

Programming Fundamentals

Lab Manual

Version 1.0



Institute of Management Sciences

Peshawar

Prepared by: Kaisar Khan

Lab Instructor

"The only way to learn a new programming language is by writing programs in it." - Dennis Ritchie

Lab Contents:

1.	IDE Installation	5
2.	Exploring cout and Text Formatting in C++.....	11
3.	Understanding Literals in C++	15
4.	Exploring Arithmetic Operators in C++	18
5.	Exploring Variables and Arithmetic Operations in C++	23
6.	Taking User Inputs in C++	28
7.	Conditional Statements in C++	32
8.	Logical Operators.....	37
9.	Logical Operators.....	43
10.	While Loop in C++	50
11.	Do While Loop	53
12.	For Loop in C++	57
13.	Nested For Loops in C++.....	61
14.	User Defined Functions in C++	65
15.	Arrays in C++	71
16.	2D Arrays in C++	78
17.	Pointers in C++	83
18.	File Handling in C++	87
19.	Suggested Small Semester Projects	93

How to Use This Lab Manual for Learning and Practice?

- 1. Engage with Lab Exercises:** Carefully read through each lab exercise, focusing on the learning objectives. Implement the provided code examples and modify them to see how changes affect the output. This hands-on practice will reinforce your understanding of key programming concepts.
- 2. Utilize Practice Tasks:** After completing the lab exercises, tackle the practice tasks included in the manual. These tasks are designed to challenge your skills and help you apply what you've learned. Attempt to solve them independently before reviewing the solutions.
- 3. Work on Suggested Projects:** Take advantage of the suggested semester projects at the end of the manual. These projects provide an opportunity to apply your knowledge in a comprehensive manner. Collaborate with peers or work individually to enhance your problem-solving skills and creativity.

Lab 01

IDE Installation

Learning Objectives:

By the end of this lab, students will be able to download, install, and configure Dev C++ IDE, allowing them to write, compile, and run basic C++ programs to verify IDE Working.

Requirements:

1. Windows Computer with Internet Access.
2. Installation File for Dev C++ IDE. (**Download** from here: <https://www.bloodshed.net/>)

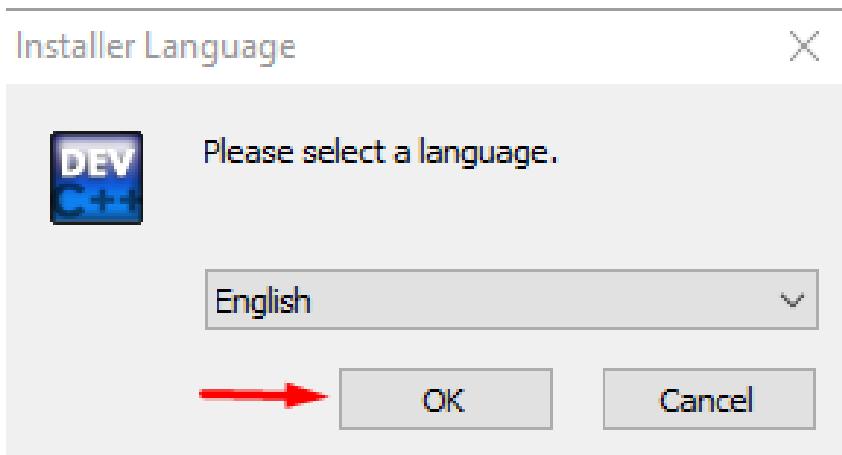
Lab Exercises:

1. Dev C++ IDE Installation
2. Creating and Running Your First C++ Program

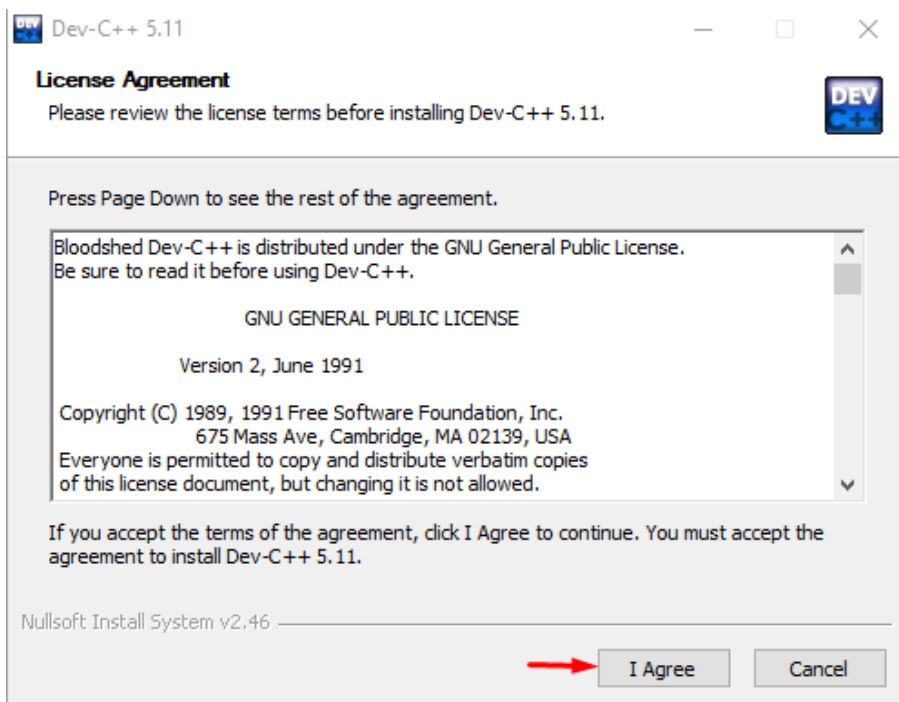
Exercise 1: Dev C++ IDE Installation

Once you download the installation file, double-click it to start the Dev C++ IDE installation. Follow these steps:

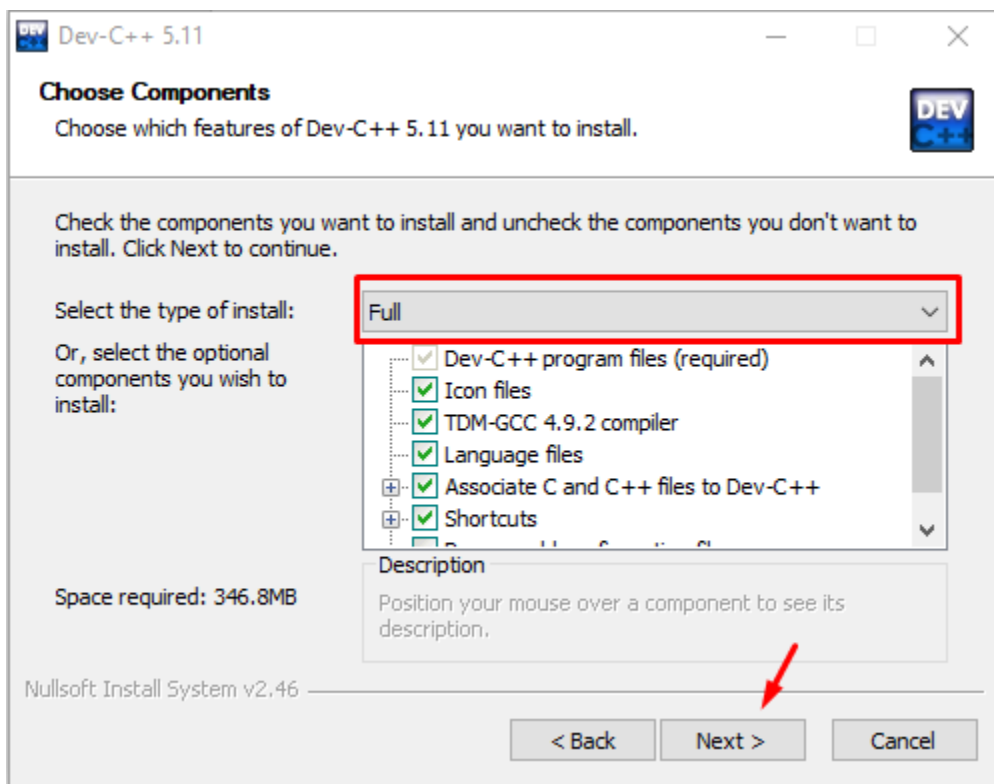
1. Select “English” as the Installer Language and click OK.



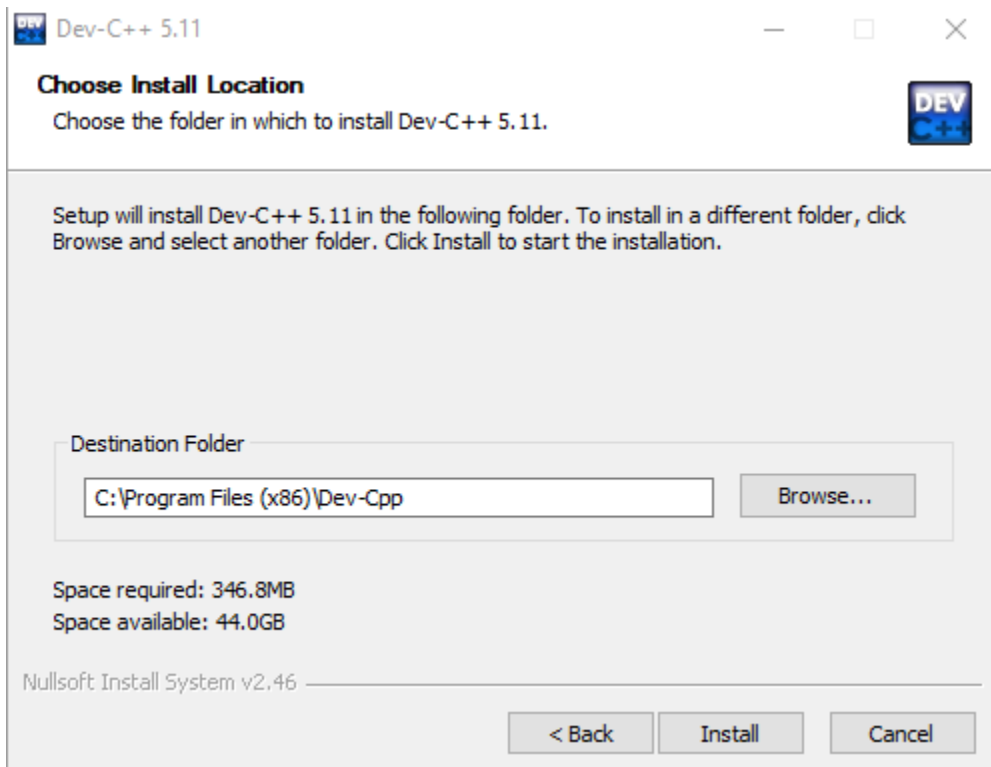
2. Click I Agree to accept the License Agreement.



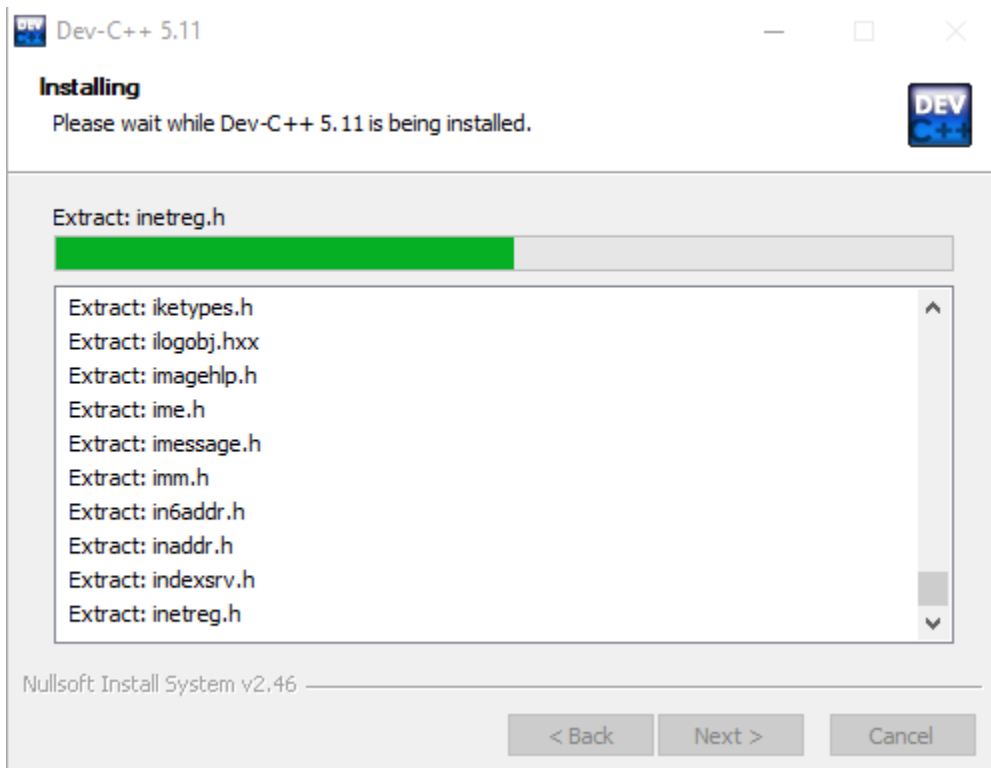
3. Choose FULL for the installation type and click Next.



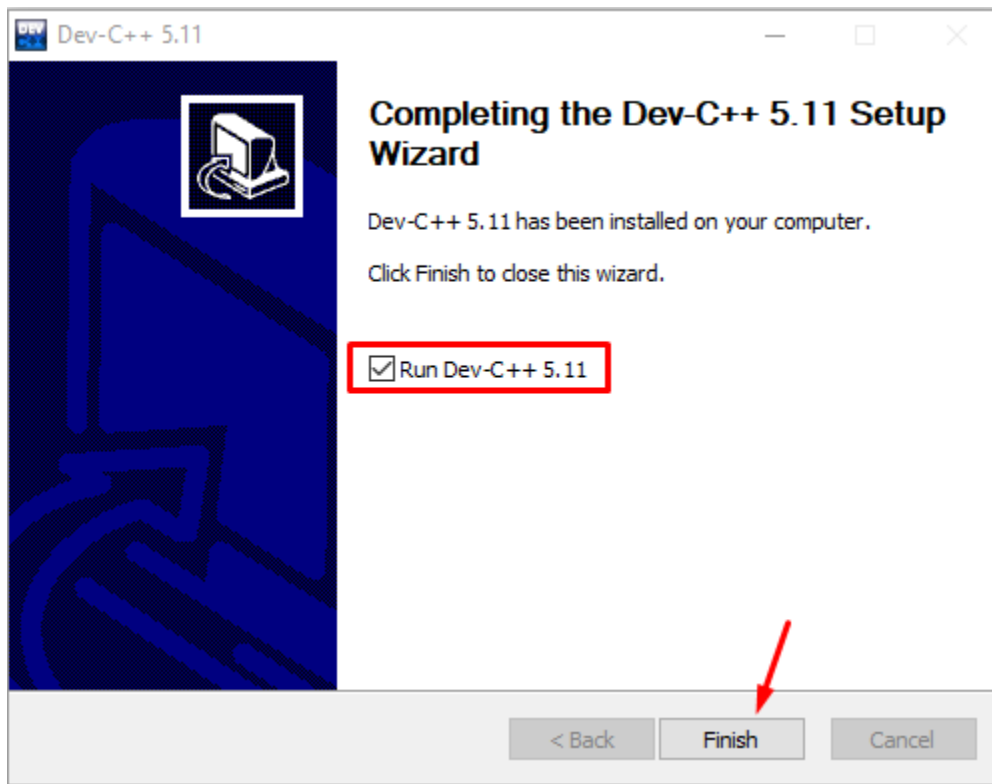
4. Accept the default install location and click Install to begin.



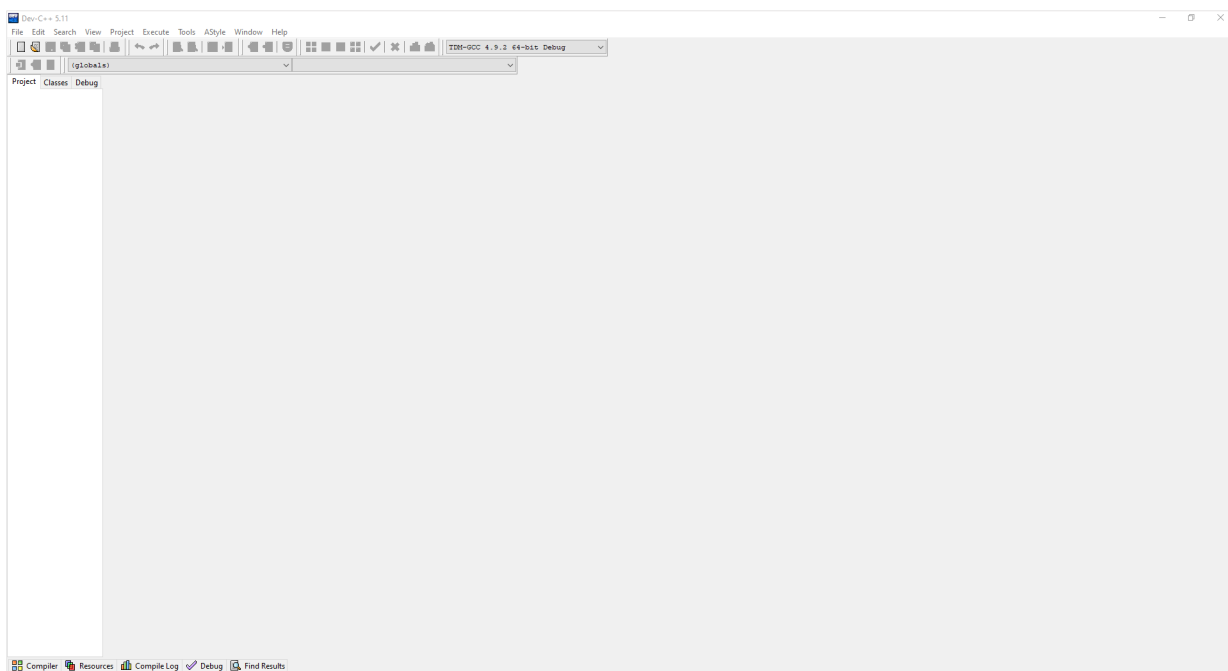
The Installation will get started, wait for it to complete:



- Once installation is complete, check the box for “Run Dev C++” and click Finish.



Dev C++ will launch upon completion.



Exercise 2: Creating and Running Your First C++ Program

In this exercise, you will create and run a simple C++ program using the Dev C++ IDE.

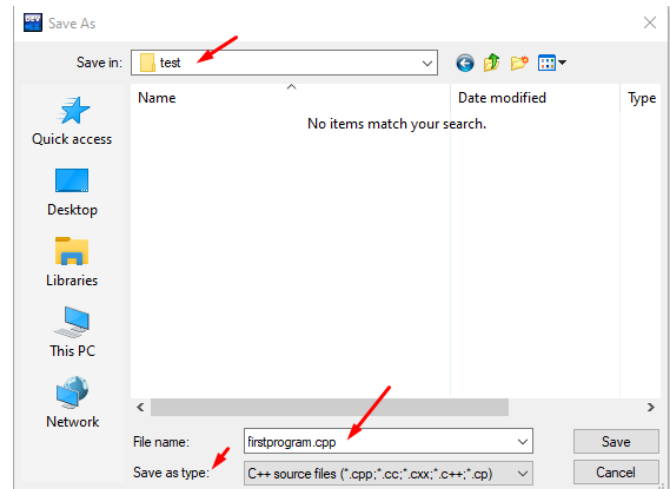
Steps:

1. **Open Dev C++:** Launch the Dev C++ application on your computer.
2. **Create a New Source File:**
 1. Click on the File menu.
 2. Select New, then choose Source File.
 3. A new area will open where you can write your C++ code.
3. **Write Your C++ Code:**

For testing purposes, copy and paste the following code into the new source file:

```
#include <iostream>
using namespace std;
int main(){
    cout << "HelloWorld!" <<endl;
    return 0;
}
```

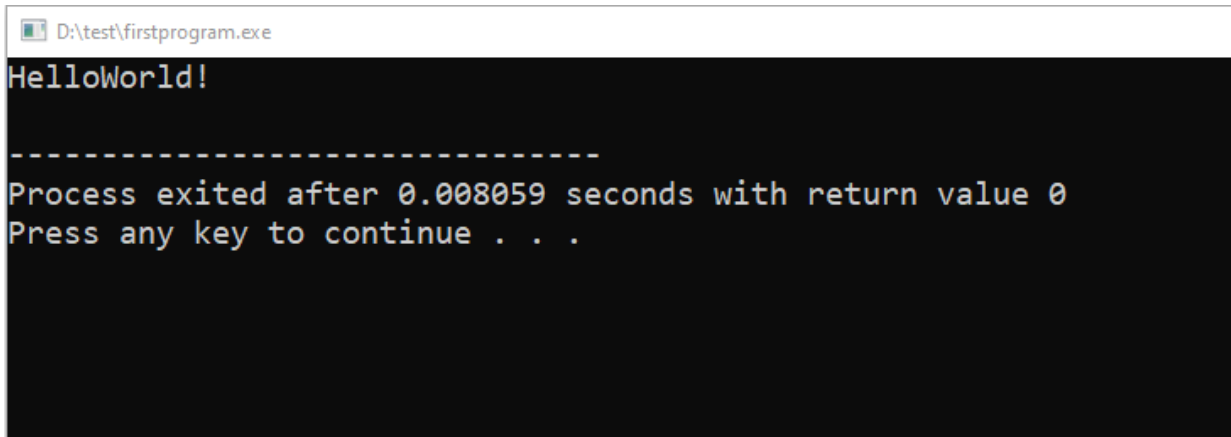
4. **Save Your File:**
 1. Click on the File menu again.
 2. Select Save or Save As.
 3. A dialog box will appear. Choose the location where you want to save the file.
 4. In the file name field, enter: firstprogram.cpp.
 5. Click the Save button.



5. **Compile and Run Your Program:**
 1. Click on the Execute menu.
 2. Select Compile and Run.
 3. If there are no errors in your code or issues with the installation, the program should execute successfully.

6. Verify Output:

A console window will appear displaying: HelloWorld!.

A screenshot of a Windows console window. The title bar at the top shows a small icon and the text "D:\test\firstprogram.exe". The main area of the window has a black background with white text. The first line of text is "HelloWorld!". Below this is a dashed line. The next line of text is "Process exited after 0.008059 seconds with return value 0". The final line of text is "Press any key to continue . . .".

```
D:\test\firstprogram.exe
HelloWorld!
-----
Process exited after 0.008059 seconds with return value 0
Press any key to continue . . .
```

This indicates that your first program has run successfully and that Dev C++ is properly installed and configured.

Lab 02

Exploring cout and Text Formatting in C++

Learning Objectives:

By the end of this lab, students will be able to use the cout statement to display text in various styles and formats, enhancing their understanding of output in C++.

Lab Exercises:

1. Basic Output with cout
 2. Styling Output
 3. Use of Escape Sequences
-

Exercise 1: Basic Output with Cout

`cout` is a way to print messages or data on the screen in C++. You use it with `<<` to show what you want to display. For example, `cout << "Hello!"` prints "Hello!" on the console. It's part of C++'s built-in tools for showing information to users.

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

Exercise 2: Styling Outputs:

```
#include <iostream>
using namespace std;

int main() {
    cout << "*****" << endl;
    cout << "*" << endl;
    cout << "    Welcome to C++ World!    " << endl;
    cout << "*" << endl;
    cout << "*****" << endl;

    cout << endl; // Blank line for spacing

    cout << "Please choose an option:" << endl;
    cout << "1. Start a new project" << endl;
    cout << "2. Load existing project" << endl;
    cout << "3. Exit" << endl;

    cout << endl; // Blank line for spacing

    cout << "=====" << endl;
    cout << "    Thank you for using our    " << endl;
    cout << "        program!            " << endl;
    cout << "=====" << endl;

    return 0;
}
```

Exercise 3: Use of Escape Sequences

Escape sequences are special characters used in programming to represent certain actions or formatting in strings. They begin with a backslash (\) followed by a character. In C++, escape sequences allow you to include characters in strings that would otherwise be difficult to type or that have special meanings. These sequences enhance the readability and formatting of output in console applications.

Here are some common escape sequences:

\n: New line - moves the cursor to the beginning of the next line.

\t: Horizontal tab - inserts a tab space, creating an indentation.

\\: Backslash - allows you to include a backslash in the string.

\": Double quote - lets you include double quotes inside a string.

\r: Carriage return - moves the cursor to the beginning of the line (overwrites the current line).

\b: Backspace - deletes the previous character in the output.

\f: Form feed - advances the cursor to the next page (rarely used).

Code:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Line 1: Hello, World!\n";
    cout << "Line 2: Hello,\tWorld!\n";
    cout << "Line 3: This is a backslash: \\n";
    cout << "Line 4: This is a quote: \"Hello!\"\n";
    cout << "Line 5: Carriage return:\rOverwrite\n";
    cout << "Line 6: Backspace:\bHello\n";
    cout << "Line 7: Form feed (not visible):\fNext Line\n";

    return 0;
}
```

Practice Tasks:

1. Write a program that displays a simple menu using escape sequences. Each menu item should be separated by a tab. The menu should look like this:

Menu:

1. Start
2. Game
3. Load
4. Game
5. Options
6. Exit

2. Create a program that prints a famous quote. Use escape sequences to include quotes within the output. The quote should look like this:

```
Albert Einstein once said, "Imagination is more important than knowledge."
```

3. Write a program that demonstrates the use of escape sequences. The program should output the following formatted text:

Name	Age	City
John	25	New York
Alice	30	Los Angeles
Bob	22	Chicago

Lab 03

Understanding Literals in C++

Learning Objectives:

By the end of this lab, students will be able to identify and utilize different types of literals in C++, including integer, floating-point, Boolean, and string literals. They will also learn how to output these literals using `cout`

Lab Exercises:

1. Demonstrating Various Data Types of Literals in C++
2. Arithmetic Operations with Literals

Exercise 1: Integer and Floating-Point Literals

A literal is a fixed value written directly in the code, representing data types such as integers, floating-point numbers, Booleans, and strings. Integer literals can be positive, negative, or large values, while floating-point literals include decimals and can be represented in standard or scientific notation. Understanding how to use these literals effectively allows programmers to initialize variables and perform calculations directly. By practicing with different types of literals, students will gain a deeper appreciation for how C++ handles data and numerical representation.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Integer literals
    cout << "Integer Literal: " << 42 << endl;
    cout << "Negative Integer Literal: " << -15 << endl;
    cout << "Large Integer Literal: " << 8574210368 << endl;

    // Floating-point literals
    cout << "Float Literal: " << 3.14f << endl;
```

```
cout << "Double Literal: " << 2.718281828459045 << endl;
cout << "Scientific Notation Literal: " << 1.23e4 << endl;

// Boolean literals
cout << "Boolean True: " << true << endl;
cout << "Boolean False: " << false << endl;

// Character literal
cout << "Character Literal: " << 'A' << endl;

// String literal
cout << "String Literal: " << "Hello, World!" << endl;

return 0;
}
```


Exercise 2: Arithmetic Operations with Literals

Arithmetic expressions in C++ are combinations of numeric literals and operators that perform mathematical calculations. These expressions can include basic operations such as addition, subtraction, multiplication, division, and modulus. When evaluated, they yield a numeric result that can be displayed or utilized in further calculations. Understanding how to use arithmetic expressions effectively allows programmers to perform computations directly in their code.

Code:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Addition of 5 and 10: " << (5 + 10) << endl;
    cout << "Subtraction of 15 and 3: " << (15 - 3) << endl;
    cout << "Multiplication of 4 and 7: " << (4 * 7) << endl;
    cout << "Division of 20 by 5: " << (20 / 5) << endl;
    cout << "Modulus of 22 and 7: " << (22 % 7) << endl;

    return 0;
}
```

Practice Tasks:

1. Create a program that combines multiple arithmetic operations in a single line for the following expression and displays the result: $(10 + 5) \times (20 - 30) \div 2$
2. Write a program that calculates the area and perimeter of a rectangle using literals for its length and width. Use the following values:

Length: 15, Width: 10

Output both the area and perimeter with descriptive messages.

3. Write a program that demonstrates the use of character and string literals. Print a sentence that includes a character literal and ends with a string literal, such as:

"The grade for A is excellent!"

Lab 04

Exploring Arithmetic Operators in C++

Learning Objectives:

By the end of this lab, students will understand the various arithmetic operators available in C++ and how to perform mathematical operations using numeric literals. They will explore operations such as addition, subtraction, multiplication, division, modulus, and more.

Lab Exercises:

1. Basic and Floating-point Operations
2. Modulus and Negative Numbers
3. Demonstrating BODMAS/BIDMAS Operations in C++

Exercise 1: Basic and Floating-point Operations

Write a program that demonstrates basic arithmetic operations using integer literals, including addition, subtraction, multiplication, and division. Additionally, showcase floating-point arithmetic with at least three operations (addition, subtraction, and division) using floating-point literals.

```
#include <iostream>
using namespace std;

int main() {
    // Integer arithmetic operations
    cout << "Integer Arithmetic Operations:" << endl;
    cout << "Addition: " << 10 + 5 << endl;           // 10 + 5
    cout << "Subtraction: " << 10 - 5 << endl;         // 10 - 5
    cout << "Multiplication: " << 10 * 5 << endl;       // 10 * 5
    cout << "Division: " << 10 / 5 << endl << endl;     // 10 / 5

    // Floating-point arithmetic operations
    cout << "Floating-point Arithmetic Operations:" << endl;
    cout << "Addition: " << 10.5 + 5.3 << endl;         // 10.5 + 5.3
    cout << "Subtraction: " << 10.5 - 3.2 << endl;      // 10.5 - 3.2
    cout << "Division: " << 7.5 / 2.5 << endl;          // 7.5 / 2.5
}
```

```
    return 0;
}
```

Exercise 2: Modulus and Negative Numbers

```
#include <iostream>
using namespace std;

int main() {
    // Modulus operations
    cout << "Modulus Operations:" << endl;
    cout << "10 % 3 = " << 10 % 3 << endl;           // Positive dividend
    cout << "11 % 5 = " << 11 % 5 << endl;           // Positive dividend
    cout << "-10 % 3 = " << -10 % 3 << endl;          // Negative dividend
    cout << "10 % -3 = " << 10 % -3 << endl;          // Negative divisor
    cout << "-10 % -3 = " << -10 % -3 << endl;        // Negative dividend/divisor
    cout << endl;

    // Demonstrating arithmetic operations with negative numbers
    cout << "Arithmetic Operations with Negative Numbers:" << endl;
    cout << "Addition with Negative: " << -10 + 5 << endl;      // -10 + 5
    cout << "Subtraction with Negative: " << 5 - (-10) << endl;  // 5 - (-10)
    cout << "Multiplication with Negative: " << -5 * 3 << endl;  // -5 * 3
    cout << "Division with Negative: " << -10 / 2 << endl;       // -10 / 2

    return 0;
}
```

Exercise 3: Demonstrating BODMAS/BIDMAS Operations in C++

BODMAS (or BIDMAS) is an acronym that helps remember the order of operations used in arithmetic calculations. It stands for:

- **Brackets:** Solve anything inside brackets first.
- **Orders** (or Indices): Calculate powers and roots next (e.g., squares, cubes).
- **Division:** Perform division operations from left to right.
- **Multiplication:** Perform multiplication operations from left to right.
- **Addition:** Perform addition operations from left to right.
- **Subtraction:** Perform subtraction operations from left to right.

Understanding BODMAS is crucial because it ensures that mathematical expressions are evaluated consistently and correctly, leading to the intended results.

For example, in the expression $2+3\times 4$, multiplication is performed before addition, resulting in $2+12=14$, not $5\times 4=20$.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Demonstrating BODMAS operations
    cout << "BODMAS/BIDMAS Operations:" << endl;

    // Brackets
    cout << "1. (10 + 5) * 2 = " << (10 + 5) * 2 << endl;
    // (10 + 5) * 2
    cout << "2. 10 + (5 * 2) = " << 10 + (5 * 2) << endl;
    // 10 + (5 * 2)

    // Orders (exponents)
    cout << "3. 2^3 + 5 = " << (2 * 2 * 2) + 5 << " (2^3 calculated
as 2 * 2 * 2)" << endl; // 2^3 + 5
```

```

    // Division and Multiplication
    cout << "4. 20 / 4 * 2 = " << 20 / 4 * 2 << endl;
// 20 / 4 * 2
    cout << "5. 20 / (4 * 2) = " << 20 / (4 * 2) << endl;
// 20 / (4 * 2)

    // Addition and Subtraction
    cout << "6. 10 + 5 - 3 = " << 10 + 5 - 3 << endl;
// 10 + 5 - 3
    cout << "7. 10 - (3 + 2) = " << 10 - (3 + 2) << endl;
// 10 - (3 + 2)

    // Complex Expression
    cout << "8. (10 + 5) * 2 - 3 / 3 = " << (10 + 5) * 2 - 3 / 3 <<
endl;    // Complex expression

    return 0;
}

```

Practice Tasks:

1. Write a program that calculates the average of three integers and three floating-point numbers. Display the results with appropriate messages, and ensure to use correct formatting for the floating-point output.
2. Write a program that calculates the simple interest for a given principal amount, rate of interest, and time period. **Use the formula:**

$$\text{Simple Interest} = \frac{\text{Principal} \times \text{Rate} \times \text{Time}}{100}$$

Use the following literal values:

- Principal: 1000
- Rate: 5
- Time: 3

Example Output:

Simple Interest for Principal 1000, Rate 5%, and Time 3 years: 150

3. Write a program that calculates the distance traveled based on the speed of a vehicle and the time it has been traveling, using the formula:

$$\text{Distance} = \text{Speed} \times \text{Time}$$

Use the following literal values: Speed = 60 km/h and Time = 2.5 hours.

Display the calculated distance on the console.

4. Write a program that calculates the area of a circle using the formula $\text{Area} = \pi r^2$.
Use the following literal values: pi = 3.14 and radius r = 5.
Display the calculated area using `cout`.

Lab 05

Exploring Variables and Arithmetic Operations in C++

Learning Objectives:

By the end of this lab, students will understand the fundamentals of variables in C++, including their declaration, initialization, and characteristics such as scope and lifetime. They will identify and utilize different data types, including integers, floats, and characters, while performing basic arithmetic operations on these variables. Students will learn to implement and differentiate between post-increment and pre-increment operators, as well as apply compound assignment operators to simplify arithmetic expressions.

Lab Exercises:

1. Variable Initialization and Arithmetic Operations in C++
2. Calculating the Area of Shapes Using Variables and Arithmetic Operations

Exercise 1: Variable Initialization and Arithmetic Operations in C++

1. **Variable Basics and Initialization:** The program initializes variables of different types (int, float, double, char) and displays their values.
2. **Arithmetic Operations:** It performs basic arithmetic operations (addition, multiplication, subtraction, division) using the integer variable and displays the results.
3. **Increment and Decrement Operators:** The program demonstrates both post-increment and pre-increment, as well as post-decrement and pre-decrement operations on the integer variable.
4. **Compound Assignment Operators:** Finally, it uses compound assignment operators to modify the integer variable, displaying the results after each operation.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Variable Basics and Initialization
    int integerVar = 10;           // Integer variable
    float floatVar = 15.5f;       // Float variable
```

```

double doubleVar = 20.99;           // Double variable
char charVar = 'A';                 // Character variable

// Display initialized values
cout << "Initialized Values:" << endl;
cout << "Integer: " << integerVar << endl;
cout << "Float: " << floatVar << endl;
cout << "Double: " << doubleVar << endl;
cout << "Character: " << charVar << endl << endl;

// Arithmetic Operations with Integers
int sum = integerVar + 5;
int product = integerVar * 2;
int difference = integerVar - 3;
int quotient = integerVar / 2;

cout << "Arithmetic Operations with Integers:" << endl;
cout << "Sum: " << sum << endl;
cout << "Product: " << product << endl;
cout << "Difference: " << difference << endl;
cout << "Quotient: " << quotient << endl << endl;

// Increment and Decrement Operators
cout << "Increment and Decrement Operators:" << endl;

// Pre-increment and Post-increment
cout << "Initial Value of integerVar: " << integerVar << endl;
cout << "Post-increment: " << integerVar++ << endl; //
Displays value then increments
cout << "After Post-increment: " << integerVar << endl;

```



```

    cout << "Pre-increment: " << ++integerVar << endl;    //
Increments then displays
    cout << "After Pre-increment: " << integerVar << endl;

    // Decrement
    cout << "Post-decrement: " << integerVar-- << endl;    //
Displays value then decrements
    cout << "After Post-decrement: " << integerVar << endl;

    cout << "Pre-decrement: " << --integerVar << endl;    //
Decrements then displays
    cout << "After Pre-decrement: " << integerVar << endl << endl;

    // Using Compound Assignment Operators
integerVar += 5; // Equivalent to integerVar = integerVar + 5
cout << "After Compound Addition: " << integerVar << endl;

integerVar -= 2; // Equivalent to integerVar = integerVar - 2
cout << "After Compound Subtraction: " << integerVar << endl;

integerVar *= 3; // Equivalent to integerVar = integerVar * 3
cout << "After Compound Multiplication: " << integerVar <<
endl;

integerVar /= 2; // Equivalent to integerVar = integerVar / 2
cout << "After Compound Division: " << integerVar << endl;

return 0;
}

```

Exercise 2: Calculating the Area of Shapes Using Variables and Arithmetic Operations

```
#include <iostream>
using namespace std;

int main() {
    // Variables for shape dimensions
    float length = 10.0f;      // Length of rectangle
    float width = 5.0f;        // Width of rectangle
    float base = 6.0f;         // Base of triangle
    float height = 4.0f;       // Height of triangle
    float radius = 3.0f;       // Radius of circle

    // Calculate areas
    float rectangleArea = length * width;
    float triangleArea = 0.5f * base * height;
    float circleArea = 3.14159f * radius * radius;

    // Display the results
    cout << "Calculating Areas of Shapes:" << endl;
    cout << "Area of Rectangle: " << rectangleArea << " square
units" << endl;
    cout << "Area of Triangle: " << triangleArea << " square
units" << endl;
    cout << "Area of Circle: " << circleArea << " square units" <<
endl;

    return 0;
}
```

Practice Tasks:

1. Write a program that calculates the total cost of three items using variables. Declare variables for the price of each item, initialize them with values, and calculate the total cost. Display the total cost using `cout`. For example, use prices: `item1 = 19.99`, `item2 = 34.50`, `item3 = 12.75`.
2. Develop a program that converts a temperature from Celsius to Fahrenheit using the formula:

$$\text{Fahrenheit} = \left(\text{Celsius} \times \frac{9}{5} \right) + 32$$

Declare a variable for the Celsius temperature, initialize it with a value (e.g., 25), and calculate the equivalent Fahrenheit temperature. Display the result using `cout`.

Lab 06

Taking User Inputs in C++

Learning Objectives:

By the end of this lab, students will understand how to use the `cin` object in C++ to take user inputs. The lab will reinforce the concept of variables and introduce the importance of user interaction in programming.

Lab Exercises:

1. **Basic User Input**
2. **Calculating the Sum of Two Numbers:**
3. **Rectangle Area Calculation:** Write a program that prompts the user to input the length and width of a rectangle. Use `cin` to gather the values, calculate the area, and display the result.

Exercise 1:

Write a program that prompts the user to enter their name and age. Use `cin` to capture the inputs and display a greeting message that includes both the name and age.

Code:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    // Declare variables to hold user inputs
    string name;
    int age;

    // Prompt the user for their name
    cout << "Please enter your name: ";
    cin >> name;
```

```

// Prompt the user for their age
cout << "Please enter your age: ";
cin >> age;

// Display a greeting message
cout << "Hello, " << name << "! You are " << age << " years
old." << endl;

return 0;
}

```

Exercise 2:

Create a program that prompts the user to enter two integers. Use cin to get the values and calculate their sum. Display the result to the user.

```

#include <iostream>
using namespace std;

int main() {
    // Declare variables to hold user inputs
    int num1, num2, sum;

    // Prompt the user for the first integer
    cout << "Enter the first integer: ";
    cin >> num1;

    // Prompt the user for the second integer
    cout << "Enter the second integer: ";
    cin >> num2;

    // Calculate the sum
    sum = num1 + num2;
}

```

```

    // Display the result
    cout << "The sum of " << num1 << " and " << num2 << " is: " <<
sum << endl;

    return 0;
}

```

Exercise 3:

Write a program that prompts the user to input the length and width of a rectangle. Use cin to gather the values, calculate the area, and display the result.

Code:

```

#include <iostream>
using namespace std;

int main() {
    // Declare variables to hold length, width, and area
    float length, width, area;

    // Prompt the user for the length of the rectangle
    cout << "Enter the length of the rectangle: ";
    cin >> length;

    // Prompt the user for the width of the rectangle
    cout << "Enter the width of the rectangle: ";
    cin >> width;

    // Calculate the area
    area = length * width;

    // Display the result
    cout << "The area of the rectangle is: " << area << endl;
}

```

```
    return 0;
}
```

Practice Tasks:

1. Write a program that calculates the average of three integers and three floating-point numbers. Display the results with appropriate messages, and ensure to use correct formatting for the floating-point output.
2. Write a program that prompts the user to enter the length and width of a rectangle. Use ``cin`` to gather the values, calculate the perimeter using the formula ``Perimeter = 2 * (length + width)``, and display the result.
3. Converting Celsius to Fahrenheit: Create a program that asks the user to input a temperature in Celsius. Use ``cin`` to capture the input, convert it to Fahrenheit using the formula ``Fahrenheit = (Celsius * 9/5) + 32``, and display the converted temperature.
4. Calculating the Average of Three Numbers: Develop a program that prompts the user to input three numbers. Use ``cin`` to collect the values, calculate their average using the formula ``Average = (num1 + num2 + num3) / 3``, and display the result.
5. Calculating the Hypotenuse of a Right Triangle: Write a program that asks the user to enter the lengths of the two legs of a right triangle. Use ``cin`` to gather the lengths, calculate the hypotenuse using the Pythagorean theorem ``Hypotenuse = sqrt(leg12 + leg22)``, and display the result. (You may need to include the ``<cmath>`` library for the square root function.)

Lab 07

Conditional Statements in C++

Learning Objectives:

In this lab, students will explore various conditional statements in C++, which allow programs to make decisions based on specific conditions. By the end of this lab, students will understand how to use if, if/else, if-else if, and nested if statements to control the flow of their programs.

- Use if statements to execute code based on a single condition.
- Implement if/else statements to provide alternative code paths.
- Create if-else if structures to handle multiple conditions.
- Utilize nested if statements to create complex decision-making logic.

Lab Exercises:

1. **Sentence 1**
2. **Sentence 2**
3. **Sentence 3**

Exercise 1: Simple *if* Statement

Write a program that prompts the user to enter a number and checks if it is positive. If it is, display a message stating that the number is positive.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Variable to store the user input
    int number;

    // Prompt the user to enter a number
    cout << "Enter a number: ";
    cin >> number;
```



```

// Check if the number is positive
if (number > 0) {
    cout << "The number is positive." << endl;
}

return 0;
}

```

Exercise 2: if/else Statement

Create a program that asks the user to input their age. Use an if/else statement to check if they are old enough to vote (age 18 or older). Display an appropriate message based on their input.

Code:

```

#include <iostream>
using namespace std;

int main() {
    // Variable to store the user's age
    int age;

    // Prompt the user to enter their age
    cout << "Enter your age: ";
    cin >> age;

    // Check if the user is old enough to vote
    if (age >= 18) {
        cout << "You are old enough to vote." << endl;
    } else {
        cout << "You are not old enough to vote." << endl;
    }
}

```

```
    return 0;
}
```

Exercise 3: if-else if Structure

Develop a program that prompts the user to enter a grade (0-100). Use if-else if statements to determine the corresponding letter grade (A, B, C, D, F) and display it.

```
#include <iostream>
using namespace std;

int main() {
    // Variable to store the user's grade
    int grade;

    // Prompt the user to enter a grade
    cout << "Enter your grade (0-100): ";
    cin >> grade;

    // Determine the letter grade using if-else if statements
    if (grade >= 90 && grade <= 100) {
        cout << "Your letter grade is: A" << endl;
    } else if (grade >= 80) {
        cout << "Your letter grade is: B" << endl;
    } else if (grade >= 70) {
        cout << "Your letter grade is: C" << endl;
    } else if (grade >= 60) {
        cout << "Your letter grade is: D" << endl;
    } else if (grade >= 0) {
        cout << "Your letter grade is: F" << endl;
    } else {
        cout << "Invalid grade. Please enter a grade between 0 and
100." << endl;
    }
}
```

```

    }

    return 0;
}

```

Exercise 4: Nested if Statement:

Write a program that asks the user for their score (0-100) and checks if they passed or failed (passing score is 60 or above). If they passed, use a nested if statement to check if they achieved a distinction (score 85 or above) and display an appropriate message.

Code:

```

#include <iostream>
using namespace std;

int main() {
    // Variable to store the user's score
    int score;

    // Prompt the user to enter their score
    cout << "Enter your score (0-100): ";
    cin >> score;

    // Check if the score is valid
    if (score < 0 || score > 100) {
        cout << "Invalid score. Please enter a score between 0 and 100." << endl;
    } else {
        // Check if the user passed or failed
        if (score >= 60) {
            cout << "You passed!" << endl;

            // Nested if statement to check for distinction
            if (score >= 85) {

```

```
        cout << "Congratulations! You achieved a  
distinction!" << endl;  
    }  
    } else {  
        cout << "You failed. Better luck next time." << endl;  
    }  
}  
  
return 0;  
}
```

Lab 08

Logical Operators

Learning Objectives:

In this lab, students will explore logical operators in C++, understanding how to combine multiple conditions in decision-making statements. By the end of the lab, students will be able to use logical operators such as **AND** (&&), **OR** (||), and **NOT** (!) to create complex conditional expressions in their programs.

Lab Exercises:

1. **Using AND Operator &&**
2. **Using OR Operator ||**
3. **Using NOT Operator !**
4. **Combining Logical Operators**

The AND operator (&&) returns true only if both operands are true; if either operand is false, the result is false. In contrast, the OR operator (||) yields true if at least one operand is true, returning false only when both are false. The NOT operator (!) negates the truth value of a boolean expression, so it returns false if the operand is true and true if the operand is false. These logical operators are fundamental in controlling the flow of programs by allowing complex conditions to be evaluated.

Exercise 1: AND Operator

Write a program that prompts the user to enter two numbers and checks if both numbers are positive. Display a message indicating whether both numbers are positive.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Declare variables for user input
    int number1, number2;

    // Prompt the user to enter two numbers
```

```

cout << "Enter the first number: ";
cin >> number1;
cout << "Enter the second number: ";
cin >> number2;

// Check if both numbers are positive
if (number1 > 0 && number2 > 0) {
    cout << "Both numbers are positive." << endl;
} else {
    cout << "At least one of the numbers is not positive." <<
endl;
}

return 0;
}

```

Exercise 2: OR Operator

Create a program that asks the user for their age and checks if they are eligible for a senior citizen discount (age 65 or older) or a student discount (age 18 or younger). Display an appropriate message based on their eligibility.

Code:

```

#include <iostream>
using namespace std;

int main() {
    // Declare a variable for user input
    int age;

    // Prompt the user to enter their age
    cout << "Enter your age: ";
    cin >> age;
}

```

```

    // Check for eligibility for discounts
    if (age >= 65) {
        cout << "You are eligible for a senior citizen discount."
<< endl;
    } else if (age <= 18) {
        cout << "You are eligible for a student discount." <<
endl;
    } else {
        cout << "You are not eligible for any discounts." << endl;
    }

    return 0;
}

```

Exercise 3: NOT Operator

Develop a program that prompts the user to enter a Boolean value (0 for false, 1 for true). Use the NOT operator to display the opposite value of the user's input.

Code:

```

#include <iostream>
using namespace std;

int main() {
    // Declare a variable to hold user input
    int userInput;

    // Prompt the user to enter a boolean value (0 or 1)
    cout << "Enter a boolean value (0 for false, 1 for true): ";
    cin >> userInput;

    // Check if the input is valid
    if (userInput != 0 && userInput != 1) {

```

```

        cout << "Invalid input. Please enter 0 or 1." << endl;
    } else {
        // Use the NOT operator to find the opposite value
        bool oppositeValue = !userInput;

        // Display the opposite value
        cout << "The opposite value is: " << oppositeValue <<
endl;
    }

    return 0;
}

```

Exercise 3: Combining Logical Operators

Write a program that prompts the user to enter their age, score (0-100), and membership status (1 for Yes, 0 for No). Use logical operators to evaluate the following conditions: check if the user is eligible for a driver's license (age 18 or older and score 60 or above) and whether they qualify for a senior citizen discount (age 65 or older and not a member). Display appropriate messages based on these evaluations.

Code:

```

#include <iostream>
using namespace std;

int main() {
    int age;
    float score;
    bool isMember;

    // Prompting user for input
    cout << "Enter your age: ";
    cin >> age;
    cout << "Enter your score (0-100): ";

```



```

cin >> score;
cout << "Are you a member? (1 for Yes, 0 for No): ";
cin >> isMember;

// Check eligibility for driver's license
if (age >= 18 && score >= 60) {
    cout << "You are eligible for a driver's license." <<
endl;
} else {
    cout << "You are not eligible for a driver's license." <<
endl;
}

// Check eligibility for senior citizen discount
if (age >= 65 && !isMember) {
    cout << "You qualify for a senior citizen discount!" <<
endl;
} else if (isMember) {
    cout << "As a member, you already receive discounts!" <<
endl;
} else {
    cout << "No discounts available." << endl;
}

return 0;
}

```

Practice Task:

1. Eligibility Checker for Discounts and Licenses:

Write a program that prompts the user to enter their age, score (0-100), and whether they are a student (1 for Yes, 0 for No). Use logical operators to check the following conditions:

- Determine if the user is eligible for a student discount (age 18 or younger).
- Check if they qualify for a driver's license (age 18 or older and score 70 or above).
- Display appropriate messages based on these evaluations, indicating which discounts or licenses the user is eligible for.

2. Game Score Validator:

Create a program that asks the user to input their game score (0-100) and their membership status (1 for Yes, 0 for No). Use nested if statements and logical operators to evaluate the following:

- If the score is 85 or higher, display a message that they qualify for a "Champion" status.
- If the score is below 85 but above 60 and they are a member, display a message that they qualify for a "Premium Player" status.
- If the score is below 60, display a message indicating they need to improve.
- Ensure to handle cases where the user inputs invalid scores (e.g., less than 0 or greater than 100) by displaying an error message.

Lab 09

Logical Operators

Learning Objectives:

By the end of this lab, students will understand how to use the switch statement in C++ to control the flow of a program based on the value of a variable.

Lab Exercises:

1. **Basic Switch Statement**
2. **Grade Assessment**
3. **Simple Calculator**

Exercise 1: Basic Switch Statement

Write a program that prompts the user to enter a number from 1 to 5 and displays the corresponding day of the week (1 for Monday, 2 for Tuesday, etc.). If the user enters a number outside this range, display an error message.

Code:

```
#include <iostream>
using namespace std;

int main() {
    int day;

    // Prompt the user for input
    cout << "Enter a number from 1 to 5: ";
    cin >> day;

    // Switch statement to determine the day of the week
    switch (day) {
        case 1:
            cout << "Monday" << endl;
            break;
```

```

        case 2:
            cout << "Tuesday" << endl;
            break;
        case 3:
            cout << "Wednesday" << endl;
            break;
        case 4:
            cout << "Thursday" << endl;
            break;
        case 5:
            cout << "Friday" << endl;
            break;
        default:
            cout << "Error: Please enter a number between 1 and
5." << endl;
            break;
    }

    return 0;
}

```

Exercise 2: Grade Assessment

Create a program that asks the user to input a letter grade (A, B, C, D, F) and uses a switch statement to display the corresponding message about the performance (e.g., "Excellent" for A, "Good" for B, etc.). Include a default case for invalid inputs.

Code:

```

#include <iostream>
using namespace std;

int main() {
    char grade;

```

```

// Prompt the user for input
cout << "Enter a letter grade (A, B, C, D, F): ";
cin >> grade;

// Switch statement to determine performance message
switch (grade) {
    case 'A':
    case 'a':
        cout << "Excellent!" << endl;
        break;
    case 'B':
    case 'b':
        cout << "Good!" << endl;
        break;
    case 'C':
    case 'c':
        cout << "Average!" << endl;
        break;
    case 'D':
    case 'd':
        cout << "Below Average!" << endl;
        break;
    case 'F':
    case 'f':
        cout << "Fail!" << endl;
        break;
    default:
        cout << "Error: Invalid grade input. Please enter A,
B, C, D, or F." << endl;
        break;
}

```

```
    return 0;
}
```

Exercise 3: Simple Calculator

Develop a calculator program that prompts the user to enter two numbers and an operator (+, -, *, /). Use a switch statement to perform the corresponding operation and display the result. Handle division by zero gracefully.

Code:

```
#include <iostream>
using namespace std;

int main() {
    double num1, num2;
    char op;

    // Prompt the user for input
    cout << "Enter first number: ";
    cin >> num1;
    cout << "Enter second number: ";
    cin >> num2;
    cout << "Enter operator (+, -, *, /): ";
    cin >> op;

    double result;

    // Switch statement to perform the operation
    switch (op) {
        case '+':
            result = num1 + num2;
```

```

        cout << "Result: " << num1 << " + " << num2 << " = "
<< result << endl;
        break;
    case '-':
        result = num1 - num2;
        cout << "Result: " << num1 << " - " << num2 << " = "
<< result << endl;
        break;
    case '*':
        result = num1 * num2;
        cout << "Result: " << num1 << " * " << num2 << " = "
<< result << endl;
        break;
    case '/':
        if (num2 != 0) {
            result = num1 / num2;
            cout << "Result: " << num1 << " / " << num2 << " = "
" << result << endl;
        } else {
            cout << "Error: Division by zero is not allowed."
<< endl;
        }
        break;
    default:
        cout << "Error: Invalid operator. Please enter +, -,
*, or /." << endl;
        break;
    }

    return 0;
}

```


Practice Tasks:

1. **Day of the Week Selector:** Implement a program using a switch statement that allows the user to input a number corresponding to a day of the week and outputs the day's name.
2. **Pizza Order Menu:** Create a program that presents a menu for different pizza sizes (Small, Medium, Large, Extra Large). Use a switch statement to capture the user's choice and display the price for the selected size. If the input is invalid, provide an error message.
3. **Menu Selection:** Write a program that presents a menu to the user with the following options: 1 for adding two numbers, 2 for finding the maximum of two numbers, and 3 for exiting the program. Use a switch statement to execute the selected option, prompting the user for input where necessary, and display appropriate messages based on the chosen task.

Lab 10

While Loop in C++

Learning Objectives:

In this lab, students will understand the concept of loops in programming, specifically the while loop. By the end of this lab, students will be able to write programs that utilize while loops to perform repetitive tasks until a specified condition is no longer true. They will learn to control the flow of execution in their programs through iteration and understand how to avoid infinite loops.

Lab Exercises:

1. **Basic While Loop Program**
2. **Comprehensive While Loop Example**

What is a While Loop?

A while loop is a control flow statement that repeatedly executes a block of code as long as a given condition evaluates to true. It checks the condition before each iteration, making it suitable for situations where the number of iterations is not known beforehand. The loop will terminate once the condition becomes false.

Exercise 1: Basic While Loop Program

```
#include <iostream>
using namespace std;

int main() {
    int count = 1;
    while (count <= 10) {
        cout << count << endl;
        count++;
    }
    return 0;
}
```

Exercise 2: Comprehensive While Loop

This example exercise covers the basics of while loops, including initialization, condition checking, iteration, and termination. This program will prompt the user to enter positive integers, calculate their sum, and count how many numbers were entered. It will also demonstrate how to gracefully exit the loop when the user inputs a negative number.

Code:

```
#include <iostream>
using namespace std;

int main() {
    int number;           // Variable to store user input
    int sum = 0;          // Variable to accumulate the sum
    int count = 0;        // Variable to count the number of inputs

    cout << "Enter positive integers to sum them up (enter a
negative number to stop):" << endl;

    // Initialize the loop
    while (true) {
        cout << "Enter a number: ";
        cin >> number; // Capture user input

        // Check for exit condition
        if (number < 0) {
            break; // Exit the loop if a negative number is
entered
        }

        // Update sum and count
        sum += number; // Add the number to the sum
        count++;       // Increment the count of numbers entered
    }
}
```

```
// Display the results
cout << "Total numbers entered: " << count << endl;
cout << "Sum of entered numbers: " << sum << endl;

return 0;
}
```

Practice Tasks:

1. **Bank ATM Withdrawal Simulation:** Write a program that simulates an ATM withdrawal process. Prompt the user to enter their current balance and then repeatedly ask how much they want to withdraw. After each withdrawal, deduct the amount from their balance and display the remaining balance. The loop should continue until the user decides to stop or until they have insufficient funds to withdraw the requested amount. Include appropriate messages for invalid withdrawals (e.g., more than the balance).
2. **Guess the Number Game:** Write a program that has a hard code integer number as a secret number and asks the user to guess it. Use a while loop to repeatedly prompt the user for their guess until they either guess the number correctly or decide to quit. After each guess, provide feedback indicating whether the guess is too high, too low, or correct. If the user chooses to quit, display the correct number. This task emphasizes user interaction and conditional logic within a loop.
3. **Reversing Integer Number:** Create a program that prompts the user to enter a positive integer. Use a while loop to reverse the digits of the integer. Display the reversed integer once the process is complete. Include a conditional statement to handle cases where the user enters a non-positive integer, prompting them to enter a valid number.

Lab 11

Do While Loop

Learning Objectives:

- Understand the structure and purpose of the do-while loop.
- Implement do-while loops to ensure code execution at least once.

Lab Exercises:

1. **Basic Do-While Program**
2. **User Input Validation:**
3. **Menu Selection**

What is a Do-While Loop?

A do-while loop is a type of loop that executes a block of code at least once before checking a condition. The loop continues as long as the specified condition evaluates to true. The syntax typically involves the `do` keyword followed by a block of code and the `while` keyword with a condition.

Exercise 1: Basic Do-While Program

Code:

```
#include <iostream>
using namespace std;

int main() {
    int number;

    // User input validation
    do {
        cout << "Enter a positive integer: ";
        cin >> number;
    } while (number <= 0); // Repeat until a valid positive
integer is entered
```

```
    cout << "You entered: " << number << endl;

    return 0;
}
```

Exercise 2:

User Input Validation: Write a program that prompts the user to enter a positive integer. Use a do-while loop to repeatedly ask for input until a valid positive integer is entered. Display the valid input after the loop concludes.

Code:

```
#include <iostream>
using namespace std;

int main() {
    int number;

    // User input validation
    do {
        cout << "Enter a positive integer: ";
        cin >> number;

        if (number <= 0) {
            cout << "Invalid input. Please enter a positive
integer." << endl;
        }
    } while (number <= 0); // Repeat until a valid positive
integer is entered

    cout << "You entered: " << number << endl;
```

```
    return 0;
}
```

Exercise 3:

Menu Selection: Create a menu-driven program that presents options to the user (e.g., 1 for adding two numbers, 2 for subtracting two numbers, 3 for exiting). Use a do-while loop to allow the user to perform multiple operations until they choose to exit.

Code:

```
#include <iostream>
using namespace std;

int main() {
    int choice;
    double num1, num2, result;

    do {
        // Display the menu
        cout << "Menu Selection:" << endl;
        cout << "1. Add two numbers" << endl;
        cout << "2. Subtract two numbers" << endl;
        cout << "3. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        // Perform the chosen operation
        switch (choice) {
            case 1:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                result = num1 + num2;
                cout << "Result: " << result << endl;
            case 2:
                cout << "Enter two numbers: ";
                cin >> num1 >> num2;
                result = num1 - num2;
                cout << "Result: " << result << endl;
            case 3:
                break;
        }
    } while (choice != 3);
}
```

```

        break;
    case 2:
        cout << "Enter two numbers: ";
        cin >> num1 >> num2;
        result = num1 - num2;
        cout << "Result: " << result << endl;
        break;
    case 3:
        cout << "Exiting the program." << endl;
        break;
    default:
        cout << "Invalid choice. Please try again." <<
endl;
    }
    cout << endl; // Blank line for better readability
} while (choice != 3); // Continue until the user chooses to
exit

return 0;
}

```

Practice Tasks:

1. **Guess the Number:** Write a program that generates a random number between 1 and 100. Prompt the user to guess the number. Use a do-while loop to allow the user to continue guessing until they correctly identify the number. Provide feedback after each guess, indicating whether the guess was too high or too low. Once they guess correctly, display a congratulatory message.
2. **Menu for Favorite Foods:** Create a program that allows users to input their favorite foods. Use a do-while loop to repeatedly prompt the user to enter a food item. After each entry, ask if they want to add another food item. When they choose to stop, display the list of all favorite foods they entered.

Lab 12

For Loop in C++

Learning Objectives:

By the end of this lab, students will be able to:

- Understand the syntax and structure of for loops in C++.
- Implement for loops to solve problems that require iteration.
- Apply for loops to real-world scenarios, such as summing values, generating sequences, or processing collections of data.

Lab Exercises:

1. **Basic For Loop Syntax**
2. **Summing Even Numbers**
3. **Multiplication Table**

A **for loop** is a control flow statement in programming that allows code to be executed repeatedly based on a specified condition. It is particularly useful for situations where the number of iterations is known before entering the loop.

Exercise 1: Basic for Loop Syntax

Create a program that uses a `for` loop to print the numbers from 1 to 10, each on a new line.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Using a for loop to print numbers from 1 to 10
    for (int i = 1; i <= 10; i++) {
        cout << i << endl; // Print the current number
    }
    return 0;
}
```

Exercise 2: Summing Even Numbers

Write a program that uses a for loop to calculate the sum of all even numbers from 1 to 100. Display the result after the loop.

Code:

```
#include <iostream>
using namespace std;

int main() {
    int sum = 0; // Variable to hold the sum of even numbers

    // For loop to iterate from 1 to 100
    for (int i = 1; i <= 100; i++) {
        // Check if the number is even
        if (i % 2 == 0) {
            sum += i; // Add the even number to the sum
        }
    }

    // Display the result
    cout << "The sum of all even numbers from 1 to 100 is: " <<
sum << endl;

    return 0;
}
```

Exercise 3: Multiplication Table

Develop a program that generates a multiplication table for a given number using a for loop. Prompt the user to enter a number and display its multiplication table from 1 to 10.

Code:

```
#include <iostream>
using namespace std;

int main() {
    int number;

    // Prompt the user to enter a number
    cout << "Enter a number to get its multiplication table: ";
    cin >> number;

    // Generate the multiplication table using a for loop
    cout << "Multiplication Table for " << number << ":" << endl;
    for (int i = 1; i <= 10; i++) {
        cout << number << " x " << i << " = " << number * i <<
endl; // Display the multiplication
    }

    return 0;
}
```

Practice Tasks:

1. **Factorial Calculation:** Write a program that calculates the factorial of a given positive integer using a for loop. Prompt the user to enter a number and display the factorial.
2. **Sum of Odd Numbers:** Develop a program that calculates the sum of all odd numbers from 1 to a specified number entered by the user. Use a for loop to iterate through the numbers and display the total sum.
3. **Week Temperature Avg Calculator:** Create a program that simulates the daily temperature readings for a week. Prompt the user to enter the number of days (up to 7) and for each day, ask for the temperature recorded. Using a for loop, calculate the average temperature for the week, identify the highest and lowest temperatures, and count how many days had temperatures above a specified threshold (e.g., 25°C). Finally, display the average temperature, the highest and lowest readings, and the count of days above the threshold, allowing users to analyze the week's weather patterns effectively.

Lab 13

Nested For Loops in C++

Learning Objectives:

By the end of this lab, students will be able to understand and implement nested loops in C++. They will learn how to utilize nested loops for various tasks, including creating patterns and solving real-world problems that require iteration through multiple levels.

Lab Exercises:

1. **Basic Nested for Loop: Pair Generation**
2. **Number Pattern**

Exercise 1: Pair Generation

Write a program that prompts the user to enter a positive integer n . The program should generate and display all possible pairs of integers from 1 to n in the format (i, j) , where i and j are both integers in the range 1 to n .

The output should be clearly formatted with a header and a footer, making it easy to read.

Code:

```
#include <iostream>
using namespace std;

int main() {
    int n;

    cout << "Enter the number of pairs you want to create: ";
    cin >> n;

    cout << "\nGenerated Pairs:\n";
    cout << "-----\n";

    // Generate and display pairs
    for (int i = 1; i <= n; i++) {
```

```
        for (int j = 1; j <= n; j++) {  
            cout << "(" << i << ", " << j << ")" << endl;  
        }  
    }  
  
    cout << "-----\n";  
    return 0;  
}
```

Exercise 2: Number Pattern

Write a program that uses nested loops to display a pattern of numbers. The output should be a triangle of numbers, where each row contains the row number repeated.

For example, for 3 rows, the output should look like this:

```
1
2 2
3 3 3
```

Code:

```
#include <iostream>
using namespace std;

int main() {
    int n;

    cout << "Enter the number of rows: ";
    cin >> n;

    for (int i = 1; i <= n; i++) {           // Outer loop for
rows
        for (int j = 1; j <= i; j++) {       // Inner loop for
columns
            cout << i << " ";               // Print the row
number
        }
        cout << endl;                       // Move to the next
line after each row
    }

    return 0;
}
```

Practical Task:

1. **Number Pyramid:** Create a program that displays a number pyramid. The user should enter the number of rows, and the program should print a pyramid of numbers aligned to the center. For example, if the user inputs 4, the output should look like this:

```
1
2 2
3 3 3
4 4 4 4
```

2. **Pair Number Generation:** Write a program that generates pairs of numbers based on user input. The program should prompt the user to enter a starting number, n , and then create a specified number of pairs where the first number in each pair increases in ascending order starting from n , while the second number decreases in descending order starting from $n + k - 1$, where k is the total number of pairs to be generated.
3. **Diamond Shape:** Develop a program that displays a diamond shape made of asterisks. For example, for 5 rows, the output should look like this:

```
  *
 ***
*****
*****
*****
*****
*****
  *
  *
```

4. **Inverted Right Triangle:** Write a program that uses nested loops to display an inverted right triangle of asterisks. For example, for 5 rows, the output should look like this:

```
*****
****
***
**
*
```

Lab 14

User Defined Functions in C++

Learning Objectives:

By the end of this lab, students will be able to:

- Understand the syntax and structure of user-defined functions in C++.
- Create functions that perform specific tasks and return values.
- Use parameters to pass information into functions and understand the concept of function overloading.

Lab Exercises:

1. **Basic Function: Display Greeting Message with a void function**
2. **Void Function with Parameters**
3. **Function returning values**
4. **Function with Multiple Parameters**
5. **Function Overriding**

Exercise 1: Display Greeting Message with a void function

Write a C++ program that defines a user-defined void function named `display_decorated_message`. This function should print a decorative welcome message to the console, enclosed in a border made of asterisks. In the `main` function, call this void function three times to display the message multiple times on the screen. Ensure that the output is well-formatted and visually appealing.

Expected Outcome:

```
*****  
  
|   Welcome to C++ Programming   |  
  
*****
```

Code:

```
#include <iostream>  
using namespace std;
```

```
// Function Definition
void display_decorated_message() {
    cout << "*****" << endl;
    cout << "|    Welcome to C++ Programming    |" << endl;
    cout << "*****" << endl;
}

int main() {
    // Call the function to display the message multiple times
    display_decorated_message(); // Function Call 1
    display_decorated_message(); // Function Call 2
    display_decorated_message(); // Function Call 3

    return 0;
}
```

Exercise 2: Void Function with Parameters

Write a C++ program that defines a user-defined void function named `display_decorated_message`, which takes a single string parameter. This function should print a decorative message to the console, with the provided string centered within a border of asterisks. In the main function, call this void function three times, passing different welcome messages as arguments to display them in the decorative format. Ensure that the output is clear, well-structured, and visually appealing.

Code:

```
#include <iostream>
using namespace std;

// Function Definition with a single parameter
void display_decorated_message(string message) {
    cout << "*****" << endl;
    cout << "|    " << message << "    |" << endl;
}
```

```

        cout << "*****" << endl;
    }

int main() {
    // Function Call 1
    display_decorated_message("Welcome to Programming!");

    // Function Call 2
    display_decorated_message("Welcome to AI!");

    // Function Call 3
    display_decorated_message("Welcome to IMSciences");

    return 0;
}

```

Exercise 3: Function returning values

Write a C++ program that defines a function to calculate the square of a number. The function should take an integer as an argument and return the square of that number. In the main function, prompt the user to enter a number, call the square function, and display the result.

Code:

```

#include <iostream>
using namespace std;

// Function Definition that returns the square of a number
int calculate_square(int number) {
    return number * number; // Calculate and return the square
}

int main() {
    int number;

```

```

// Prompt the user for input
cout << "Enter a number to find its square: ";
cin >> number;

// Call the function and store the result
int square = calculate_square(number);

// Display the result
cout << "The square of " << number << " is " << square << "."
<< endl;

return 0;
}

```

Exercise 4: Function with Multiple Parameters

```

#include <iostream>
using namespace std;

// Function to calculate area of a rectangle
float calculate_area(float length, float width) {
    return length * width; // Calculate and return the area
}

int main() {
    float length, width;

    // Prompt user for length and width
    cout << "Enter the length of the rectangle: ";
    cin >> length;
    cout << "Enter the width of the rectangle: ";
    cin >> width;

    // Call the function to calculate area

```

```

float area = calculate_area(length, width);

// Display the result
cout << "The area of the rectangle is: " << area << endl;

return 0;
}

```

Exercise 5: Function Overloading

Code:

```

#include <iostream>
using namespace std;

int find_sum(int a, int b) {
    return a + b;
}

int find_sum(int a, int b, int c) {
    return a + b + c;
}

int find_sum(int a, int b, int c, int d) {
    return a + b + c + d;
}

int main() {
    cout << "Sum of 5 + 6 is      : " << find_sum(5, 6) << endl;
    cout << "Sum of 5 + 6 + 1 is   : " << find_sum(5, 6, 1) << endl;
    cout << "Sum of 5 + 6 + 22 is  : " << find_sum(5, 6, 22) << endl;

    return 0;
}

```

Practice Task:

1. Develop a C++ program that includes a user-defined function named `display_table`. This function should calculate and display the multiplication table for a specified integer value entered by the user. Ensure that the output is clearly formatted and easily readable.
2. Construct a C++ program that incorporates a user-defined function to calculate the age of a user based on the birth year they provide. The function should accept the birth year as an input, compute the age, and return the result. Include appropriate input validation to ensure the birth year is reasonable.
3. Write a C++ function that collects three pieces of information from the user: their name, age, and gender. The function should accept these parameters with different data types and display the information in a visually appealing format. This function should not return any values. Ensure that the presentation is clear and well-structured.
4. Create a C++ program that includes overloaded functions to calculate the area of different shapes. Implement the following functions:

- **Circle Area:**

- `double calculate_area(double radius)`

This function should calculate and return the area of a circle using the formula:

$$\text{Area} = \pi r^2$$

- **Rectangle Area:**

- `double calculate_area(double length, double width)`

This function should calculate and return the area of a rectangle using the formula: *length* × *width*

- **Triangle Area:**

- `double calculate_area(double base, double height)`

This function should calculate and return the area of a triangle using the formula:

$$\frac{1}{2} \times \text{base} \times \text{height}.$$

Lab 15

Arrays in C++

Learning Objectives:

- Understand the concept and structure of arrays in C++.
- Perform various operations on arrays, including initialization, manipulation, and searching.
- Apply array operations to solve real-world problems.

Lab Exercises:

1. **Student Grades Management – Basic**
2. **Students' Scores Management**
3. **Array Operations**

Exercise 1: Student Grades Management

Write a C++ program to manage a student's scores in five subjects. The program should:

- Store the subject names in an array.
- Prompt the user to enter scores for each subject.
- Calculate and display the average score, the highest score, and the lowest score along with the corresponding subjects.

Code:

```
#include <iostream>
using namespace std;

int main() {
    const int NUM_SUBJECTS = 5; // Number of subjects
    string subjects[NUM_SUBJECTS] = {"Math", "Science", "English",
    "History", "Art"}; // Array of subjects
    int scores[NUM_SUBJECTS]; // Array to store scores
    int sum = 0; // Variable to hold the sum of scores
```

```

int highest, lowest;

// Prompt user to input scores for each subject
for (int i = 0; i < NUM_SUBJECTS; i++) {
    cout << "Enter score for " << subjects[i] << ": ";
    cin >> scores[i];
    sum += scores[i]; // Update the sum
}

// Calculate average score
double average = static_cast<double>(sum) / NUM_SUBJECTS;

// Initialize highest and lowest
highest = scores[0];
lowest = scores[0];

// Determine highest and lowest scores
for (int i = 1; i < NUM_SUBJECTS; i++) {
    if (scores[i] > highest) {
        highest = scores[i];
    }
    if (scores[i] < lowest) {
        lowest = scores[i];
    }
}

cout << "\nAverage score: " << average << endl;
cout << "Highest score: " << highest << endl;
cout << "Lowest score: " << lowest << endl;

return 0;
}

```


Exercise 2: Student Score Management

Write a C++ program that manages and analyzes scores of students in a class using arrays.

Requirements:

1. Array Declaration:

- Use an array to store the scores of up to 10 students.

2. Input Scores:

- Prompt the user to enter the scores for each student.

3. Calculate Statistics:

- Calculate and display the following:
 - ✓ The average score of the class.
 - ✓ The highest score and the corresponding student number.
 - ✓ The lowest score and the corresponding student number.

4. Display Scores:

- After calculating the statistics, display all student scores along with their corresponding numbers.

Code:

```
#include <iostream>
using namespace std;

const int MAX_STUDENTS = 10; // Maximum number of students

int main() {
    int scores[MAX_STUDENTS]; // Array to store student scores
    int numStudents;
    double sum = 0.0; // Variable to hold the sum of scores
    int highest, lowest;
    int highestIndex, lowestIndex;
```

```

// Prompt user for the number of students
cout << "Enter the number of students (up to " << MAX_STUDENTS
<< "): ";
cin >> numStudents;

// Input validation
if (numStudents < 1 || numStudents > MAX_STUDENTS) {
    cout << "Invalid number of students!" << endl;
    return 1;
}

// Input scores for each student
for (int i = 0; i < numStudents; i++) {
    cout << "Enter score for student " << (i + 1) << ": ";
    cin >> scores[i];
    sum += scores[i]; // Update the sum
}

// Calculate average score
double average = sum / numStudents;

// Initialize highest and lowest scores
highest = scores[0];
lowest = scores[0];
highestIndex = 0;
lowestIndex = 0;

// Find highest and lowest scores
for (int i = 1; i < numStudents; i++) {
    if (scores[i] > highest) {
        highest = scores[i];
        highestIndex = i;
    }
}

```

```

    }
    if (scores[i] < lowest) {
        lowest = scores[i];
        lowestIndex = i;
    }
}

// Display results
cout << "\nAverage score: " << average << endl;
cout << "Highest score: " << highest << " (Student " <<
(highestIndex + 1) << ")" << endl;
cout << "Lowest score: " << lowest << " (Student " <<
(lowestIndex + 1) << ")" << endl;

// Display all student scores
cout << "\nStudent Scores:\n";
for (int i = 0; i < numStudents; i++) {
    cout << "Student " << (i + 1) << ": " << scores[i] <<
endl;
}

return 0;
}

```

Exercise 3: Searching an Array

Write a C++ program to look up a student's total marks using parallel arrays.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Hard-coded arrays
    int studentIDs[10] = {101, 102, 103, 104, 105, 106, 107, 108,
109, 110}; // Student IDs
    int totalMarks[10] = {85, 90, 78, 88, 92, 75, 84, 95, 80, 88};
// Corresponding total marks

    int inputID;
    bool found = false;

    // Prompt user for Student ID
    cout << "Enter Student ID: ";
    cin >> inputID;

    // Search for the Student ID
    for (int i = 0; i < 10; i++) {
        if (studentIDs[i] == inputID) {
            cout << "Total Marks for Student ID " << inputID << ":
" << totalMarks[i] << endl;
            found = true;
            break;
        }
    }

    // If ID is not found
    if (!found) {
```

```
        cout << "Student ID " << inputID << " not found." << endl;
    }

    return 0;
}
```

Practice Tasks:

1. Write a C++ program that stores daily temperatures for a week in Celsius and converts them to Fahrenheit.

Requirements:

- Use an array to store temperatures for 7 days.
- Prompt the user to enter temperatures for each day.
- Convert each temperature to Fahrenheit and display the results.

Example Output:

```
Enter temperature for day 1 (Celsius): 20
Enter temperature for day 2 (Celsius): 22
...
Temperatures in Fahrenheit:
Day 1: 68
Day 2: 71.6
...
```

2. Write a C++ program that searches for a specific number in a single array and returns its position.
3. Write a C++ program that counts how many times a specific number appears in a single array.
4. Write a C++ program that finds the minimum and maximum values in a single array of integers.
5. Write a C++ program that reverses the elements of a single array and displays the reversed array.

Lab 16

2D Arrays in C++

Learning Objectives:

The objective of this lab is to understand the basic concepts of 2D arrays in C++, including declaration, initialization, and accessing elements. Students will practice simple operations using 2D arrays.

Lab Exercises:

1. Write a C++ program that adds two 2D arrays (matrices) of size 3x3.
2. Write a C++ program that finds the maximum element in a 2D array.
3. Write a C++ program that computes the transpose of a 2D array.

Exercise 1:

Write a C++ program that adds two 2D arrays (matrices) of size 3x3.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Declare and initialize two 3x3 matrices
    int matrixA[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    int matrixB[3][3] = {
        {9, 8, 7},
        {6, 5, 4},
        {3, 2, 1}
    };
}
```

```

};

// Declare a matrix to hold the result
int result[3][3];

// Add the two matrices
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        result[i][j] = matrixA[i][j] + matrixB[i][j];
    }
}

// Display the matrices
cout << "Matrix A:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        cout << matrixA[i][j] << " ";
    }
    cout << endl;
}

cout << "\nMatrix B:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        cout << matrixB[i][j] << " ";
    }
    cout << endl;
}

cout << "\nResultant Matrix (A + B):\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {

```

```

        cout << result[i][j] << " ";
    }
    cout << endl;
}

return 0;
}

```

Exercise 2:

Write a C++ program that finds the maximum element in a 2D array.

Code:

```

#include <iostream>
using namespace std;

int main() {
    // Declare and initialize a 3x3 matrix
    int matrix[3][3] = {
        {4, 2, 3},
        {5, 8, 1},
        {9, 6, 7}
    };

    // Initialize variables to store the maximum value and its
    position
    int maxValue = matrix[0][0];
    int maxRow = 0, maxCol = 0;

    // Loop through the matrix to find the maximum value
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (matrix[i][j] > maxValue) {

```



```

        maxValue = matrix[i][j];
        maxRow = i;
        maxCol = j;
    }
}

// Display the maximum value and its position
cout << "Maximum value: " << maxValue << " at position (" <<
maxRow << ", " << maxCol << ")" << endl;

return 0;
}

```

Exercise 3:

Write a C++ program that computes the transpose of a 2D array.

Code:

```

#include <iostream>
using namespace std;

int main() {
    // Declare and initialize a 2x3 matrix
    int matrix[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    // Declare a matrix to hold the transpose (3x2)
    int transpose[3][2];

    // Compute the transpose

```

```

for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        transpose[j][i] = matrix[i][j];
    }
}

// Display the original matrix
cout << "Original Matrix (2x3):\n";
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        cout << matrix[i][j] << " ";
    }
    cout << endl;
}

// Display the transposed matrix
cout << "\nTransposed Matrix (3x2):\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 2; j++) {
        cout << transpose[i][j] << " ";
    }
    cout << endl;
}

return 0;
}

```

Practice Tasks:

1. Write a C++ program that computes the sum of each row in a 2D array and displays the results.
2. Write a C++ program that checks if a given 2D square matrix is an identity matrix.
3. Write a C++ program that finds the row in a 2D array with the maximum sum.

Lab 17

Pointers in C++

Learning Objectives:

The objective of this lab is to understand the basic concepts of pointers in C++, including declaration, initialization, pointer arithmetic, and their applications in program writing.

Lab Exercises:

1. **Pointer Basics**
2. **Pointer Arithmetic**
3. **Swapping Values Using Pointers**

Exercise 1:

Write a C++ program that demonstrates the use of pointers.

Requirements:

1. Declare an integer variable and a pointer that points to it.
2. Initialize the variable and use the pointer to modify its value.
3. Display the value of the variable using both the variable and the pointer.

Code:

```
#include <iostream>
using namespace std;

int main() {
    // Step 1: Declare an integer variable and a pointer that
    points to it
    int variable = 10;          // Initialize the variable
    int* pointer = &variable; // Pointer pointing to the variable

    // Step 2: Use the pointer to modify the variable's value
    *pointer = 20; // Modify the value of the variable using the
    pointer
```

```

    // Step 3: Display the value of the variable using both the
    variable and the pointer
    cout << "Value of variable: " << variable << endl;        //
Using the variable
    cout << "Value using pointer: " << *pointer << endl;        //
Using the pointer

    return 0;
}

```

Exercise 2:

Write a C++ program that demonstrates pointer arithmetic with an array.

Requirements:

1. Declare and initialize an array of 5 integers.
2. Use a pointer to traverse the array and calculate the sum of the elements using pointer arithmetic.
3. Display the sum of the array elements.

Code:

```

#include <iostream>
using namespace std;

int main() {
    // Step 1: Declare and initialize an array of 5 integers
    int numbers[5] = {1, 2, 3, 4, 5};

    // Step 2: Use a pointer to traverse the array and calculate
the sum
    int sum = 0;
    int* ptr = numbers; // Pointer pointing to the first element
of the array

    for (int i = 0; i < 5; i++) {

```

```

        sum += *(ptr + i); // Accessing array elements using
pointer arithmetic
    }

    // Step 3: Display the sum of the array elements
    cout << "Sum of the array elements: " << sum << endl;

    return 0;
}

```

Exercise 3:

Write a C++ program that swaps the values of two integer variables using pointers.

Code:

```

#include <iostream>
using namespace std;

// Function to swap two integers using pointers
void swap(int* a, int* b) {
    int temp = *a; // Store the value at the address pointed by a
in temp
    *a = *b;        // Assign the value at the address pointed by b
to the address pointed by a
    *b = temp;      // Assign the value in temp to the address
pointed by b
}

int main() {
    // Step 1: Declare two integer variables and initialize them
    int x = 5;
    int y = 10;

```

```
// Display values before swapping
cout << "Before swap: x = " << x << ", y = " << y << endl;

// Step 2: Call the swap function
swap(&x, &y); // Pass the addresses of x and y to the swap
function

// Step 3: Display values after swapping
cout << "After swap: x = " << x << ", y = " << y << endl;

return 0;
}
```

Practice Tasks:

1. Write a C++ program that uses pointers to traverse and display the elements of an array.
2. Write a C++ program that calculates the sum of elements in an array using pointer arithmetic.

Lab 18

File Handling in C++

Learning Objectives:

The objective of this lab is to understand the basic operations of file handling in C++, including opening a file for reading, reading data from it, and displaying the contents.

Lab Exercises:

1. **Writing Data to a file.**
2. **Reading Data from a file.**
 - Write a C++ program that opens the file created in Exercise 1 and reads its contents using `ifstream`.
3. **Combining Reading from and Writing to a File.**
 - Write a C++ program that prompts the user for input, writes it to a file, and then reads from that same file to display the contents.

Exercise 1:

Write a C++ program that creates a file and writes some sample data to it using *ofstream*.

Code:

```
#include <iostream>
#include <fstream> // Include the fstream library for file
handling

using namespace std;

int main() {
    // Create an output file stream object
    ofstream outFile("sample_data.txt");

    // Check if the file opened successfully
    if (outFile.is_open()) {
```

```

        // Write sample data to the file
        outFile << "Hello, World!" << endl;
        outFile << "This is a sample file for file handling in
C++." << endl;
        outFile << "File handling is essential for data
persistence." << endl;
        outFile << "C++ makes it easy to read and write files." <<
endl;

        // Close the file after writing
        outFile.close();
        cout << "Data written to file successfully." << endl;
    } else {
        // Display an error message if the file could not be
opened
        cout << "Error opening file for writing." << endl;
    }

    return 0;
}

```

Exercise 2:

Write a C++ program that opens the file created in Exercise 1 and reads its contents using *ifstream*

Code:

```

#include <iostream>
#include <fstream>    // Include the fstream library for file
handling
#include <string>      // Include the string library for using
std::string

```



```

using namespace std;

int main() {
    // Create an input file stream object
    ifstream inFile("sample_data.txt");
    string line;

    // Check if the file opened successfully
    if (inFile.is_open()) {
        cout << "Contents of the file:" << endl;

        // Read each line from the file until the end is reached
        while (getline(inFile, line)) {
            cout << line << endl; // Display the line
        }

        // Close the file after reading
        inFile.close();
    } else {
        // Display an error message if the file could not be
opened
        cout << "Error opening file for reading." << endl;
    }

    return 0;
}

```

Exercise 3:

Write a C++ program that prompts the user for input, writes it to a file, and then reads from that same file to display the contents.

Code:

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main() {
    // Create an output file stream object
    ofstream outFile("user_input.txt");
    string userInput;

    // Prompt the user for input
    cout << "Enter three lines of text (press Enter after each
line):" << endl;
    for (int i = 0; i < 3; ++i) {
        getline(cin, userInput); // Get user input
        outFile << userInput << endl; // Write input to the file
    }
    outFile.close(); // Close the file after writing

    // Create an input file stream object to read from the file
    ifstream inFile("user_input.txt");

    // Check if the file opened successfully
    if (inFile.is_open()) {
        cout << "\nContents of the file:" << endl;
        string line;
```

```

        // Read each line from the file until the end is reached
        while (getline(inFile, line)) {
            cout << line << endl;    // Display the line
        }

        inFile.close();    // Close the file after reading
    } else {
        // Display an error message if the file could not be
opened
        cout << "Error opening file for reading." << endl;
    }

    return 0;
}

```

Practice Tasks:

Task 1: Student Record Management

Write a C++ program that manages student records. The program should:

- Prompt the user to enter details for multiple students, including name, ID, and grades.
- Write these records to a file named students.txt.
- Provide an option to read the file and display all student records.

Task 2: To-Do List Application

Develop a simple to-do list application in C++. The program should:

- Prompt the user to enter tasks to add to a to-do list.
- Write the tasks to a file named todo.txt.
- Provide an option to read the file and display all tasks.
- Allow the user to mark tasks as completed by writing to the same file (this can be done by overwriting the existing file).

Task 3: Library Book Inventory

Write a C++ program that manages a library's book inventory. The program should:

- Allow the user to enter book titles, authors, and publication years.
- Write this information to a file named library_inventory.txt.
- Provide an option to read from the file and display the list of all books in the inventory.

Task 4: Contact Management System

Design a simple contact management system. The program should:

- Allow the user to input contact details (name, phone number, email).
- Write the contacts to a file named contacts.txt.
- Provide an option to read from the file and display all stored contacts.

Suggested Small Semester Projects

1. Student Management System

Description: Develop a comprehensive system to manage student records.

Features:

- Add, delete, and update student details (name, ID, grades).
- Save records to a file and read from it.
- Calculate and display statistics like average grades and highest/lowest scores.

2. Library Management Systems

Description: Create a system for managing library books and members.

Features:

- Add and remove books, and manage member details.
- Check in and check out books, maintaining records in a file.
- Display current inventory and member statistics.

3. Quiz Application

Description: Create a multiple-choice quiz application.

Features:

- Load questions from a file and present them to the user.
- Keep track of scores and provide feedback on correct/incorrect answers.
- Save the quiz results to a file for review.

4. Simple Banking System

Description: Develop a basic banking application to manage accounts.

Features:

- Create accounts with names and balances.
- Allow deposits, withdrawals, and transfers between accounts.
- Save account details to a file and load them for future sessions.

5. Simple Quiz Application

Description: Develop a quiz application that stores questions in a file

Features:

- Load quiz questions from a file and present them to the user.
- Store user scores and allow review of answers.
- Save quiz results to a file for later analysis.

"Good code is its own best documentation."

– Steve McConnell