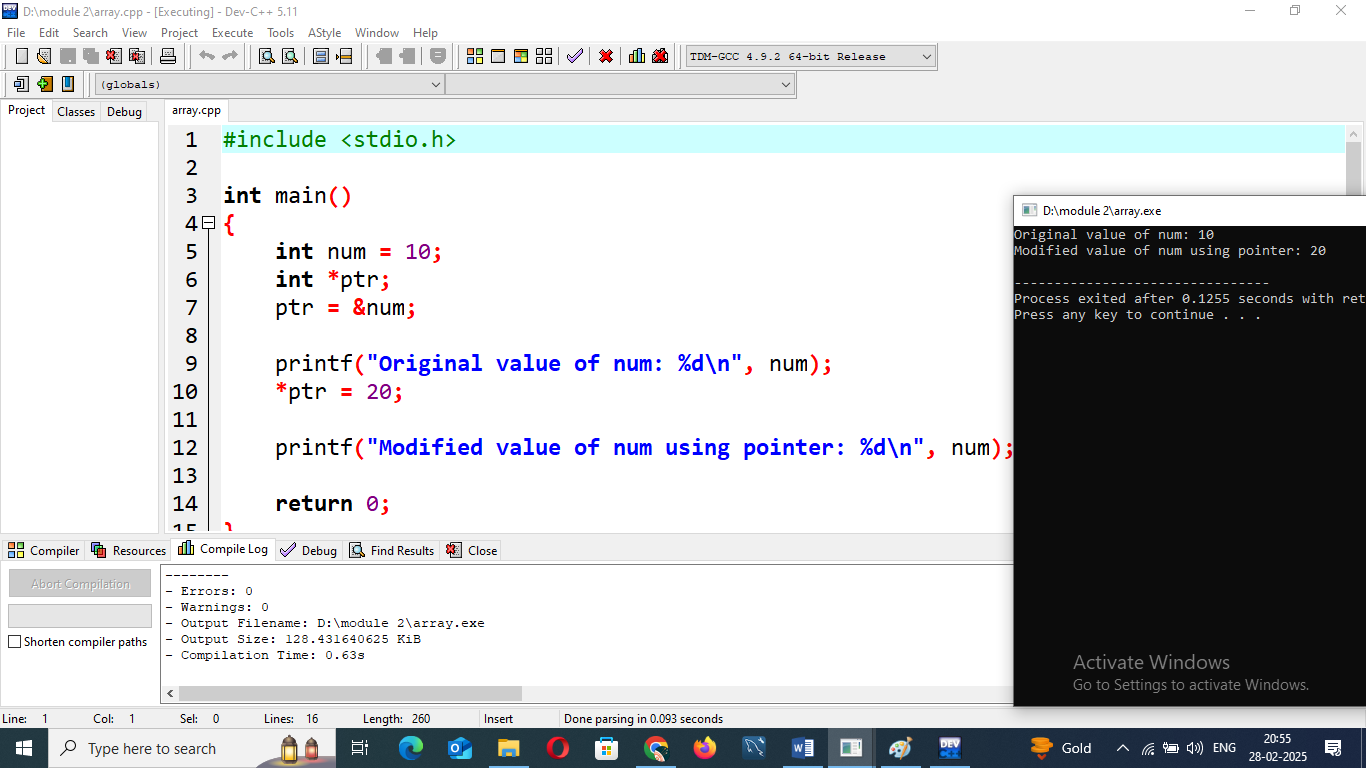
=> A **pointer** in C is a variable that stores the **memory address** of another variable. Instead of storing an actual value, it holds the location where a value is stored.

A pointer is declared using the \* symbol.

**Importance of Pointers in C**

1. **Efficient Memory Management**
   * Pointers allow **direct memory access** and manipulation, making C powerful in low-level programming.
2. **Function Argument Passing (Call by Reference)**
   * Enables **modification of variables** inside functions without returning values explicitly.

**LAB EXERCISE: -Write a C program to demonstrate pointer usage. Use a pointer to modify the value of a variable and print the result.**

**=>**

**12.** **THEORY EXERCISE: -Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.**

=> Declaring Structure Variables

After declaring the structure, you can create variables of that structure type.

Initializing Structure Variables

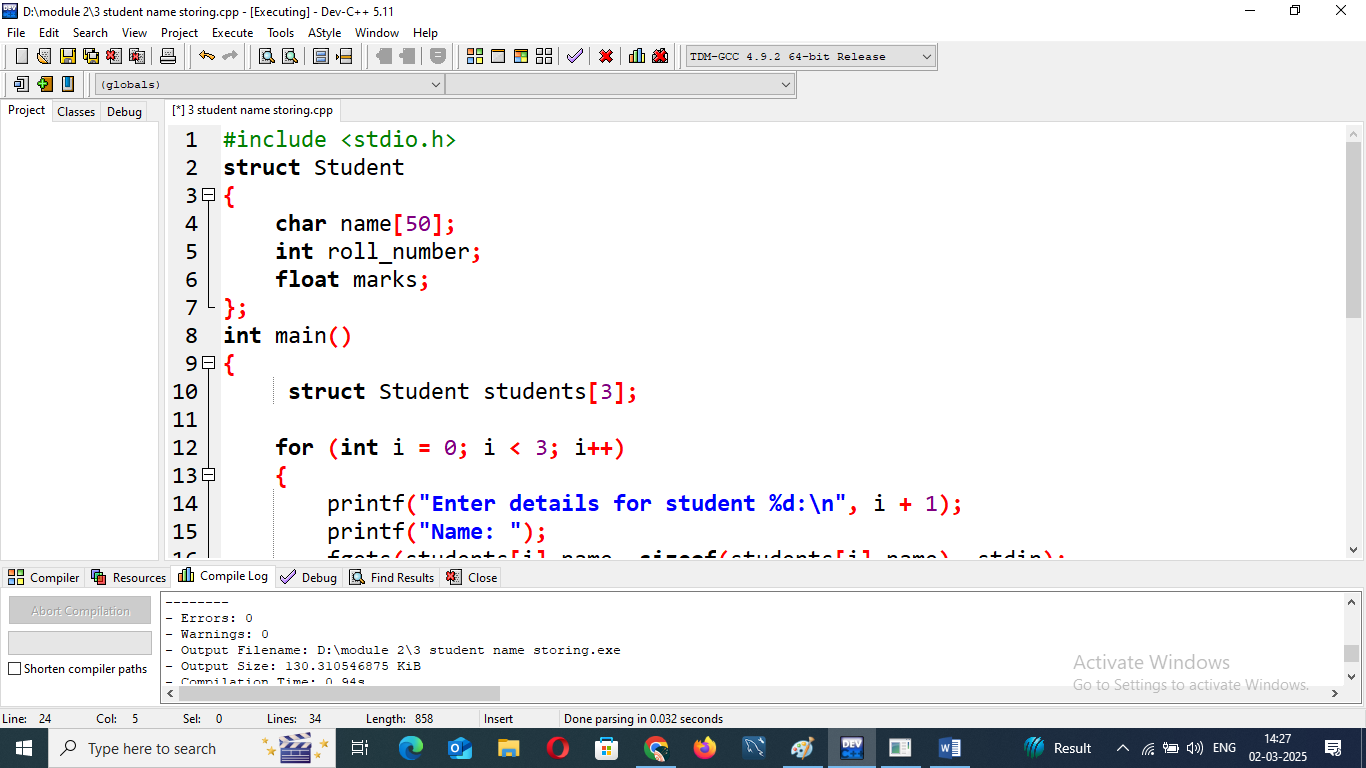
There are several ways to initialize a structure:

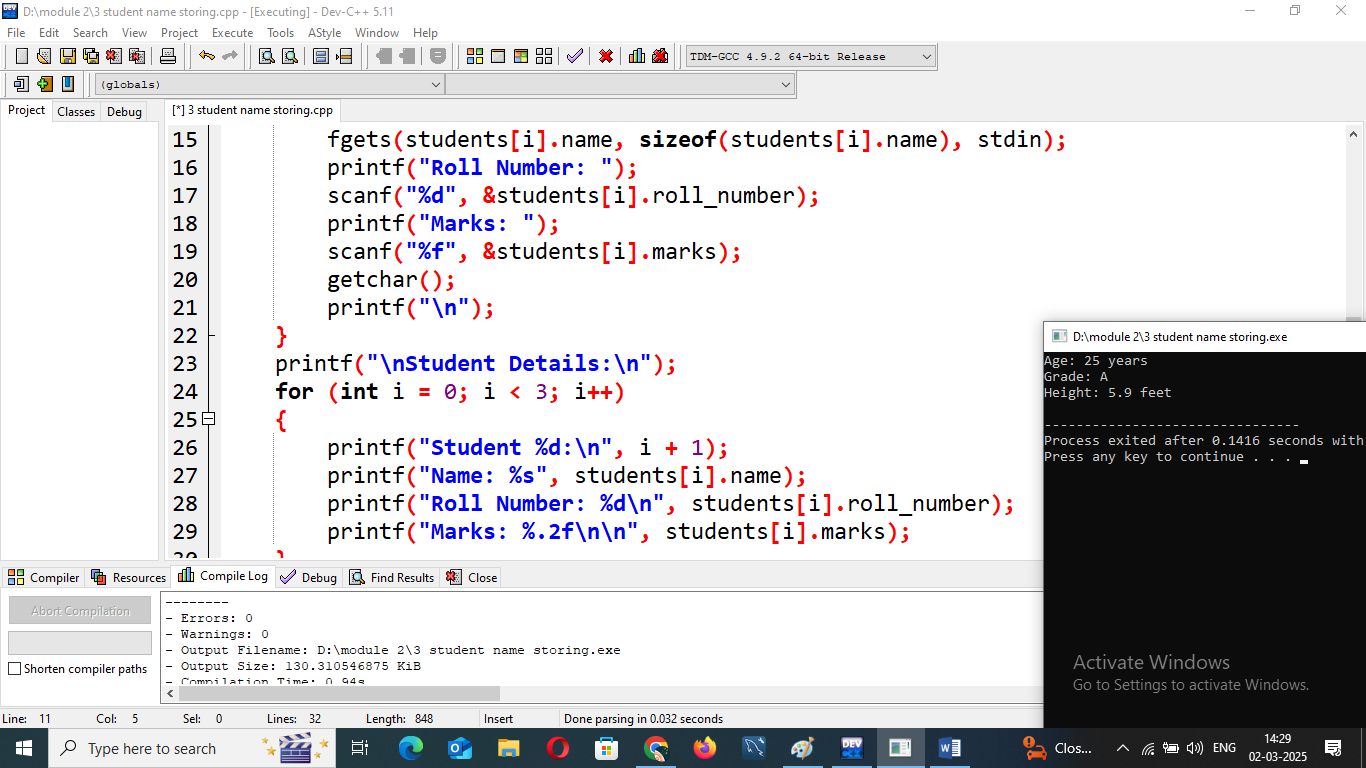
1. **At the time of declaration** using an initializer list.
2. **Individually** after declaration by accessing members.

Accessing Structure Members

To access members of a structure, you use the **(**.**)** when dealing with a structure variable.

**LAB EXERCISE: -Write a C program that defines a structure to store a student's details (name, roll number, and marks). Use an array of structures to store details of 3 students and print them.**

**=>**

****

**13. THEORY EXERCISE: -Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.**

**=>** File handling in C is essential for **storing, retrieving, and manipulating data** persistently. Unlike standard input/output operations (e.g., scanf and printf), which handle temporary data, file handling allows users to store data permanently for later use.

**O**pening a File

Before performing any operation on a file, it must be opened using the fopen() function.

**Closing a File**

After performing file operations, it's good practice to close the file using fclose() to free resources.

**Writing to a File**

To write data to a file, we use functions like:

* fputc() → Writes a single character.
* fputs() → Writes a string.
* fprintf() → Writes formatted data.
* fwrite() → Writes binary data.

**Reading from a File**

To read data from a file, we use functions like:

* fgetc() → Reads a single character.
* fgets() → Reads a string.
* fscanf() → Reads formatted data.
* fread() → Reads binary data.

**LAB EXERCISE: -Write a C program to create a file, write a string into it, close the file, then open the file again to read and display its contents.**

**=>no idea!!!**