Department of Electronic Engineering

NED University of Engineering & Technology

PRACTICAL WORKBOOK

For the course

Computer and Programming (EL-105) F.E (Electronic)

Instructor Name: _ __

Student Name: _ __

Roll no: Batch: _ __

Semester: Year: _ _

Department: ____

LABORATORY WORKBOOK

FOR THE COURSE

EL-105 COMPUTER AND PROGRAMMING

Prepared by

Dr. Danish Mahmood Khan

Assistant Professor

Approved By

THE BOARD OF STUDIES OF DEPARTMENT OF ELECTRONIC ENGINEERING

CONTENTS

S. NO	Level	List of Experiments
1	-	To provide students with hands-on experience to understand the basic components of a computer system, the role of peripherals, and the functional units of a CPU.
2	C5	Introduction to C++ Integrated Development Environment (IDE), installation, and an overview of its features.
3	C5	To examine the fundamental components of the C++ language, including data types and input-output functions, to identify their characteristics, distinctions, and how they contribute to program functionality.
4	C5	To synthesize the principles of decision making in C++ using if and nested if statements, analyze complex scenarios, and propose logical solutions, demonstrating a higher-level understanding of control flow and problem-solving.
5	C5	To analyze and evaluate the functioning of conditional statements, such as if-else, else-if, nested if-else, and Switch-Case within the context of C++ programming, and to discern their respective applications and implications in program flow control.
6	C5	To cultivate a comprehensive understanding of C++ loops by employing for loops and nested for loops in diverse scenarios and demonstrate proficiency in basic and advanced operations, comprehend multi-dimensional data processing, generate dynamic patterns, and optimize loop performance for enhanced code efficiency.
7	C5	To develop a thorough understanding of while and do-while loops in C++ by adeptly applying these constructs in various contexts. Demonstrate skill in executing fundamental and advanced operations, grasp intricate details in iterative data processing, and optimize loop performance for enhanced code efficiency.
8	C5	To gain a comprehensive understanding of structures in C++, covering their definition, declaration, and implementation. Apply this knowledge to create and manipulate structured data, demonstrating proficiency in solving programming challenges through the effective utilization of C++ structures.
9	C5	To demonstrate the principles and execution of functions in C++ and create user-defined functions, showcasing an advanced application of programming concepts.
10	C5	To comprehend the concepts of arrays and strings in C++, including their implementation and manipulation, and to apply these acquired skills in addressing a variety of practical scenarios.
11	C5	To demonstrate effective utilization of pointers in tasks such as dynamic memory allocation, data manipulation, and code optimization for addressing and solving programming challenges.
12	C5	Synthesize object-oriented programming principles to design and implement classes, demonstrate inheritance relationships, and create effective object-oriented solutions for various real-life problems.
13	C5	Open Ended Lab

S. No	**Date**	**Lab Session**	**Marks**	**Sign**
1		Lab Session 01		
2		Lab Session 02		
3		Lab Session 03		
4		Lab Session 04		
5		Lab Session 05		
6		Lab Session 06		
7		Lab Session 07		
8		Lab Session 08		
9		Lab Session 09		
10		Lab Session 10		
11		Lab Session 11		
12		Lab Session 12		
13		Lab Session 13		

Lab Session 1

Objective:

To provide students with hands-on experience to understand the basic components of a computer system, the role of peripherals, and the functional units of a CPU.

Components:

- Computers (laptops or desktops)
- Projector and screen
- Various computer peripherals (keyboard, mouse, printer, scanner, etc.)
- CPU chip (for demonstration purposes)
- Motherboard (for demonstration purposes)
- Power supply unit (for demonstration purposes)
- RAM modules (for demonstration purposes)
- Hard drive (for demonstration purposes)
- Various cables (power cables, USB cables, etc.)

Theory:

Computer

A computer is an electronic machine that takes input from the user (data), processes the given input, and generates output in the form of useful information.

The most elementary computing concepts as shown in Figure 1-1 include receiving input—known as data— from the user, manipulating the input according to the given set of instructions and delivering the output—known as information—to the user.

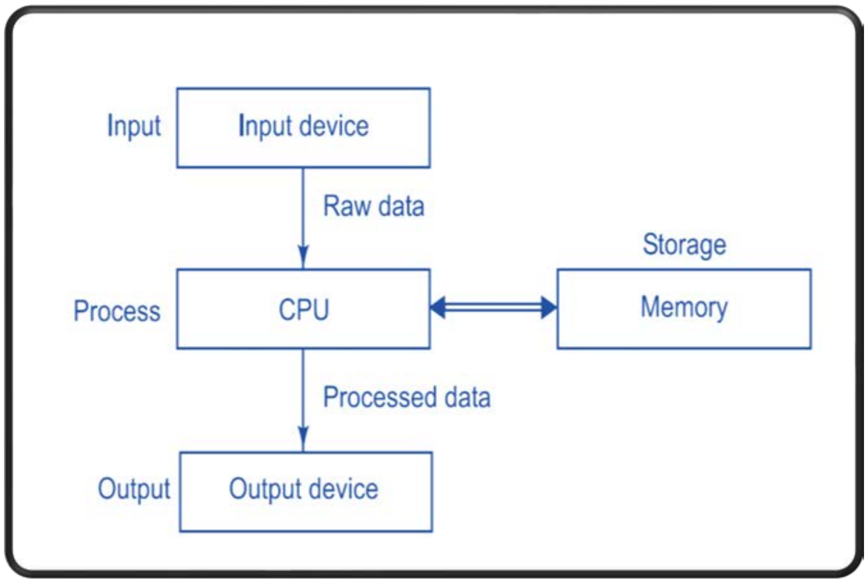


Figure 1-1 Elementary Computing Concept

Peripheral Devices:

Peripheral devices are hardware components that connect to a computer and provide additional functionality and features. These devices are called "peripherals" because they are auxiliary to the central processing unit (CPU) and main components of the computer. Peripheral devices serve various purposes, such as input, output, or storage.

Central Processing Unit:

The CPU (Central Processing Unit) is the core component of a computer responsible for executing instructions and performing data processing. The CPU consists of several functional units that work together to carry out these tasks. These functional units include:

- Arithmetic Unit
- Logic Unit
- Control Unit
- Main Memory Unit
- Registers

Procedure:

Remind yourself of the importance of handling computer components with care and ensuring that the computer is powered off and unplugged before any internal exploration.

a. Hands-on Peripheral Interaction:

1. Ensure that all the necessary peripherals, including the keyboard and mouse, are connected to the computer.
2. Start by practicing typing on the keyboard.
3. Type a few sentences, words, or your name into a text document to get a feel for the keyboard.
4. Move the mouse on the mousepad to see how it moves the cursor on the screen.
5. Click the left and right mouse buttons to understand their functions.
6. Try dragging and releasing.
7. Open a sample document or application.
8. Practice using the mouse to:
9. Click on icons, buttons, or links.
10. Right-click to access context menus.
11. Scroll using the mouse wheel.
12. If other peripherals are available (e.g., printer or scanner) explore and interact with these peripherals under supervision.

b. Opening a Computer Case:

1. Before proceeding, ensure that the computer is powered off and unplugged from the electrical outlet.
2. Locate the computer case, which is the outer housing of the desktop computer.
3. Ensure you have a screwdriver ready for opening the case.
4. Observe the computer case for screws or fasteners that secure the side panel.
5. Commonly, there are two or more screws on the rear side of the case.
6. Use the screwdriver to carefully remove the screws from the side panel of the computer case.
7. Place the screws in a safe location, so they aren't lost.
8. Depending on the computer case design, you may need to slide the side panel back or unscrew a latch to open it.
9. Gently slide or open the side panel to reveal the internal components.

c. Interactive CPU Demonstration:

1. Once the case is open, take a moment to visually inspect the interior of the computer.
2. Look for the major components, including the motherboard, CPU (Central Processing Unit), and RAM (Random Access Memory).
3. Identify the largest circuit board inside the computer. This is the motherboard.
4. Observe the various connectors, slots, and ports on the motherboard.
5. Locate the CPU, which is a small chip mounted on the motherboard.
6. Take note of its position and any markings or labels on the CPU.
7. Identify the RAM modules, which are usually small, rectangular chips mounted on the motherboard.
8. Note the number of RAM modules and their locations.

d. Closing a Computer Case:

1. After inspecting the internal components, carefully close the computer case.
2. Make sure it is securely fastened and the screws are reattached.

Observations:

Results:

Lab Session 2

Objective:

To analyze the C++ Integrated Development Environment (IDE) with a focus on its installation process and a comprehensive overview of its features. Identify and evaluate key components, tools, and functionalities within the IDE to gain a deeper understanding of its capabilities.

Code::Blocks IDE:

Code::Blocks shown in **Error! Reference source not found.** is an open-source integrated development environment (IDE) primarily used for C and C++ programming. It is available on multiple platforms, including Windows, Linux, and macOS.

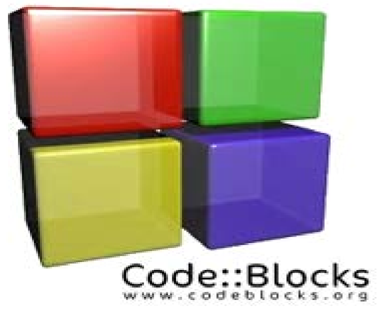


Figure 2-1 Code::Blocks Logo

Code::Blocks Features Overview:

- **Cross-Platform Support**: Code::Blocks is available for Windows, Linux, and macOS, making it accessible on a wide range of operating systems.
- **Customizable Interface:** You can customize the user interface to suit your preferences. This includes the ability to change themes and layouts.
- **Code Editor:** The IDE provides a code editor with features like syntax highlighting, code folding, code completion, and the ability to work with multiple source files simultaneously.
- **Compiler Integration:** Code::Blocks supports multiple C and C++ compilers, including GCC (GNU Compiler Collection) and Microsoft Visual C++. You can configure the IDE to work with your preferred compiler.
- **Build and Debugging Tools:** It offers built-in build and debugging tools. You can compile and run your programs directly from the IDE. Debugging features include breakpoints, watch variables, call stack, and a GUI-based debugger.
- **Project Management:** You can manage your projects with ease using Code::Blocks. It supports various project types, including console applications, GUI applications, and dynamic link libraries (DLLs).
- **Code Templates:** The IDE includes code templates and code snippets for C and C++ to help streamline development and reduce typing.
- **Plugins and Extensions:** Code::Blocks has a plugin architecture that allows you to extend its functionality. There are various plugins available to enhance the IDE's features further.
- **Integrated Development Tools:** It provides tools for version control, like Subversion, and integrated development support for popular libraries and frameworks.
- **Resource Management:** Code::Blocks allows you to manage resources like images, icons, and other assets used in your projects.
- **Auto-Completion:** The IDE provides auto-completion and code suggestions, which can improve your coding productivity.
- **Project Wizards:** You can use project wizards to quickly set up new projects with predefined configurations.
- **Multiple Compiler Support**: As mentioned earlier, Code::Blocks supports multiple compilers. This

flexibility allows you to choose the best compiler for your needs.

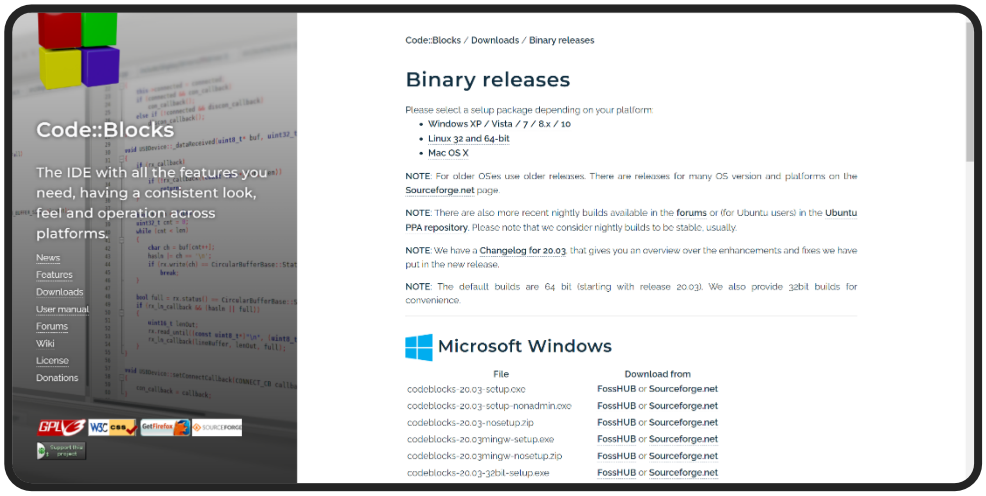
- **Multi-Language Support:** While it is primarily used for C and C++ development, Code::Blocks also supports other programming languages, making it a versatile IDE.
- **Free and Open Source:** Code::Blocks is open-source software, which means it is free to download, use, and modify. It has an active community of developers, which can be helpful for finding support and updates.

Code::Blocks Installation Guide

Download:

To download the IDE on Microsoft Windows, follow these steps. For other operating systems, please download the compatible file.

1. Open your preferred web browser and go to www.codeblocks.org/downloads.
2. Click on "Download the Binary Release."
3. Download the Code::Blocks with Mingw setup file as shown in Figure 2-2. As of the time of writing this text, the file is named " codeblocks-20.03mingw-setup.exe."
4. Your installation is now complete, and you're all set to get started.



The IDE with all the features you need, having a consistent look, feel and operation across platforms.

Code::Blocks / Downloads / Binary releases

Binary releases

Please select a setup package depending on your platform:

- Windows XP / Vista / 7 / 8.x / 10
- Linux 32 and 64-bit
- Mac OS X

NOTE: For older OSes use older releases. There are releases for many OS version and platforms on the Sourceforge.net page.

NOTE: There are also more recent nightly builds available in the forums or (for Ubuntu users) in the Ubuntu PPA repository. Please note that we consider nightly builds to be stable, usually.

NOTE: We have a Changelog for 20.03, that gives you an overview over the enhancements and fixes we have put in the new release.

NOTE: The default builds are 64 bit (starting with release 20.03). We also provide 32bit builds for convenience.

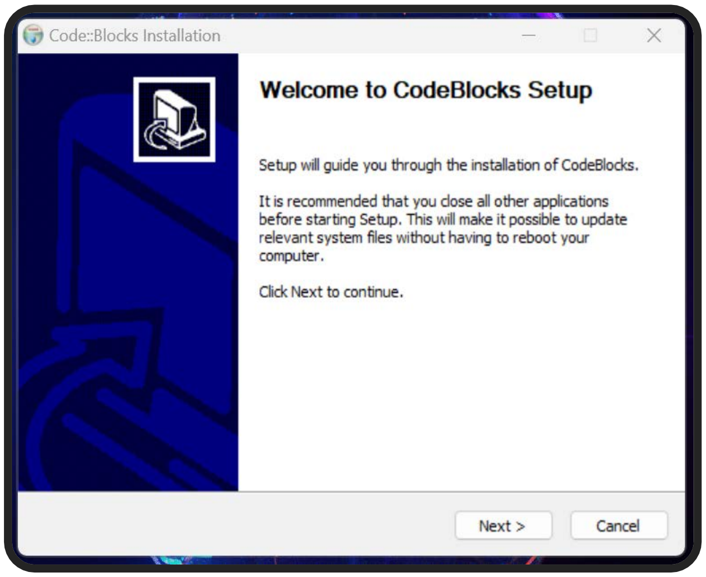
Microsoft Windows

File	Download from
codeblocks-20.03-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03-setup-nonadmin.exe	FossHUB or Sourceforge.net
codeblocks-20.03-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03mingw-setup.exe	FossHUB or Sourceforge.net
codeblocks-20.03mingw-nosetup.zip	FossHUB or Sourceforge.net
codeblocks-20.03-32bit-setup.exe	FossHUB or Sourceforge.net

Figure 2-2 Obtaining Code::Blocks Installation Package

Installation:

1. The installation process is straightforward. Execute the previously downloaded executable file (or the one obtained from the Computer Lab).
2. Running the setup file will initiate the installation wizard (Figure 2-3), which will seamlessly guide you through the entire process.


Code::Blocks Installation

Welcome to CodeBlocks Setup

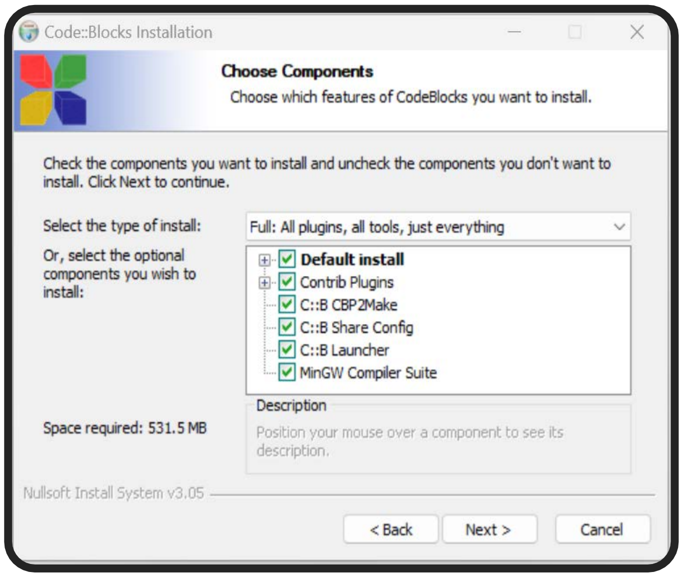
Setup will guide you through the installation of CodeBlocks.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 2-3 Installation Process Wizard Guide

3. Click Next.
4. Read and accept the license terms to proceed with the installation of Code::Blocks (click 'I Agree' in after reviewing the terms).
5. Once you have agreed to the terms, the installation wizard will prompt you to select the components to install as shown in Figure 2-4 . Ensure that all components are checked, and then click 'Next' to continue.


Code::Blocks Installation

Choose Components

Choose which features of CodeBlocks you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Full: All plugins, all tools, just everything

Or, select the optional components you wish to install:

- ✓ Default install
- ✓ Contrib Plugins
- ✓ C::B CBP2Make
- ✓ C::B Share Config
- ✓ C::B Launcher
- ✓ MinGW Compiler Suite

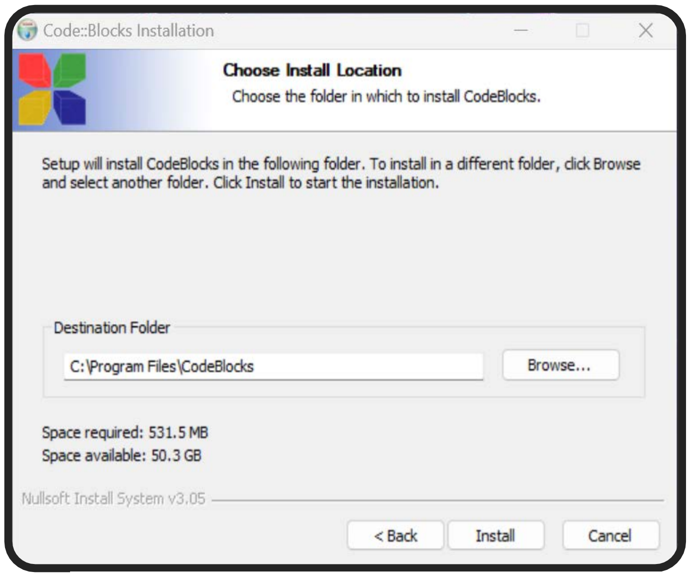
Description

Position your mouse over a component to see its description.

Space required: 531.5 MB

Nullsoft Install System v3.05

Figure 2-4 Components to install


Code::Blocks Installation

Choose Install Location

Choose the folder in which to install CodeBlocks.

Setup will install CodeBlocks in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder

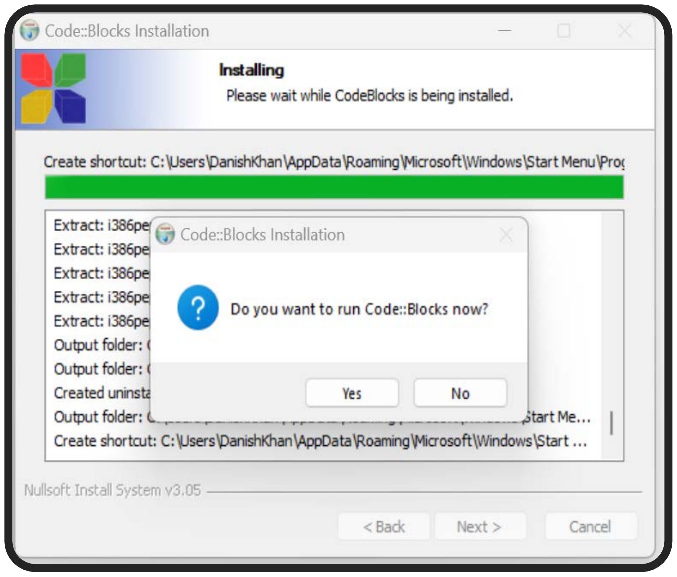
C:\Program Files\CodeBlocks

Space required: 531.5 MB
Space available: 50.3 GB

Nullsoft Install System v3.05

Figure 2-5 Choose destination

6. Next, choose the installation location on your hard disk for Code::Blocks (Figure 2-5). Using the default location is recommended.
7. After clicking the "Install" button, the installation process will begin.
8. Upon successful installation, a confirmation message will be displayed (Figure 2-7).



Code::Blocks Installation

Installing

Please wait while CodeBlocks is being installed.

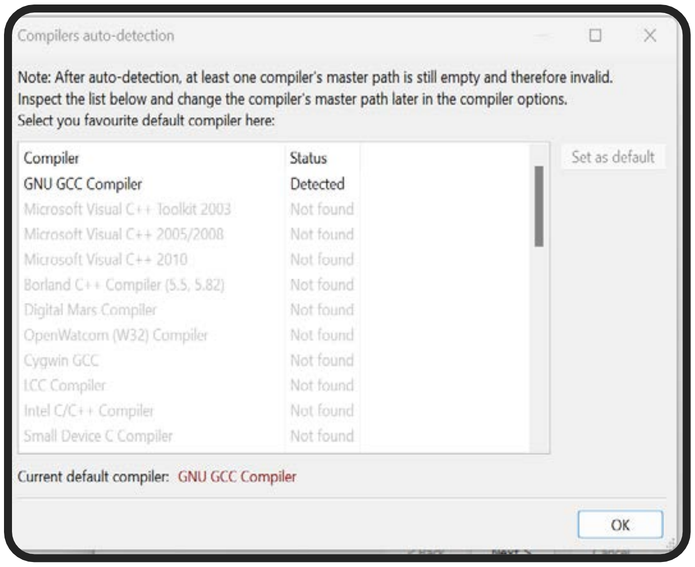
Create shortcut: C:\Users\DanishKhan\AppData\Roaming\Microsoft\Windows\Start Menu\Prog

Extract: i386pe
Extract: i386pe
Extract: i386pe
Extract: i386pe
Extract: i386pe
Output folder:
Output folder:
Created uninsta
Output folder: ... Start Me...
Create shortcut: C:\Users\DanishKhan\AppData\Roaming\Microsoft\Windows\Start ...

Code::Blocks Installation

Do you want to run Code::Blocks now?

Figure 2-7 Installation Successful



Compilers auto-detection

Note: After auto-detection, at least one compiler's master path is still empty and therefore invalid. Inspect the list below and change the compiler's master path later in the compiler options.
Select you favourite default compiler here:

Compiler	Status
GNU GCC Compiler	Detected
Microsoft Visual C++ Toolkit 2003	Not found
Microsoft Visual C++ 2005/2008	Not found
Microsoft Visual C++ 2010	Not found
Borland C++ Compiler (5.5, 5.82)	Not found
Digital Mars Compiler	Not found
OpenWatcom (W32) Compiler	Not found
Cygwin GCC	Not found
LCC Compiler	Not found
Intel C/C++ Compiler	Not found
Small Device C Compiler	Not found

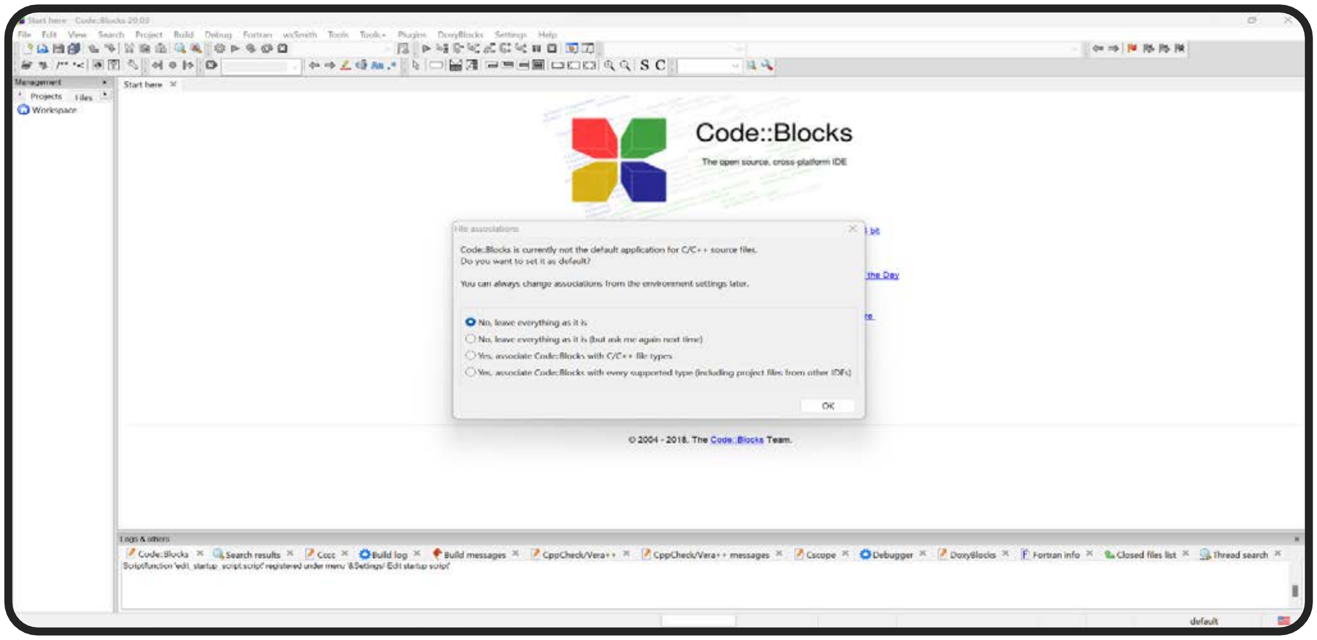
Current default compiler: GNU GCC Compiler

Figure 2-6 Compiler selection

9. Once the installation is complete, you can choose between two options: "Finish Installation" or "Run Code::Blocks."
10. You may be prompted to select compilers that are already installed on your PC.
11. Select "GNU GCC Compiler" from the list.

Running Code::Blocks:

1. Locate the Code::Blocks icon on your desktop or in your application menu, then double-click it to launch the IDE.



File associations

Code::Blocks is currently not the default application for C/C++ source files.
Do you want to set it as default?

You can always change associations from the environment settings later.

[✓ No, leave everything as it is]
No, leave everything as it is (but ask me again next time)
Yes, associate Code::Blocks with C/C++ file types
Yes, associate Code::Blocks with every supported type (including project files from other IDEs)

Figure 2-8 Code::Blocks IDE

2. You can choose to set it as the default program for running C/C++ code (Figure 2-8).
3. Create a new empty file (use the shortcut Ctrl + Shift + N) as shown in Figure 2-9.

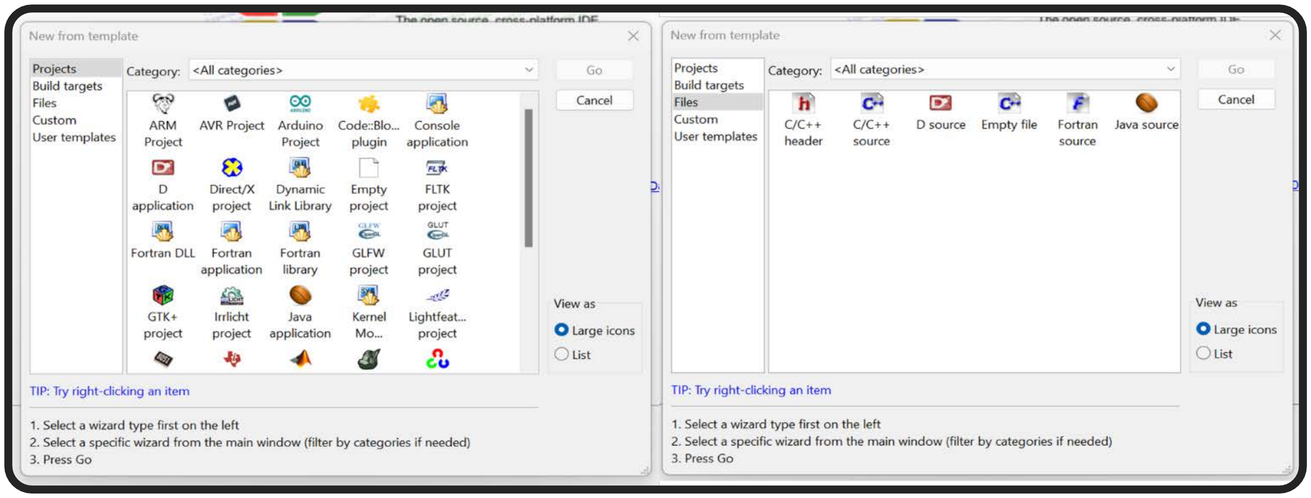


Figure 2-9 Creating an Empty C++ File

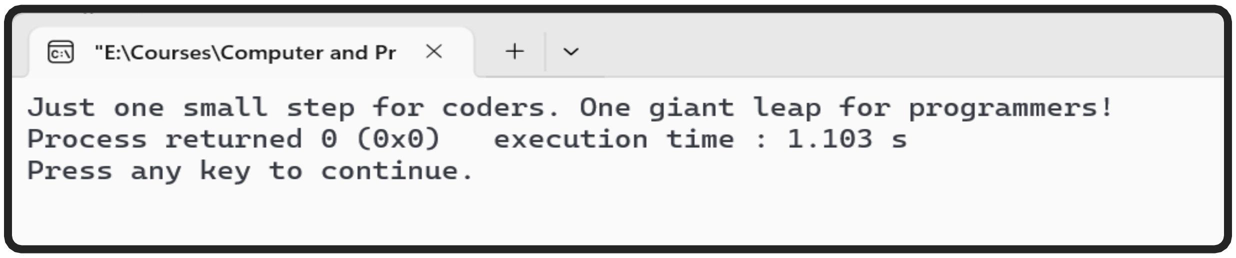
4. Save the file with the name "Lab_02_Pgm_01.cpp."
5. Be sure to save it in the correct format, which should be ".cpp."

Creating Your First C++ Program:

1. Enter the following code into the blank file.

```
#include<iostream>
using namespace std;
int main() {
   cout<<"Just one small step for coders. One giant leap for";
   cout<<" programmers!";
return 0;
}
```

2. After entering your code, navigate to "BUILD" and select "BUILD and RUN" (or simply press F9 for a shortcut).



```
"E:\Courses\Computer and Pr
Just one small step for coders. One giant leap for programmers!
Process returned 0 (0x0)   execution time : 1.103 s
Press any key to continue.
```

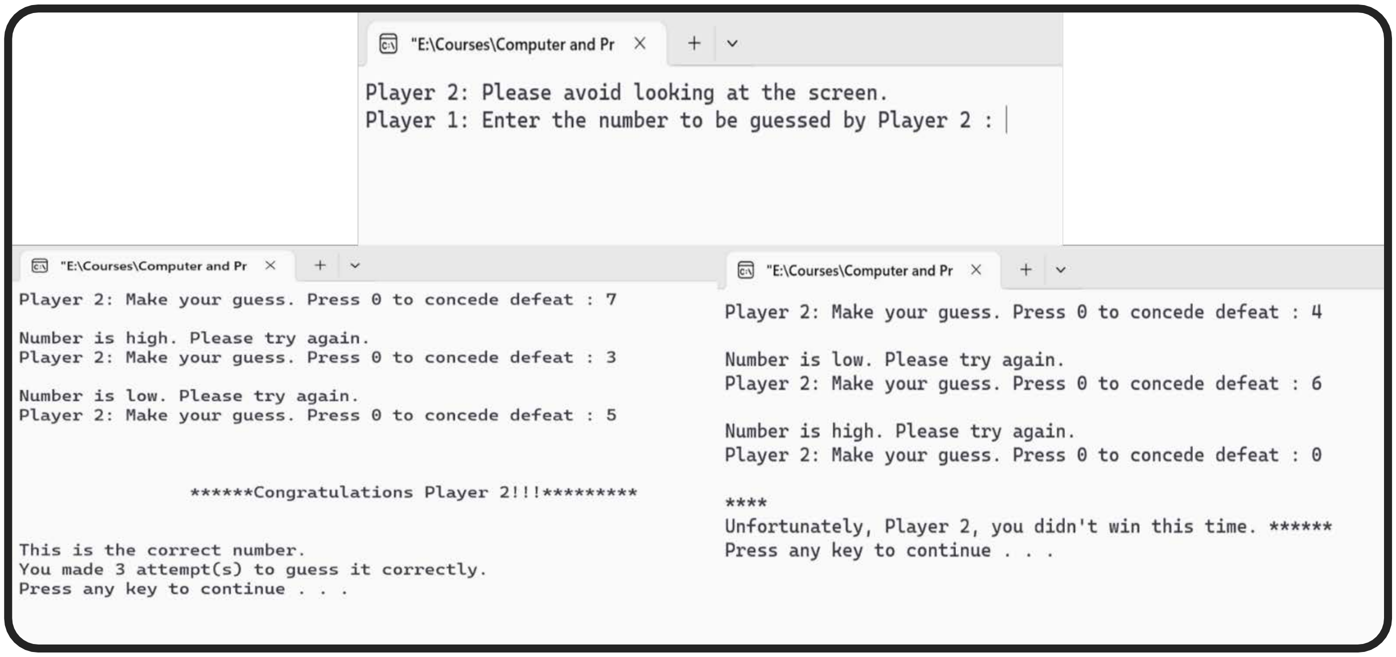
Figure 2-10 Output: Lab_02_Pgm_01.cpp

3. If your program compiles successfully, it will execute (Output is shown in Figure 2-10). In case of

any compilation errors, you will receive an error message.

Playing Your First C++ Game:

1. Begin by double-clicking the "Lab02_GuessingGame.exe" file, which has already been provided to you.
2. This is a two-player game, where one player inputs a number, and the other player's task is to guess it.
3. Simply follow the in-game instructions provided (Figure 2-11).
4. Don't worry if you don't fully grasp the underlying code at this stage. You'll have the opportunity to code this game in future lab sessions.


"E:\Courses\Computer and Pr
"E:\Courses\Computer and Pr
"E:\Courses\Computer and Pr

```
Player 2: Please avoid looking at the screen.
Player 1: Enter the number to be guessed by Player 2 :
```

```
Player 2: Make your guess. Press 0 to concede defeat : 7

Number is high. Please try again.
Player 2: Make your guess. Press 0 to concede defeat : 3

Number is low. Please try again.
Player 2: Make your guess. Press 0 to concede defeat : 5


                ******Congratulations Player 2!!!*********


This is the correct number.
You made 3 attempt(s) to guess it correctly.
Press any key to continue . . .
```

```
Player 2: Make your guess. Press 0 to concede defeat : 4

Number is low. Please try again.
Player 2: Make your guess. Press 0 to concede defeat : 6

Number is high. Please try again.
Player 2: Make your guess. Press 0 to concede defeat : 0

****
Unfortunately, Player 2, you didn't win this time. ******
Press any key to continue . . .
```

Figure 2-11 Snippets from the Guessing Game Output

Exercise:

Q1. Given the code snippets below, identify and correct the syntax errors to make each code snippet compile and run successfully. Explain the errors you found and the corrections you made.

SNo	Incorrect Code	Correct Code	Explanation

(i)	#include <iostream>
	using namespace std;
	int main() {
	   cout << "Hello, World!"
	   return 0;
	}		

(iii)	#include <iostrm>
	using namespace std;
	int main() {
	   cout << "Hello, World!";
	   return 0;
	}		

(iv)	#include <iostream>
	using namespace std;
	int add(int a, int b) {
	   return a + b;
	}		

(v)	#include <iostream>
	using namespace std;
	int Main() {
	   cout << "Hello, World!";
	   return 0;
	}		

(vi)	#include <iostream>
	using namespace std;
	int main() {
	   cout << "Hello, World!";
	   return (0;
	}		

	#include <iostream>
	using namespace std;
	int main() {
	   cout >> "Hello, World!";
	   return 0;
	}		

Q2. Write a C++ program to display the following information:

Full Name: [Your Full Name]

Seat Number: [Your Seat Number]

Favorite Teacher: [Your Favorite Teacher's Name]

Code:

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ______________________ Date: ____________________

Software Use Rubric

Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 3

Objective:

To examine the fundamental components of the C++ language, including data types and input-output functions, to identify their characteristics, distinctions, and how they contribute to program functionality.

Theory:

This lab explores the fundamental elements that serve as the building blocks of C++ programming. These crucial components cover the C++ character set, identifiers, keywords, data types, constants, variables, expressions, statements, and escape sequences. Within the scope of this lab, you will study the following key topics:

- Basic data types in C++
- Declaring and using variables
- Comments in a C++ program
- Printing variable values with cout
- Interactive computing with cin
- Escape sequences.

This comprehensive exploration equips you with the foundational knowledge necessary to excel in C++ programming, providing the theoretical underpinning for practical application and problem-solving in this versatile language.

Basic Data Types in C++:

In C++, basic data types are fundamental building blocks used to define and store different types of data in a program. Some of the commonly used basic data types in C++ are shown in Figure 3-1.

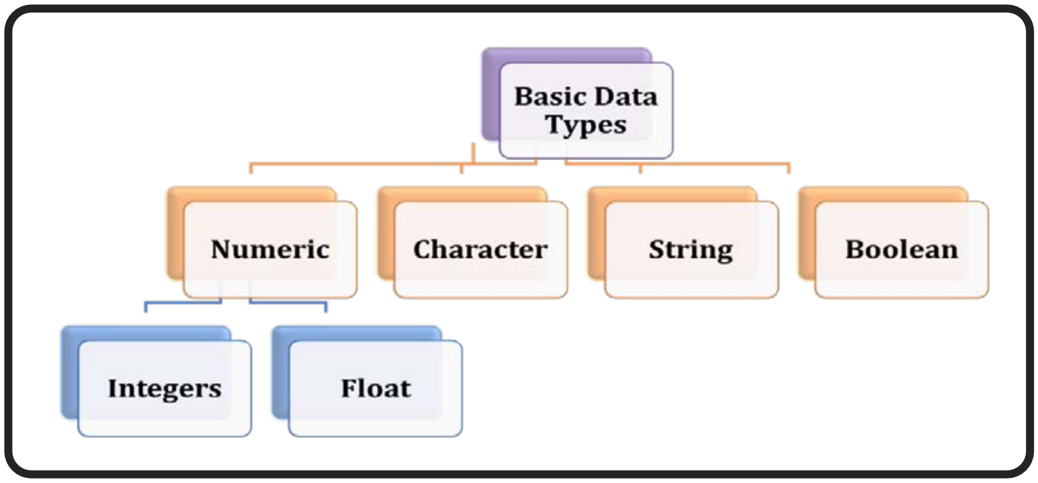


Figure 3-1 Basic Data Types in C++

- **Numeric:** Numeric data includes a wide range of numbers, encompassing both integers (whole numbers) and floating-point values (numbers with decimal points). Below are examples of numeric data:
  - Integer: These are whole numbers, such as 5, -10, 0, and 100.
  - Floating-Point: These are numbers with decimal points, like 3.14, -0.5, 2.71828, and 100.0.
  - Double: A higher-precision floating-point type, e.g., 3.14159265359, -0.0001, and

123.456789.
    - Long: Used for larger integer values, e.g., 1,000,000, -987654321.
    - Short: Used for smaller integer values, e.g., 42, -32768, 100.
- **Character:** Character data comprises alphanumeric characters and special symbols, each enclosed within single quotes. Some examples of character data are as follows:
    - 'A': An uppercase letter 'A'.
    - '3': The numeric digit '3'.
    - '$': The dollar sign symbol.
    - '!': An exclamation mark.
    - ' ': A space character.
- **String:** It consists of text values enclosed within double quotes. Examples of string data are given below:
    - "Hello, World!": A simple greeting.
    - "12345": A string representation of numeric digits.
    - "OpenAI": A string representing a company name.
    - "C++ Programming": A string containing spaces and multiple characters.
    - "Special characters: @#$%": A string with various symbols.
    - "": An empty string with no characters.
- **Boolean:** Boolean data consists of two values: true and false.

Demonstrating Data Type Usage in C++

```
#include <iostream>
using namespace std;
int main() {
  int integerVar = 42;    float floatVar = 3.14;
  double doubleVar = 2.71828;    char charVar = 'A';
  bool boolVar = true;   string stringVar = "Hello, World!";
  cout << "Integer: " << integerVar << ", Float: " << floatVar << ", Double: " << doubleVar
     << ", Character: " << charVar << ", Boolean: " << boolVar << ", String: " << stringVar;
  return 0;}
```

Variable Declaration and Usage in C++:

In C++, a variable is a fundamental programming concept used to store and manipulate data. It is essentially a named storage location in the computer's memory where programmer can assign and retrieve values during program execution.

Variable declaration is the process of specifying the variable's name and data type, which determines the kind of data it can store. For example, one can declare an integer variable named "age" by writing**: int age;** This informs the compiler to set aside memory for an integer and associate it with the name "age" for later use in the program. Variable declarations are crucial for managing data in C++ programs, and they set the foundation for efficient memory allocation and data manipulation. Table 3-1 displays the keywords and memory requirements for various data types.

Table 3-1 Data Types, Their Keywords, and Memory Allocation

Data Type	Keyword	Bytes Allocation
Integer	int	4
Float	float	4
Character	char	1
Boolean	bool	1
Double	double	8
String	string	varies

The following code can be utilized to ascertain the memory requirements of different data types.

```
#include <iostream>
using namespace std;
int main() {
   cout << "Size of int: " << sizeof(int) << " bytes" << endl;
   cout << "Size of float: " << sizeof(float) << " bytes" << endl;
   cout << "Size of double: " << sizeof(double) << " bytes" << endl;
   cout << "Size of char: " << sizeof(char) << " bytes" << endl;
   cout << "Size of bool: " << sizeof(bool) << " bytes" << endl;
   cout << "Size of long: " << sizeof(long) << " bytes" << endl;
   cout << "Size of short: " << sizeof(short) << " bytes" << endl;    return 0;}
```

Rules for Variable Naming:

- Variable names must start with a letter or an underscore (_).
- They may contain letters, numbers, and the underscore character only; no other special characters are allowed.
- Uppercase and lowercase letters are treated as distinct in variable names.
- A variable name should not be a reserved keyword.
- It should not have the same name as a built-in or user-defined function.

Input and Output Functions (cout and cin) in C++

C++ provides two primary I/O functions for handling output and input: cout for output and cin for input. These are part of the C++ Standard Library and are associated with the iostream library.

```
#include <iostream>
using namespace std;
int main() {
   int age;
   cout << "Enter your age: ";    cin >> age;
   cout << "You entered: " << age << " years." << endl;    return 0;}
```

Comments

Comments are used solely to enhance program readability and properly document the program. Comment statements will not be compiled. Comments begin with /* and end with */, with text placed in between them. For single-line comments, // is used.

Example:

```
// This is a single-line comment. It is used for brief notes or explanations.
/* This is a multi-line comment.
   It provides detailed explanations of code blocks. */
```

Escape Sequences:

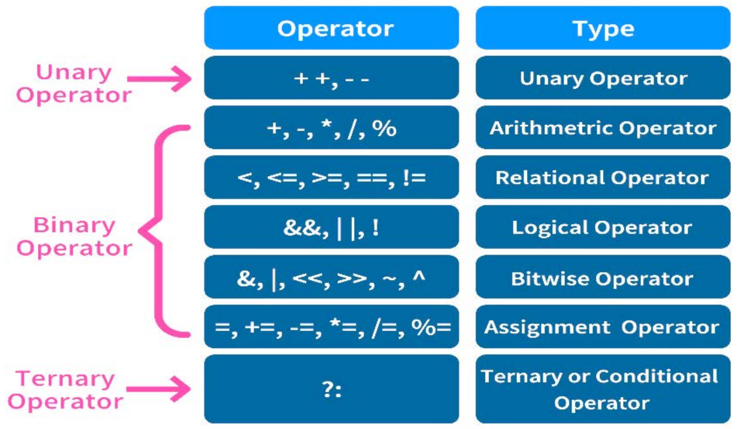
Escape sequences are special character sequences used in string literals and character literals to represent characters that are not easily typable or visible, as well as to insert formatting characters. Escape sequences start with a backslash (\) followed by one or more characters, and they have specific meanings. Explanation of commonly used escape sequences in C++ is given in Table 3-2.

Table 3-2 Common Escape Sequences in C++

Sequence	Purpose
Newline: \n	Represents a newline character. When printed or displayed, it moves the cursor to the beginning of the next line.
Carriage return: \r	Represents a carriage return character. It moves the cursor to the beginning of the line.
Horizontal Tab: \t	Represents a horizontal tab character. It's used for creating horizontal spacing or indentation.
Vertical Tab: \v	Represents a vertical tab character. It's less commonly used and may not have a visible effect in many environments.
Alert Beep: \a	Produces audible or visual alert. Its behavior can vary depending on the platform or environment.
Backspace: \b	Represents a backspace character. It moves the cursor one position to the left but doesn't erase the character.
Backslash: \\	Represents a literal backslash character. It's used when you want to include a backslash in a string or character literal.
Single Quote: \’	Represents a single quotation mark. It's used to include single quotes within a character literal.
Double Quotes: \”	Represents a double quotation mark. This is used to include double quotes within a string literal.
Null Character: \0	Represents the null character, which has a numeric value of 0.

Operators:

Operators are symbols that represent operations to be performed on operands. Operands can be variables, constants, expressions, or values. As illustrated in Figure 3-2., C++ provides a wide range of operators for various types of operations, including arithmetic, relational, logical, bitwise, assignment, and more.



	Operator	Type
Unary Operator	+ +, - -	Unary Operator
Binary Operator	+, -, *, /, %	Arithmetic Operator
	<, <=, >=, ==, !=	Relational Operator
	&&, | |, !	Logical Operator
	&, |, <<, >>, ~, ^	Bitwise Operator
	=, +=, -=, *=, /=, %=	Assignment Operator
Ternary Operator	?:	Ternary or Conditional Operator

Figure 3-2 Operators in C++

Usage in C++:

```
#include <iostream>
using namespace std;
int main() {
  // Arithmetic operators
  int a = 10, b = 5;
  cout << "Arithmetic Operators:" << endl;        cout << "a + b = " << a + b << endl;
  cout << "a - b = " << a - b << endl;             cout << "a * b = " << a * b << endl;
  cout << "a / b = " << a / b << endl;             cout << "a % b = " << a % b << endl;

  // Relational operators
  cout << "\nRelational Operators:" << endl;     cout << "a == b: " << (a == b) << endl;
  cout << "a != b: " << (a != b) << endl;          cout << "a > b: " << (a > b) << endl;
  cout << "a < b: " << (a < b) << endl;            cout << "a >= b: " << (a >= b) << endl;
  cout << "a <= b: " << (a <= b) << endl;

  // Logical operators
  bool x = true, y = false;
  cout << "\nLogical Operators:" << endl;         cout << "x && y: " << (x && y) << endl;
  cout << "x || y: " << (x || y) << endl;          cout << "!x: " << !x << endl;

  // Bitwise operators
  int m = 5, n = 3;
  cout << "\nBitwise Operators:" << endl;          cout << "m & n = " << (m & n) << endl;
  cout << "m | n = " << (m | n) << endl;           cout << "m ^ n = " << (m ^ n) << endl;
  cout << "~m = " << ~m << endl;

  // Assignment operators
  int result = 0;          result += 10;
  cout << "\nAssignment Operators:" << endl;
  cout << "result += 10: " << result << endl;

  return 0; }
```

Operators Precedence and Associativity:

Operator precedence determines the order in which operators are evaluated within an expression. Operators with higher precedence are evaluated before operators with lower precedence. When writing an expression, it's essential to know which operator takes precedence to ensure that the expression is evaluated as intended. The precedence of common operators is given in Table 3-3.

Example:

In C++, multiplication (*) has higher precedence than addition (+). So, in the expression 3 + 5 * 2, the multiplication is performed first because * has higher precedence, resulting in 3 + 10, which evaluates to 13.

Operator associativity determines the order of evaluation when operators with the same precedence appear in an expression. It specifies whether operators are evaluated from left to right (left-associative) or right to left (right-associative).

Example:

Addition (+) is left-associative, which means that in the expression 3 + 5 + 2, the leftmost + is evaluated first, followed by the next leftmost +, resulting in (3 + 5) + 2, which equals to 10.

Table 3-3 Precedence of Arithmetic, Relational, Logical, and Assignment Operators

Operators	**Operations**	**Precedence**
()	Parentheses	Evaluated first. If parentheses are nested, the innermost expression is evaluated first.
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are multiple evaluations, they are performed from left to right.
+, -	Addition, Subtraction	Evaluated last. If there are several, they are evaluated from left to right.
<, >, <=, >=, ==, !=	Relational Operation	Comparison or relational operators have lower precedence than arithmetic operators.
&&, ||,!	Logical Operation	Logical operators have lower precedence than comparison operators.
=, +=, -=	Assignment Operation	Assignment operators have lower precedence than most other operators.

Exercises:

Q1. Write the output of the following statements and provide a brief explanation of how each statement works.

Sno.	**Statements**	**Output**	**Explanation**
(i)	cout << "Hello, " << "world!";		

(ii)	cout << "This is a backslash: \\";		
(iii)	cout << “I am a computer geek, \rits a \blie.”		
(iv)	cout <<"a"<<"\t"<<"b"<<"\t"<<"c"<<endl;		
(v)	cout << "Line 1\nLine 2\nLine 3";		

(vi)	int x = 5;
	int y = 10;
	cout << "The sum of " << x << " and " << y << " is " << x + y;		

Q2. Create a C++ program that asks the user to enter their name and age using cin. Then, display a greeting message using cout. Ensure the program handles different data types for user input, such as strings and integers.

Q3. Write a C++ program that calculates the area of a rectangle. Ask the user for the length and width of the rectangle, perform the calculation, and display the result using cout. Use appropriate arithmetic and assignment operators for this task.

Q4. Write a C++ program that calculates a person's Body Mass Index (BMI). The program should use cin to obtain the necessary inputs, calculate the BMI using proper operator precedence, and display the result using cout. The BMI is calculated using the following formula:

$$BMI = \frac{Weighs\ in\ Kilograms}{Height\ in\ Meters^2}$$

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ____________________ Date: __________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 4

Objective:

To synthesize the principles of decision making in C++ using if and nested if statements, analyze complex scenarios, and propose logical solutions, demonstrating a higher-level understanding of control flow and problem-solving.

Theory:

Decision making in C++ is a fundamental concept that empowers programmers to take control of the program's execution flow. In our previous laboratory sessions, we've primarily explored programs that execute sequentially from start to finish. However, in real-world scenarios, it is often essential to introduce conditional logic, enabling specific code segments to execute only when certain conditions are met. This capability to direct program flow and make decisions on code execution is invaluable in programming.
In programming, decision making revolves around evaluating expressions, whether logical or relational. The outcome of this evaluation falls into one of two categories: TRUE or FALSE. When the result is TRUE, a specified piece of code is executed, allowing the program to perform a specific action. In the case of a FALSE outcome, there are two potential avenues to explore. The program can either execute a different piece of code than that designated for the TRUE case or follow an alternative branch.

Operators for Decision Making

In C++, operators for decision making are essential for evaluating conditions and controlling the flow of a program based on those conditions. Decision-making operators are also known as *relational* (Table 4-1), *logical* (
Table 4-2) and *ternary* (
Table 4-3) operators. They are fundamental tools for comparing values, checking for equality, and combining conditions to ascertain whether a specific condition evaluates to true or false.

Table 4-1 Relational Operators for decision making in C++

Operators	Description	Example
==	Equal to	10 == 9 is False
!=	Not equal to	10 != 9 is True
<	Less than	10 < 9 is False
>	Greater than	10> 9 is Ture
<=	Less than or equal to	10 <= 9 is False
>=	Greater than or equal to	10 >= 9 is True

Code 4-01:

```
#include <iostream>
using namespace std;
int main() {
   int a = 42, b = 19;  float radius = 5.0, circumference = 31.4;
   string city = "Paris", country = "France";
    cout << "Is a greater than b? " << (a > b) << " (1=true, 0=false)" << endl;
   cout << "Is the radius equal to the circumference?"<< (radius == circumference)<< endl;
   cout << "Is the city the same as the country? " << (city == country) << endl;
   cout << "Is a equal to b? " << (a==b) << endl;
   return 0;}
```

Synthesize, analyze, and explain your understanding of the Code 4-01.

Table 4-2 Logical Operators for decision making in C++

Operators	C++ Symbol	Example				
		A	**B**	**A&&B**	**A||B**	**!(A&&B)**
AND	&&	0	0	0	0	1
		0	1	0	1	1
OR	||	1	0	0	1	1
		1	1	1	1	0
NOT	!	**A**	**!A**	**A**	**!A**	
		0	1	1	0	

Code 4-02:

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

int main() {
   bool p = true, q = true, r = false, result;

   result = (p || q);
   cout << "p AND q = " << (p && q) << endl;
   cout << "p OR q = " << result << endl;
   cout << "NOT p = " << (!p) << endl;
   cout << "NOT q = " << (!q) << endl;
   cout << "NOT r = " << (!r) << endl;

   return 0;
}
```

Synthesize, analyze, and explain your understanding of the Code 4-02.

Table 4-3 Ternary Operator for decision making in C++

Operators	Description	Example
?:	Conditional Operator	int max = (x > y) ? x : y;

Code 4-03:

```
#include <iostream>
using namespace std;
int main() {
 int x,y; cout<<"Enter Values of x and y: "; cin>>x>>y;
 int result = (x > y) ? x : y;
 cout << "The greater of x and y is: " << result << endl;
 return 0;}
```

Synthesize, analyze, and explain your understanding of the Code 4-03.

The if() statement in C++

The 'if' statement is a powerful construct that forms the cornerstone of decision-making in C++. It serves as a gatekeeper, determining whether a program should enter a particular section of code based on the truth or falsehood of a given condition as shown in Figure 4-1. Among its diverse applications, one of the most notable functions of the 'if' statement is enabling programs to respond to user input. For instance, an 'if' statement can assess a user-entered password, granting or denying access to the program based on the evaluation of the condition.

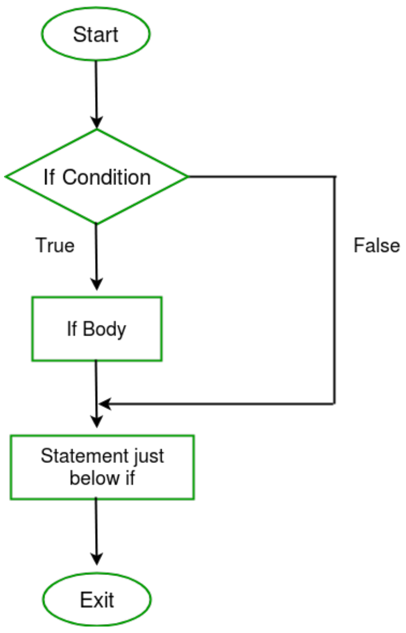


Figure 4-1 Flowchart of an IF Statement

Syntax:

```
if (condition) {
  // code to execute if condition is true
}
```

The condition can be any Boolean expression. If the condition evaluates to true, then the code inside the curly braces is executed. If the condition evaluates to false, then the code inside the curly braces is skipped.

Code 4-04:

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

int main() {
  int x;    cout<<”Enter x: “;    cin>>x;
  if (x > 10) {
    cout << "x is greater than 10" << endl;
  }
  return 0;
}
```

Synthesize, analyze, and explain your understanding of the Code 4-04.

Nested if() statements in C++

A nested IF statement in C++ is a programming construct that involves placing an IF statement within another IF statement. This allows for more complex decision-making processes, enabling the program to handle multiple conditions simultaneously. The nested IF structure typically involves checking for multiple conditions and executing specific code blocks based on the outcomes of these checks.

Syntax:

```
if (condition1)
   {
    // Code to execute if condition1 is true
   if (condition2)
      { //Code to execute if condition2 and condition 1 is true }
    }
```

Exercises:

Q1. Develop a C++ program to determine whether an inputted alphabet is uppercase or lowercase. (Hint: The ASCII code for A is 65.)

Q2. Write a C++ program that determines whether an inputted alphabet is a vowel or a consonant. The program should handle both uppercase and lowercase letters.

NED University of Engineering & Technology

Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ______________________ Date: ____________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 5

Objective:

To analyze and evaluate the functioning of conditional statements, such as if-else, else-if, nested if-else, and Switch-Case within the context of C++ programming, and to discern their respective applications and implications in program flow control.

Theory:

The if-else statement is a control structure that enables you to make decisions in your program based on the evaluation of a condition. It comprises two blocks of code: one block is executed if the condition is true, and the other block is executed if the condition is false, as illustrated in Figure 5-1. In contrast, as indicated in Table 5-1, the if statement alone allows the execution of a single block of code only if the specified condition evaluates to true.

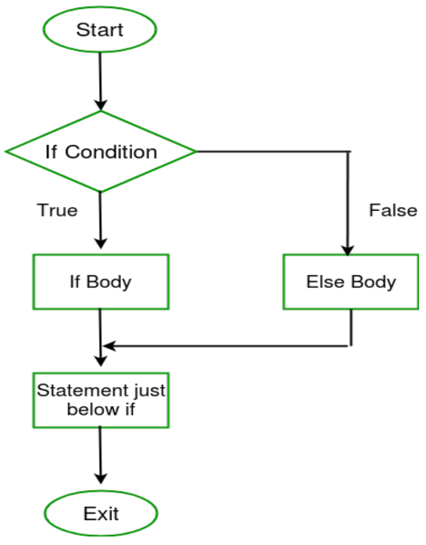


Figure 5-1 Flowchart of an if-else statement

Syntax

```
if (condition) {
   // Code to be executed if the condition is true
} else {
   // Code to be executed if the condition is false
}
```

Table 5-1 Comparison of if and if-else Statements

If statement	If-else statement
Executes a block of code only if the condition is true.	Ensures that one of the specified blocks of code will be executed.
No alternative action is specified for the false case.	Provides an alternative block of code to be executed if the condition is false.

Code 5-01:

```
#include <iostream>
using namespace std;
int main() {
 int x;           cout<<"Enter x: ";              cin>>x;
  // if-else statement
  if (x > 5) {
    cout << "x is greater than 5." << endl;  }
  else {
    cout << "x is not greater than 5." << endl;  }
  return 0;}
```

Synthesize, analyze, and explain your understanding of the Code 5-01.

The if-else if statement:

In C++, the if-else if (often written as if-elseif) statement is an extension of the if and if-else statements, providing a way to handle multiple conditions in a more structured manner. The if-elseif statement allows you to evaluate multiple conditions sequentially. If the first if condition is true, the corresponding block of code is executed, and the rest of the conditions are skipped. If the first if condition is false, the program checks the next elseif condition. This process continues until a true condition is found, and its associated block of code is executed.

Syntax

```
if (condition1) {
  // Code to be executed if condition1 is true
} else if (condition2) {
  // Code to be executed if condition2 is true
} else {
  // Code to be executed if none of the conditions are true
}
```

Code 5-02:

```
#include <iostream>
using namespace std;
int main() {
   char operation;    double num1, num2;
   // Get user input
   cout << "Enter an operation (+, -, *, /): ";        cin >> operation;
   cout << "Enter two numbers: ";          cin >> num1 >> num2;
   // Perform calculations based on the user's choice
   if (operation == '+') {
      cout << num1 << " + " << num2 << " = " << num1 + num2 << endl; }
   else if (operation == '-') {
      cout << num1 << " - " << num2 << " = " << num1 - num2 << endl; }
   else if (operation == '*') {
      cout << num1 << " * " << num2 << " = " << num1 * num2 << endl; }
   else if (operation == '/') {
         // Check for division by zero
         if (num2 != 0) {
              cout << num1 << " / " << num2 << " = " << num1 / num2 << endl; }
          else {
               cout << "Error: Division by zero is not allowed." << endl;  }
   } else {
      cout << "Invalid operation. Please enter +, -, *, or /." << endl;    }
   return 0; }
```

Synthesize, analyze, and explain your understanding of the Code 5-02.

Nested if-else if statement:

A nested if-else if structure involves placing one or more if-else if statements inside the body of another if or else block. This arrangement allows for the evaluation of multiple conditions in a hierarchical manner, with each level of the nested structure introducing additional conditions to check.

Execution:

1. Evaluation of Outer Condition:
   - The outermost "if" condition is evaluated first.
   - If it is true, the corresponding block of code is executed.
   - If it is false, the control may move to the else if conditions (if any) associated with the outer "if".
2. Evaluation of Inner Conditions:
   - If an "else if" condition is true, the corresponding block of code is executed, and the rest of the inner "else if" conditions are skipped.
   - If none of the "else if" conditions are true, the "else" block (if present) is executed.
3. Nested Layers:
   - Additional levels of nesting can be introduced by placing 'if-else if' structures within the blocks of outer 'if' or 'else' statements, and the program will execute accordingly.

Code 5-03:

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

int main() {
int x;          cout<<"Enter x: ";          cin>>x;
  if (x > 0) {
    cout << "x is positive." << endl;
    if (x % 2 == 0) {
       cout << "x is even." << endl;  }
    else {
      cout << "x is odd." << endl;    }

  } else if (x < 0) {
    cout << "x is negative." << endl;  }

  else {
    cout << "x is zero." << endl;   }
  return 0;
}
```

Synthesize, analyze, and explain your understanding of the Code 5-03.

Switch-Case statement:

The switch-case statement is a control flow statement that allows a program to execute different blocks of code based on the value of an expression. It is a convenient way to write a series of if-else statements when you have a single variable whose value needs to be tested against multiple possible values.

Syntax

```
switch (expression)
{
  case value1:
    // code to be executed if expression == value1
    break;
  case value2:
    // code to be executed if expression == value2
    break;
  // more cases can be added as needed
  default:
    // code to be executed if expression doesn't match any case
}
```

Code 5-04:

```
#include <iostream>
using namespace std;
int main() {
   char unit;
   cout << "Select voltage unit: (V)olts, (m)illivolts, (u)microvolts: ";    cin >> unit;
   switch (unit) {
      case 'V':
      case 'v':
          cout << "Voltage unit selected: Volts" << endl;          break;

      case 'M':
      case 'm':
          cout << "Voltage unit selected: Millivolts" << endl;          break;

      case 'U':
      case 'u':
          cout << "Voltage unit selected: Microvolts" << endl;          break;

      default:
         cout << "Invalid selection!" << endl;    }
   return 0;}
```

Synthesize, analyze, and explain your understanding of the Code 5-04.

Exercises:

Q1. Write a C++ program that takes the input voltage (V) from a user and determines whether it is within the acceptable range for a digital logic HIGH signal. If the voltage is greater than or equal to 2.0 volts, print "Logic HIGH." Otherwise, print "Logic LOW." (Method: if-else)

Q2. Create a C++ program for a simple temperature control system using if-else if statements. Take the temperature input from a sensor, and based on the temperature range, provide feedback on the heating status. If the temperature is below 20 degrees Celsius, print "Heating OFF." If it's between 20 and 25 degrees Celsius, print "Heating ON, maintaining temperature." If it's above 25 degrees Celsius, print "Heating ON, cooling down."

Q3. Develop a C++ program to analyze a digital signal using nested if-elseif statements. Take input for signal frequency and duty cycle. Based on the values, classify the signal as follows:

- If frequency is less than 1 kHz, check duty cycle:
  - If duty cycle is less than 50%, print "Low-frequency, low-duty cycle signal."
  - If duty cycle is equal to or greater than 50%, print "Low-frequency, high-duty cycle signal."
- If frequency is 1 kHz or greater, check duty cycle:
  - If duty cycle is less than 50%, print "High-frequency, low-duty cycle signal."
  - If duty cycle is equal to or greater than 50%, print "High-frequency, high-duty cycle signal."

Q4. You're tasked with designing a control system for a robotic arm with dynamic movement options based on user input. The choices include circular, square, and rectangular trajectories. Implement a switch-case structure to handle trajectory selection efficiently. For circular movement, prompt the user for the radius to calculate the area. In the case of a square trajectory, request the side length for area calculation. For a rectangular trajectory, collect both length and width inputs to calculate the corresponding area.

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ____________________ Date: __________________

Software Use Rubric

Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 6

Objective:

To cultivate a comprehensive understanding of C++ loops by employing for loops and nested for loops in diverse scenarios and demonstrate proficiency in basic and advanced operations, comprehend multi-dimensional data processing, generate dynamic patterns, and optimize loop performance for enhanced code efficiency.

Theory:

In C++, loops are control structures used for executing a block of code repeatedly based on a specified condition. C++ has three main types of loops: the for loop, the while loop, and the do-while loop. These loop structures are fundamental in programming for handling repetitive tasks and iterating through data structures. It's important to define the loop conditions correctly to achieve the desired behavior and to prevent unintended issues, such as infinite loops.

for loop:

The for loop is a control flow statement designed to repeatedly execute a block of code for a specific number of iterations as shown in Figure 6-1 . It is particularly useful in situations where the number of iterations is known beforehand.

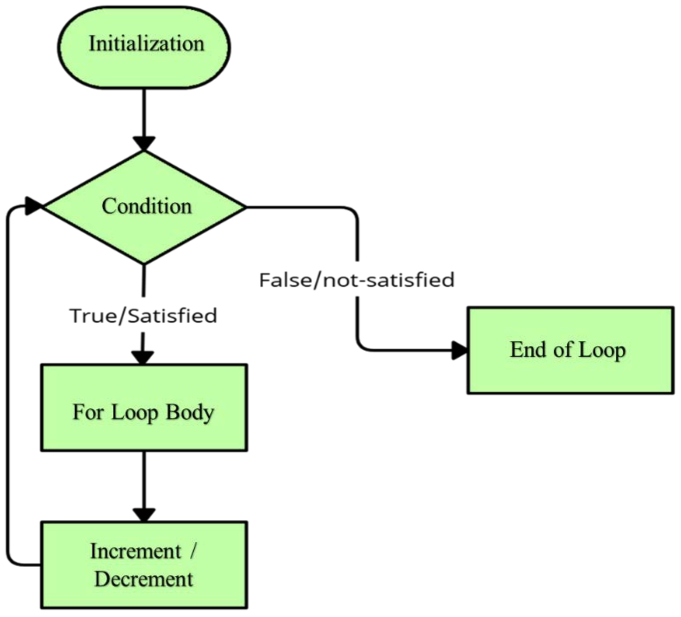


Figure 6-1 Sequential Flow Structure Illustrating the Iterative Process of a C++ for Loop

Syntax

```
for (initialization; condition; update) {
   // code to be executed
}
```

Code 6-01:

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

int main() {
int sum = 0;    // Initialize a variable to store the sum
// Use a for loop to calculate the sum of the first 5 natural numbers
   for (int i = 1; i <= 5; ++i) {
      sum += i; // Add the current number to the sum
   }
// Display the result
cout << "The sum of the first 5 natural numbers is: " << sum << endl;
return 0; }
```

Synthesize, analyze, and explain your understanding of the Code 6-01.

Nested For Loop:

A nested for loop is a loop structure placed inside another for loop. This implies that there is a loop inside the body of another loop, and the inner for loop is entirely contained within the body of the outer for loop. The inner loop will be executed multiple times for each iteration of the outer loop. Nested for loops are commonly employed for tasks involving 2D arrays, matrix operations, and pattern generation. Each iteration of the outer loop triggers a series of iterations of the inner loop, enabling more complex control flow and handling of repetitive tasks.

Syntax

```
for (initialization; condition; update) {
   // Outer loop body

   for (inner initialization; inner condition; inner update) {
      // Inner loop body
   }
}
```

Code 6-02:

```
#include <iostream>
Using namespace std;
int main() {
   for (int i = 0; i < 5; ++i) {       // Outer loop: controls rows

      for (int j = 0; j < 5; ++j) {         // Inner loop: controls columns
         cout << "* ";
      }
      cout << endl; // Move to the next line after each row
   }
return 0; }
```

Synthesize, analyze, and explain your understanding of the Code 6-02.

Code 6-03:

```
#include <iostream>
using namespace std;
int main() {
   for (int i = 1; i <= 5; ++i) {
         for (int j = 1; j <= 5; ++j) {
               cout << i * j << "\t";
          }
         cout << "\n";
   }
return 0; }
```

Synthesize, analyze, and explain your understanding of the Code 6-03.

Exercises:

Q1. Design a C++ program that calculates the total resistance of a series resistor network using for-loop. The user should be prompted to enter the number of resistors in the network and their individual resistance values. The program should then compute and display the total resistance of the network.

Q2. Assume you are tasked with designing the real-time clock interface for an embedded system using a microcontroller. Create a C++ program that simulates the behavior of a digital clock. The clock should display hours, minutes, and seconds and simulate real-time updates. Use the windows.h header for a one-second delay between each iteration.
Requirements:

- Implement a real-time clock interface with hours (24-hour format), minutes, and seconds.
- Utilize the windows.h header for a one-second delay between each clock update.
- Ensure that the displayed time is formatted as HH:MM:SS. (You may use setfill() and setw())
- Simulate the clock for a specific duration input by the user in seconds.

Q3. In the field of Electronic Engineering, designing traffic signal controllers is a critical aspect of traffic management systems. Your task is to create a C++ program that simulates a basic traffic signal controller. This simulation will help in understanding the timing and coordination of signal phases.

Write a C++ program that simulates a traffic signal with three phases: Red, Yellow, and Green. The program should take two inputs:

- The total number of simulation cycles to run.
- The duration (in seconds) for each phase of the traffic signal.
- Utilize the Sleep function from windows.h for introducing delays in each phase.

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ______________________ Date: ____________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 7

Objective:

To develop a thorough understanding of while and do-while loops in C++ by adeptly applying these constructs in various contexts. Demonstrate skill in executing fundamental and advanced operations, grasp intricate details in iterative data processing, and optimize loop performance for enhanced code efficiency.

Theory:

In C++, while and do-while are loop constructs that allow you to repeatedly execute a block of code as long as a certain condition is true. These loops are used for repetitive tasks where you want to perform the same set of instructions multiple times.

while Loop:

The while loop is a fundamental control structure in C++ that facilitates the repetitive execution of a block of code. It is categorized as a pre-test loop, as it evaluates a condition before entering the loop body, as shown in Figure 7-1. This implies that if the initial condition is false, the loop will not execute at all.

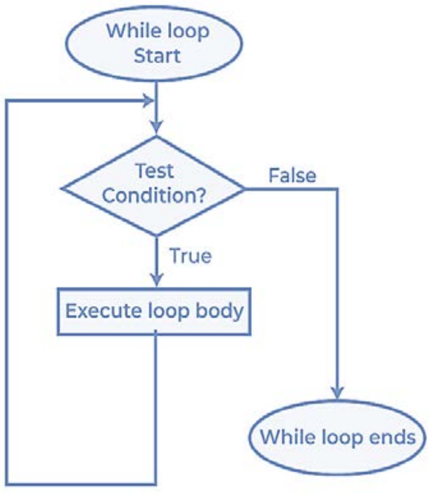


Figure 7-1 Flowchart illustrating the execution of a 'while' loop in C++.

Syntax

```
while (condition) {
  // Code to be executed as long as the condition remains true.
  // Increment or update variables to eventually make the condition false.
}
```

Code 7-01:

```
#include <iostream>
using namespace std;
int main()
{
  int i = 1;
  while (i <= 5) {
    cout << i << " ";
    i++;
  }
  return 0;
}
```

Synthesize, analyze, and explain your understanding of the Code 7-01.

Code 7-02:

```
#include <iostream>
using namespace std;
int main() {
   int i = 2; // Start with the first even number
   while (i <= 10) {
      cout << i << " ";
      i += 2; // Increment by 2 to get the next even number
   }
   return 0;
}
```

Synthesize, analyze, and explain your understanding of the Code 7-02.

do-while Loop:

In contrast with the while loop, the do-while loop is another looping construct in C++ that functions as a post-test loop. It ensures that the loop body is executed at least once before checking the loop condition, as shown in Figure 7-2 . This is particularly useful when you want to guarantee the execution of certain code before validating the loop condition.

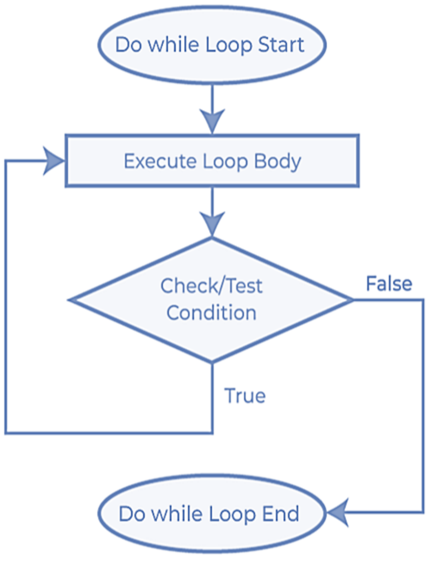


Figure 7-2 Flowchart illustrating the execution of a 'do-while' loop in C++.

Syntax

```
do {
  // Code to be executed
  // Increment or update variables to eventually make the condition false
    }
while (condition);
```

Code 7-03:

```
#include <iostream>
using namespace std;
int main() {
  int i = 6;
  do {
    cout << i << " ";
    i++;
  } while (i <= 5);
  return 0;
}
```

Synthesize, analyze, and explain your understanding of the Code 7-03.

Code 7-04:

```
#include <iostream>
using namespace std;
int main() {
   int number;
   do {
      cout << "Enter a positive number: ";
      cin >> number;
      if (number <= 0) {
         cout << "Invalid input. Please enter a positive number." << endl;
      }
   } while (number <= 0);

   cout << "You entered: " << number << endl;
   return 0;}
```

- **Synthesize, analyze, and explain your understanding of the Code 7-04.**

Best Practices:

- Ensure that the loop condition will eventually become false to avoid infinite loops.
- Initialize and update loop control variables within the loop body as needed.
- Use while loops when the number of iterations is uncertain, and use do-while loops when you need the loop body to execute at least once.

Exercises:

Q1. Recall the number guessing game from Lab session 02. Utilizing the concepts and skills acquired thus far, develop two distinct programs for the number guessing game using a while loop and a do-while loop, respectively. Consult Figure 2-11 to comprehend the execution flow and program requirements. Additionally, analyze both programs and provide justification for the choice of the loop construct that is better suited for this game and elucidate the reasons behind your preference.

Q2. The standard resistor values are organized into a set of series known as the E-Series. These values are spaced to ensure that the top of the tolerance band of one value and the bottom of the tolerance band of the next one do not overlap. Your task is to create a program for calculating and displaying the first few resistor values in the E-Series. The program should allow the user to specify the number of resistor values they want to generate using a while loop. Ensure that the user enters a valid number of terms (at least 2) and output the resistor values in the E-Series. Note: The E-Series consists of resistor values that follow a geometric progression. The program should calculate these values based on the common ratio between consecutive E-Series values.

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ____________________ Date: __________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 8

Objective:

To gain a comprehensive understanding of structures in C++, covering their definition, declaration, and implementation. Apply this knowledge to create and manipulate structured data, demonstrating proficiency in solving programming challenges through the effective utilization of C++ structures.

Theory:

In C++, a structure represents a user-defined data type that facilitates the grouping of diverse variables under a single name. Unlike basic data types such as int or float, which imply singular values, structures empower the creation of intricate data structures encapsulating various pieces of information.

Syntax

```
//Definition
struct Student {
    int rollNumber;
   char name[50];
    float marks;  };

//Declaration
    Student student1, student2;

//Member Access
   student1.rollNumber = 101;
   strcpy(student1.name, "Danish");
   student1.marks = 88.2;

 // Printing structure members
    cout << "Roll Number: " << student1.rollNumber << endl;
    cout << "Name: " << student1.name << endl;
    cout << "Marks: " << student1.marks << endl;
```

- A structure is defined using the struct keyword, followed by a block of variables known as members.
- Members can be of different data types, including other structures.
- After defining a structure, variables of that type can be declared.
- Each variable becomes an instance of the structure and contains its own set of member variables.
- Individual members of a structure can be accessed using the dot (.) operator.

Code 8-01:

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

// Definition of the structure
struct Person {
   string name;    int age;    float height;    };
int main() {
   Person person1, person2;
   // Initialization of structure variables
   person1.name = "Alice";     person1.age = 25;    person1.height = 1.75;
   person2.name = "Bob";       person2.age = 30;    person2.height = 1.80;
   // Displaying information using namespace std
   cout << "Person 1: " << person1.name << ", Age: " << person1.age << ", Height: " <<
    person1.height << " meters\n";
   cout << "Person 2: " << person2.name << ", Age: " << person2.age << ", Height: " <<
    person2.height << " meters\n";
   return 0; }
```

Synthesize, analyze, and explain your understanding of the Code 8-01.

Nested Structures:

Structures can be nested within other structures, enabling the representation of more complex relationships and hierarchies.

Syntax

```
struct Address {
   char street[50];    char city[50];   };

struct Employee {
   int empID;    char name[50];    Address empAddress;   };
```

Code 8-02:

```
#include <iostream>
using namespace std;
// Definition of the Point structure
struct Point {
   double x;    double y;    };
// Definition of the Rectangle structure with two nested Point structures
struct Rectangle {
   Point topLeft;     Point bottomRight;  };
int main() {
   // Declaration and initialization of the nested structures
   Point p1 = {2.5, 4.0};      Point p2 = {6.0, 1.0};
   // Declaration and initialization of the main structure
   Rectangle rect = {p1, p2};
   // Printing information using nested structures
   cout << "Rectangle Coordinates:" << endl;
   cout << "Top Left: (" << rect.topLeft.x << ", " << rect.topLeft.y << ")" << endl;
   cout << "Bottom Right: (" << rect.bottomRight.x << ", " << rect.bottomRight.y << ")" ;
   return 0;    }
```

Synthesize, analyze, and explain your understanding of the Code 8-02.

Structures provide a powerful and flexible way to organize and manage data in C++. They are particularly useful when dealing with entities that have multiple attributes or when creating more sophisticated data structures for practical programming tasks.

<u>Exercises:</u>

Q1. Develop a C++ program that manages information about resistors in a structure, allows users to input resistor details, and provides options for relevant calculations. The primary focus is on code quality, efficiency, and user-friendly interactions. The program should adhere to best coding practices, handle inputs in proper units, and utilize appropriate data types.

Requirements:

1. **Resistor Structure:**
   - Implement a Resistor structure with fields for uniqueID, resistance, powerRating, and tolerance.
   - Ensure the use of appropriate data types for each field.

2. **User Input:**
   - Prompt the user to enter information for a resistor, including ID, resistance, power rating, and tolerance in suitable units.
   - Validate user inputs to ensure proper units and reasonable values. For example, resistance and power rating should be positive, and tolerance should be within the valid percentage range.

3. **Calculation Options:**
   - Implement two calculation options for the user to choose from:
     - **Option 1:** Calculate Actual Resistance within Tolerance Range
       - Calculate and display the minimum and maximum resistance within the tolerance range.
     - **Option 2**: Calculate Current using Power and Resistance.
       - Calculate and display the rated/maximum current using the formula:

$$\boldsymbol{I} = \sqrt{\frac{\boldsymbol{P}}{\boldsymbol{R}}}$$

$$\boldsymbol{where, P\ is\ the\ Power\ Rating\ (watts)\ of\ resistor\ R\ (ohms).}$$

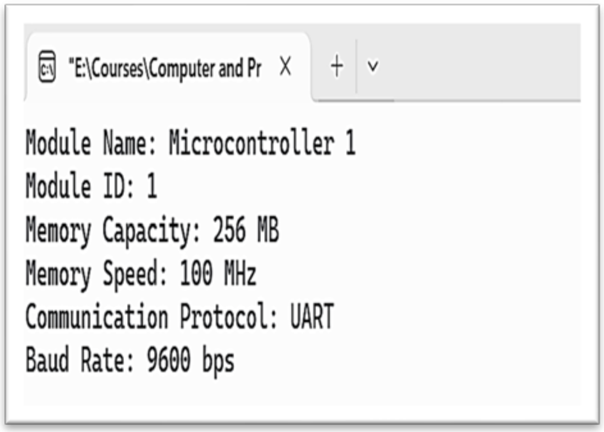
4. **Display Output:**
   - Present the entered resistor details, including the unique ID, resistance, power rating, and tolerance, in a clear and organized manner.
   - Display the result of the calculation opted by the user in the program in a clear and formatted manner.

5. **Code Quality and Efficiency:**
   - Use meaningful variable names and maintain a well-organized code structure.
   - Minimize redundancy and promote code reusability.
   - Properly handle invalid inputs, providing meaningful error messages.
   - Utilize appropriate data types for variables and ensure proper unit consistency.

Q2. Develop a C++ program to manage information about microcontroller modules within an electronic system, utilizing nested structures to represent specific attributes. The program aims to enhance understanding and proficiency in working with nested structures.

Requirements:

1. **Microcontroller Module Structure:**
   - Create a structure named MicrocontrollerModule with the following attributes:
     - Module name (string)
     - Module ID (integer)
     - Module type (character, 'M' for microcontroller)
     - Nested structure for Memory:
       - Capacity (integer, in megabytes)
       - Speed (integer, in megahertz)
     - Nested structure for Communication Interface:
       - Protocol (string)
       - Baud rate (integer, in bits per second)



```
Module Name: Microcontroller 1
Module ID: 1
Memory Capacity: 256 MB
Memory Speed: 100 MHz
Communication Protocol: UART
Baud Rate: 9600 bps
```

2. **User Input:**
   - Prompt the user to input information for a microcontroller module, including the module name, ID, memory details (capacity and speed), and communication interface details (protocol and baud rate).
   - Validate user inputs to ensure proper data types and reasonable values.

3. **Display Information:**
   - Display the entered information for the microcontroller module in a clear and formatted manner.

4. **Nested Structure Usage:**
   - Emphasize the use of nested structures to represent specific attributes such as memory and communication interface within the main MicrocontrollerModule structure.

5. **Documentation:**
   - Include comments in the code to explain the purpose of the program, highlight the use of nested structures, and provide explanations for complex logic or decisions.

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ______________________ Date: ____________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 9

Objective:

To demonstrate the principles and execution of functions in C++ and create user-defined functions, showcasing an advanced application of programming concepts.

Theory:

In C++, a function is a self-contained block of code designed to perform a specific task. Functions enhance code modularity, reusability, and readability. They consist of a declaration and a definition. Declarations specify the function's prototype, while definitions contain the actual code.

- Modularity: Breaking down a program into smaller, manageable functions.
- Reusability: Functions can be reused in different parts of the program.
- Readability: Functions make code more organized and easier to understand.

The functions can be widely divided into two categories as shown in Figure 9-1.

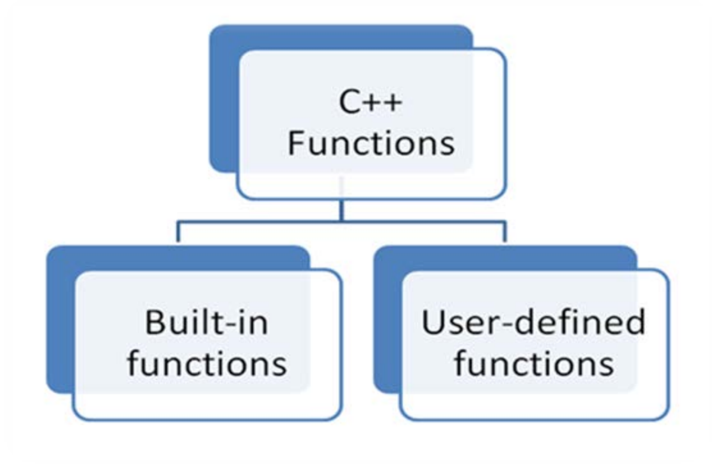


Figure 9-1 Types of Functions

Built-in functions are predefined functions provided by the programming language or its standard libraries to perform common operations. They are ready-made functions that can be directly used without the need for explicit implementation, providing essential functionalities and enhancing code efficiency.

For example:

- Input/Output Functions: cin, cout, getline().
- Math Functions: sqrt(), pow(), abs().
- String Functions: strlen(), strcpy(), strcmp().

A **user-defined function** is a custom, programmer-created function in a programming language, allowing the encapsulation of specific logic or tasks into a modular and reusable code block within a program.

Syntax

```
// Function declaration
        int addNumbers(int a, int b);

// Function definition
        int addNumbers(int a, int b) {
        return a + b;
}

// Function call
        int result = addNumbers(5, 7);
```

Parameters:

Parameters are variables or values that are used to pass information into a function. They allow a function to accept input, perform operations, and produce output based on the provided values. Parameters are declared in the function's parameter list within parentheses following the function name.

Return Type:

The return type of a function specifies the data type of the value that the function will provide as output. It is declared before the function name in the function declaration. Functions in C++ can return values of various types, including integers, floating-point numbers, characters, and even custom data types.

Function Call:

A function call is the act of invoking or executing a function. It involves specifying the function name, providing the necessary arguments (values or expressions) in the parentheses, and, in the case of functions with a return type, capturing or utilizing the returned value.

Code 9-01:

```
#include <iostream>
using namespace std;
void displayMessage(); // Function declaration
int main() {
  displayMessage();    // Function call
  return 0; }
// Function definition
void displayMessage() {
  cout << "Hello, this is a simple function!" << endl;  }
```

Synthesize, analyze, and explain your understanding of the Code 9-01.

<u>Code 9-02:</u>

```
#include <iostream>
   using namespace std;
// Function to calculate the average of two integers and return a float
float calculateAverage(int num1, int num2) {
   return (float)(num1 + num2) / 2.0;    }

int main() {
   int firstNumber, secondNumber;
   cout << "Enter the first integer: ";
   cin >> firstNumber;
   cout << "Enter the second integer: ";
   cin >> secondNumber;

   // Call the function and display the result
   float average = calculateAverage(firstNumber, secondNumber);
   cout << "The average of " << firstNumber << " and " << secondNumber << " is: " <<
average << endl;

   return 0;
}
```

Synthesize, analyze, and explain your understanding of the Code 9-02.

Recursive Functions:

Recursion is a technique where a function calls itself to solve a smaller instance of the same problem. It involves a base case to prevent infinite recursion.

Code 9-03:

```
#include <iostream>
 using namespace std;
// Recursive function for factorial
int factorial(int n) {
   if (n == 0 || n == 1) {
      return 1;   } // Base case
else {
      return n * factorial(n - 1); // Recursive case   }
             }
int main() {
    int num;
   cout << "Enter a number to calculate its factorial: ";
   cin >> num;

   if (num < 0) {
      cout << "Factorial is undefined for negative numbers." << endl;
   } else {
      int result = factorial(num);
      cout << "Factorial of " << num << " is: " << result << endl;   }
   return 0;}
```

- **Synthesize, analyze, and explain your understanding of the Code 9-03.**

Best Practices:

- Create small, focused functions for improved readability and reusability.
- Use descriptive names to convey the purpose of each function.
- Minimize parameters, use meaningful names, and explicitly define return types.
- Avoid use of global variables and pass variables as parameters to limit unintended side effects.

Exercises:

Q1. Implement a C++ program to identify the smallest inductance value among three inductors (L1, L2, L3) entered by the user. The program prompts the user to input the inductance values, calculates the minimum inductance using a user-defined function, and then displays the result to the user.

Q2. Develop a C++ program to compute the equivalent capacitance for capacitors organized in both series and parallel configurations. The program should take three capacitance values as input from the user. Additionally, incorporate two separate user-defined functions to efficiently calculate the equivalent capacitance for both series and parallel connections.

Requirements:

1. **User Input:**
   - Implement a user-friendly interface to collect three capacitance values from the user.
   - Ensure proper validation and handling of input errors.

2. **User-Defined Functions:**
   - Create a function that should take three capacitance values as parameters and return the equivalent capacitance for capacitors in series.

- Additionally, create another function that takes the same three capacitance values as parameters and returns the equivalent capacitance for capacitors in parallel.

3. **Output:**
   - Display the calculated equivalent capacitance for both series and parallel configurations.

4. **Testing:**
   - Test the program with different sets of capacitance values to verify its correctness.

5. **Efficiency and Quality:**
   - Implement code that adheres to good coding practices, including meaningful variable names, proper indentation, and comments where necessary.
   - Ensure that the functions are efficient and correctly calculate the equivalent capacitance based on the series and parallel configurations.

Q3. Utilizing the user-defined functions created in the previous question, implement a C++ program to calculate the equivalent capacitance for a circuit configuration as depicted in Figure 9-2.

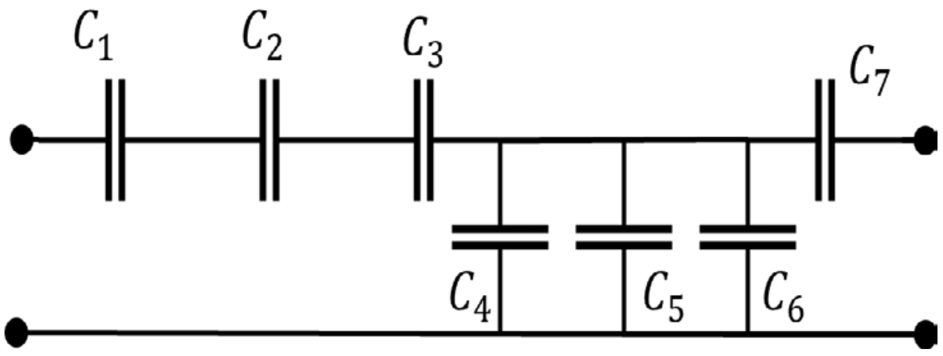


Figure 9-2 Capacitance Arrangement

In the circuit:

- Capacitors C_1, C_2 and C_3 are arranged in series.
- Capacitors C_4, C_5 and C_6 are arranged in parallel.
- The equivalent capacitance of the series combination and the parallel combination is in series with C_7.

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ______________________ Date: ____________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 10

Objective:

To comprehend the concepts of arrays and strings in C++, including their implementation and manipulation, and to apply these acquired skills in addressing a variety of practical scenarios.

Theory:

Arrays are fundamental data structures in programming that facilitate the storage of multiple values of the same data type under a single variable name. They provide an efficient way to organize and access elements in memory. In C++, arrays are zero-indexed.

- Arrays provide a straightforward way to access elements sequentially using index values. This allows for efficient iteration through all elements, making them suitable for tasks that involve processing each element in a sequence.
- Arrays allocate a contiguous block of memory to store elements of the same data type. This contiguous allocation reduces memory fragmentation and allows for efficient use of memory, especially when dealing with large datasets.
- Arrays are well-suited for searching and sorting algorithms.
- Arrays can be extended to multiple dimensions, such as 2D arrays or matrices. This feature is beneficial for representing tabular data, images, and other structures where a two-dimensional layout is more natural.

Syntax:

```
// Array declaration
 int numbers[3]; // Declares an integer array with a capacity for 3 elements.

// Array initialization
numbers[0] = 10;  numbers[1] = 20;   numbers[2] = 30;

// Array declaration and Initialization
int numbers[ ] = {10, 20, 30, 40, 50}; // Compiler infers the size.

// Accessing Elements
int x = numbers[2]; // Accesses the third element (30) and assigns it to x.

//Iterating Through an Array
for (int i = 0; i < arraySize; ++i) {
    // Access and process each element, e.g., numbers[i]
     }
```

Code 10-01:

```
#include <iostream>
using namespace std;
int main() {
  const int arraySize = 5; // Size of the array
  int numbers[arraySize];  // Declaration of an integer array
  // Input phase: Reading numbers from the user
  cout << "Enter " << arraySize << " numbers:\n";
  for (int i = 0; i < arraySize; ++i) {
      cout << "Number " << i + 1 << ": ";
      cin >> numbers[i];     }
  // Processing phase: Calculating sum and average
  int sum = 0;
  for (int i = 0; i < arraySize; ++i) {
    sum += numbers[i];   }
  double average = sum / arraySize;
  // Output phase: Displaying results
  cout << "\nSum: " << sum << "\n";   cout << "Average: " << average << "\n";
  return 0; }
```

Synthesize, analyze, and explain your understanding of the Code 10-01.

Multidimensional Array:

A multidimensional array in C++ refers to an array that incorporates more than one dimension. The most common type of multidimensional array is a two-dimensional array, but arrays with three or more dimensions can also be defined in C++. A two-dimensional array in C++ is essentially an array of arrays. It can be visualized as a table or grid with rows and columns.

Syntax:

```
// Declaration and Initialization of a 2D Array with 3 rows and 4 columns
int matrix[3][4] = {
   {1, 2, 3, 4},
   {5, 6, 7, 8},
   {9, 10, 11, 12}
 };
```

Code 10-02:

```
#include <iostream>
using namespace std;
int main() {
   // Declaration and Initialization of a 2D Array
   int matrix[3][4] = {
      {1, 2, 3, 4},       {5, 6, 7, 8},       {9, 10, 11, 12}    };

  // Printing the 2D Array
   cout << "2D Array Contents:\n";
   for (int i = 0; i < 3; ++i) {
         for (int j = 0; j < 4; ++j) {
               cout << matrix[i][j] << "\t";       }
      cout << "\n";    }

 // Accessing Individual Elements
   int element = matrix[1][2];  // Accessing element in the first row, second column
   cout << "\nElement at matrix[1][2]: " << element << "\n";
   return 0;   }
```

Synthesize, analyze, and explain your understanding of the Code 10-02.

Strings:

In C++, strings are sequences of characters represented by the ‘string’ class. Unlike arrays of characters, C++ strings offer dynamic memory management and various member functions that facilitate string manipulation.

Syntax:

```
// Declaration and Initialization of Strings
   string greeting = "Hello, World!";
```

C++ offers a comprehensive set of member functions and algorithms in the <string> header, facilitating the manipulation and processing of strings. Code 10-03 demonstrates the utilization of various C++ string manipulation functions.

Code 10-03:

```
#include <iostream>
#include<string>
using namespace std;
int main() {
// length() / size(): Returns the number of characters in the string.
        string myString = "Hello, World!";
        int length = myString.length();  // or myString.size();
// append(): Appends a string or a portion of it to the end of another string.
        string greeting = "Hello";
        greeting.append(", World!");
// substr(): Returns a substring of the original string.
        string sentence = "The quick brown fox";
        string substring = sentence.substr(4, 5);  // Starting location: 4, length:5
// find(): Searches for a substring within a string and returns the position of the first
occurrence.
        string phrase = "The cat in the hat";
        int position = phrase.find("cat"); // you can also use size_t type instead of int
// compare(): Compares two strings lexicographically.
        string str1 = "apple";        string str2 = "banana";
        int result = str1.compare(str2);
// replace(): Replaces a portion of the string with another string.
        string sentence = "I like programming.";
        sentence.replace(7, 11, "coding");  // Replaces "programming" with "coding"
// erase(): Erases a portion of the string.
        string word = "apple";
        word.erase(3, 1);  // Erases the character at position 3
// empty(): Checks if the string is empty.
        string message = "Hello, World!";
        if (message.empty()) {
                // String is empty    }
// at(): Accesses the character at a specified position.
        string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        char letter = alphabet.at(4);  // Accesses the character 'E'

return 0;   }
```

Code 10-04:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
  string phrase = "The cat in the hat";
  int position = phrase.find("cat");
  if (position != -1) {
    cout << "Found at position: " << position << endl;   }
  else {
    cout << "Not found." << endl;   }
  return 0;  }
```

Synthesize, analyze, and explain your understanding of the Code 10-04.

Exercises:

Task: Universal Number System Conversion Utility for Computer Architecture

As part of an embedded system project, you are tasked with developing a Universal Number System Conversion Utility that incorporates a range of number systems crucial for low-level programming and computer architecture. The utility should handle conversions between binary, octal, decimal, and hexadecimal representations, providing a comprehensive tool for developers working close to the hardware.

Requirements:

1. **User Input:**
   - A number.
   - The base of the provided number.
   - The target base for conversion.

2. **User-Defined Functions:**
   - Decimal to any base function.
   - Binary to any base function.
   - Octal to any base function.
   - Hexadecimal to any base function.

3. **Hints:**
   - String Representation
   - ✓ The input binaryNumber is a string representing a binary number.
   - ✓ Characters in a string can be accessed using iteration.

   - Character to Numeric Value Conversion
   - ✓ (digit - '0') converts a character representing a digit to its numeric value.
   - ✓ ASCII values are used, where '0' to '9' have consecutive integer values.

   - Binary to Decimal Conversion
   - ✓ Use a loop to iterate through each binary digit.
   - ✓ Perform a left shift (decimalEquivalent * 2) for each binary digit.
   - ✓ Add the numeric value of the current binary digit to decimalEquivalent.

   - Conversion to Target Base
   - ✓ After binary to decimal conversion, use another function (decimalToBase) to convert to the target base.
   - ✓ Encourage a modular and reusable approach to number system conversions.

   - Use of Functions
   - ✓ Break down the problem into smaller functions for readability and reusability.
   - ✓ Functions like decimalToBase can handle common logic for conversion between decimal and other bases.

   - Testing and Debugging
   - ✓ Test the function with different binary numbers and target bases to ensure correctness.
   - ✓ Use debugging techniques to identify and fix any issues.

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. _____________________ Date: ___________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can’t identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can’t use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn’t handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor’s Signature with Date	

Lab Session 11

Objective:

To demonstrate effective utilization of pointers in tasks such as dynamic memory allocation, data manipulation, and code optimization for addressing and solving programming challenges.

Theory:

Pointers are variables that store the memory address of another variable. They provide a powerful mechanism for handling memory and manipulating data at a low level. Understanding pointers is essential for efficient memory management and working with complex data structures.

- **Dynamic Memory Allocation:** Pointers allow for dynamic memory allocation, enabling flexible and efficient use of memory at runtime.
- **Efficient Memory Usage:** Pointers reduce memory duplication by enabling indirect access to data, optimizing overall memory usage.
- **Passing Parameters by Reference:** Pointers facilitate passing parameters by reference to functions, allowing for modifications to the original data.
- **Dynamic Data Structures:** Pointers are crucial for implementing dynamic data structures like linked lists, trees, and graphs.
- **Array Manipulation:** Pointers simplify array manipulation, making it easier to iterate over elements, allocate memory, and pass arrays to functions.

Syntax:

```
int *ptr;  // declares a pointer to an integer
int x = 20;
int *ptr = &x;  // ptr now stores the address of variable x
int y = *ptr;  // y now holds the value stored at the address pointed by ptr
```

Code 11-01:

```
#include <iostream>
using namespace std;
int main() {
   int i = 5;
   int *ptr = &i;
   cout << "Value of i through pointer: " << *ptr << endl;
  *ptr = 10;
   cout << "Updated value of i: " << i << endl;
   return 0;
}
```

- **Synthesize, analyze, and explain your understanding of the Code 11-01.**

Pointer arithmetic is the manipulation of pointers using arithmetic operations. In C++, pointer arithmetic is particularly useful when working with arrays or dynamically allocated memory. The key operations involved in pointer arithmetic are addition (+), subtraction (-), increment (++), and decrement (--).

- Incrementing a pointer (ptr++) moves it to the next memory location of its type.
- Decrementing a pointer (ptr--) moves it to the previous memory location of its type.
- Adding an integer to a pointer (ptr + n) moves it forward by n times the size of its type.
- Subtracting an integer from a pointer (ptr - n) moves it backward by n times the size of its type.

Code 11-02:

```
#include <iostream>
using namespace std;
int main()  {
int a=25; float b=10.329; char c='k';
int* p1=&a; float* p2 = &b; char* p3 = &c;
cout<<"******** Pointer P1 ********"<<endl;
cout<<"Value at *P1: "<<*p1<<endl;      cout<<"Value in P1: "<<p1<<endl<<endl;
cout<<"******** Pointer P2 ********"<<endl;
cout<<"Value at *P2: "<<*p2<<endl;     cout<<"Value in P2: "<<p2<<endl<<endl;
cout<<"******** Pointer P3 ********"<<endl;
cout<<"Value at *P3: "<<*p3<<endl;
cout<<"Value in P3: "<<static_cast<void*>(p3)<<endl<<endl; //to resolve address
display issue due to compiler
*p2 = *p2 + *p1;
cout<<"******** UpdatedPointer P2 ********"<<endl;
cout<<"Value at *P2: "<<*p2<<endl;      cout<<"Value in P2: "<<p2<<endl;
return 0; }
```

Synthesize, analyze, and explain your understanding of the Code 11-02.

Passing pointers to functions and returning values through pointers are common practices in C++ and are used to manipulate data in a more efficient way compared to passing by value. When passing a pointer to a function, it allows the function to access and modify the data at the memory location pointed to by the pointer. This is particularly useful when avoiding making copies of large data structures or when a function needs to modify the original data. Pointers can also be used to return multiple values from a function. Instead of returning a single value, pointers are passed to the function to store the results. This is particularly useful when a function needs to return more than one value or when avoiding returning complex data structures by value.

Code 11-03:

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

void calculateValues(int num, int *square, int *cube) {
   *square = num * num;
   *cube = num * num * num;  }

int main() {
   int number = 3;
   int square, cube;
   calculateValues(number, &square, &cube);
   cout << "Square: " << square << endl;
   cout << "Cube: " << cube << endl;
   return 0;
}
```

- **Synthesize, analyze, and explain your understanding of the Code 11-03.**

Pointers in C++ offer powerful capabilities for efficient memory management and data manipulation. They enable dynamic memory allocation, facilitate the passing of data by reference, and provide a means to work with arrays and dynamic data structures. Function pointers contribute to code modularity, and pointer arithmetic allows for low-level memory manipulation. While pointers enhance flexibility, it's crucial to use them judiciously to avoid common pitfalls such as memory leaks and segmentation faults. Modern C++ introduces features like smart pointers, providing safer alternatives for memory management.

Exercises:

Q1. Design and implement a C++ program that uses pointers to process real-time sensor data from an embedded system. Create a function that calculates statistical parameters (mean, variance, etc.) of the sensor readings stored in an array. Use pointers to efficiently iterate through the data and perform the calculations.

Q2. Design a C++ program that calculates and displays properties of geometric shapes. The program should have functions to calculate properties for both a sphere and a cylinder. Implement the following tasks:

- Create a menu to allow the user to choose between calculating properties for a sphere or a cylinder.
- For a sphere, prompt the user for the radius and calculate/display the diameter, surface area, and volume.
- For a cylinder, prompt the user for the radius and height, then calculate/display the diameter, surface area, and volume.
- Use a switch-case structure to efficiently handle the user's choice and call the appropriate function for calculations.
- Ensure that the program handles invalid input gracefully, such as entering negative values for radius or height.

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. ______________________ Date: ____________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

Lab Session 12

Objective:

Synthesize object-oriented programming principles to design and implement classes, demonstrate inheritance relationships, and create effective object-oriented solutions for various real-life problems.

Theory:

Object-Oriented Programming (OOP) is a programming paradigm that organizes code structure around the concept of "objects." It's a way of designing and structuring software to make it more modular, flexible, and reusable. In OOP, everything is considered an object, which is an instance of a class. A class is a blueprint or template that defines the properties (attributes or fields) and behaviors (methods or functions) that the objects of the class will have.

- **Class:** A class is a user-defined blueprint or prototype from which objects are created. It defines the properties and behaviors common to all objects of that type.
- **Object:** An object is an instance of a class, created based on the class blueprint. Objects have both state (attributes or properties) and behavior (methods or functions).
- **Encapsulation:** Encapsulation is the bundling of data (attributes) and methods (functions) that operate on that data into a single unit (class). It helps in hiding the internal details and protecting the integrity of the data.

Code 12-01:

```
#include <iostream>
// Class declaration
class MyClass {
public: // Access specifier - members
are accessible outside the class
  // Attributes
  int myInteger;
  double myDouble;
  // Constructor
  MyClass(); // Constructor
declaration
  // Method
  void displayInfo();  };
// Constructor definition
MyClass::MyClass() {
    myInteger = 0;
  myDouble = 0.0; }
```

```
// Method definition
void MyClass::displayInfo() {
    std::cout << "Integer: " << myInteger
<< ", Double: " << myDouble << std::endl;
}

int main() {
  // Creating an object of MyClass
  MyClass myObject;

  // Accessing attributes and methods
  myObject.myInteger = 42;
  myObject.myDouble = 3.14;
  myObject.displayInfo();

  return 0;
}
```

- **Synthesize, analyze, and explain your understanding of the Code 12-01.**

- **Inheritance:** Inheritance is a mechanism that allows a new class (subclass or derived class) to inherit properties and behaviors from an existing class (superclass or base class). It promotes code reuse and establishes a relationship between classes.
- **Polymorphism:** Polymorphism allows objects of different classes to be treated as objects of a common base class. It enables the use of a single interface to represent different types of objects.
- **Abstraction:** Abstraction is the process of simplifying complex systems by modeling classes based on essential properties and behaviors. It hides the unnecessary details while exposing only what is necessary for the outside world.

Code 12-02:

```
#include <iostream>
// Base class
class Vehicle {
public:
   virtual void start() const {
      std::cout << "Vehicle starts." << std::endl;   }  };
// Derived class
class Car : public Vehicle {
public:
   void start(){
      std::cout << "Car engine starts." << std::endl;   }   };
int main() {
   Vehicle myVehicle;   Car Sedan;
   myVehicle.start();   Sedan.start();
   return 0; }
```

- **Synthesize, analyze, and explain your understanding of the Code 12-02.**

Object-Oriented Programming (OOP) offers several advantages that contribute to the development of robust, maintainable, and scalable software systems.

- Modularity: Encapsulation allows bundling of data and methods, enhancing code organization.
- Reusability: Objects and classes can be reused, reducing development time and effort.
- Flexibility and Extensibility: Code can be easily modified and extended without affecting the entire system.
- Maintainability: OOP code is more readable, making bug fixing and updates simpler.
- Readability: Classes and objects mirror real-world scenarios, enhancing code comprehension.
- Code Organization: OOP provides a natural structure for organizing code based on the problem domain.
- Improved Problem Solving: Breaking down complex problems into smaller parts promotes better problem-solving.
- Encapsulation and Information Hiding: Data and methods are encapsulated, controlling access and protecting data integrity.
- Inheritance and Polymorphism: Enable code reuse and flexibility in treating objects as instances of their base class.

Exercises:

Q1. Write a C++ program that defines a class named Student. The class should have member variables for the student's name, roll number, and marks. Implement member functions to set and display this information for a student. Create an object of the Student class, set its information, and then display it.

Q2. Design a simple class hierarchy for electronic components in C++. Implement a base class ElectronicComponent with a member variable type and two derived classes, Resistor and Capacitor. Create objects for a resistor and a capacitor, and display their types using the displayInfo() function.

NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-105 Computer and Programming

Laboratory Session No. _____________________ Date: ___________________

Software Use Rubric					
Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can’t identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently	Can’t use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab)	Doesn’t handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging)	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor’s Signature with Date	

Lab Session 13

OPEN-ENDED LAB

Department of Electronic Engineering

Course Title: Computer and Programming

Course Code: EL-105 **Year**: First Year **Semester**: Fall 2023

Designing and Implementation of Sentiment Analysis System in C++

This comprehensive problem is strategically designed to address the learning objectives associated with Course Learning Outcomes CLO2 (C4/PLO4) and CLO3 (C5/PLO6). It serves as a dual-purpose evaluation tool, seamlessly integrating the Complex Engineering Problem (CEP) and Open-Ended Lab (OEL) components.

For the CLO2 assessment, the emphasis lies in gauging students' analytical ability as they investigate the complexities of the problem. Their cognitive skills, honed throughout the course, will be put to the test as they analyse the problem and devise a detailed algorithm to address the complexities inherent in the CEP.

Conversely, CLO3 evaluation focuses on the practical application of programming concepts. Students will be tasked with synthesizing their understanding of programming principles, particularly in C++, and translating them into functional code. The focus is on their ability to bridge the gap between theoretical knowledge and practical implementation, showcasing a robust grasp of programming concepts.

Distinct rubrics tailored for CEP and OEL assessments will be employed, ensuring a nuanced evaluation that captures the essence of both problem-solving and practical programming proficiency. This multifaceted approach aims to comprehensively measure students' achievements in CLO2 and CLO3, providing valuable insights into their analytical thinking and programming acumen, targeting the attainment of PLO-4: Investigation and PLO-6: The Engineer and Society.

Objective:

Design and implement a sentiment analysis system in C++ to analyse real-time text input by the user and predict the sentiment of the text into three classes: Positive, Negative, and Neutral sentiments.

Task Components:

Assessment Emphasis:

- Implementation of a sentiment analysis system in C++.
- Evaluation of students' ability to translate theoretical knowledge into practical implementation.
- Assessment of the clarity and efficiency of the code, adherence to coding standards, and the effectiveness of the implemented sentiment analysis system.
- Showcasing creativity and problem-solving skills during the coding and implementation phases.

Deliverables:

- Report on sentiment analysis, addressing societal issues related to sentiments in text over social media and proposing solutions achievable through a sentiment analysis system.
- Algorithm and flowchart detailing the design and logic of the sentiment analysis system.
- Comprehensive documentation explaining the thought process, decisions made during algorithm design, and challenges addressed during implementation.
- Sentiment analysis system coded in C++.
- Well-documented, efficient C++ code implementing the sentiment analysis system.
- Example sentences demonstrating the prediction of sentiment in the text through the implemented system.

Outcomes:

- **Practical Application of Programming Concepts:** In this OEL, students will practically apply programming concepts by implementing the sentiment analysis system in C++, bridging the gap between theoretical knowledge and real-world application (CLO3:C5).
- **Tangible Solution:** The culmination of the exercise is the tangible outcome of a well-executed sentiment analysis system. This reflects not only mastery of programming skills but also a practical understanding of how these skills can be employed to address real-world challenges.
- **Comprehensive Skill Set:** Overall, the exercise aims to equip students with a comprehensive skill set, encompassing analytical thinking, effective problem-solving capabilities, and proficiency in practically applying programming concepts to solve complex engineering problems.

Associated Course Learning Outcome:

Following course learning outcome is associated with this activity:

	CLOs	Taxonomy level	Programme learning outcome (PLO)
CLO 3	**Design** and **develop** innovative programming and AI based solutions to tackle complex real-world engineering challenges.	C5	PLO-6

TARGETED CEA ATTRIBUTES:

CEA Attributes
☑ Range of Resources
☑ Level of Interaction
☑ Innovation
☐ Consequences
☑ Familiarity

Assessment:

Assessment will be done according to the attached rubric.

NED University of Engineering & Technology
Department of Electronic Engineering
Course Code & Title: EL-105 Computer and Programming
Assessment Rubric for OEL

F/OBEM 01/18/00

Criterion	**Level of Attainment**				
	Below Average (1)	**Average (2)**	**Good (3)**	**Very Good (4)**	**Excellent (5)**
Identification of software menu (syntax, components, commands, tools, layout etc.)	Can't identify software menus	Rarely identifies software menus	Occasionally identifies software menus	Able to identify software menus	Perfectly able to identify software menus
Adherence to coding standards, code quality, and overall system effectiveness	Fails coding standards; poor code quality; ineffective system	Minimal adherence; basic code quality; basic system effectiveness	Consistent adherence; good code quality; proficient system effectiveness	Exemplary adherence; advanced code quality; highly effective system	Sets gold standard; exceptional code quality; outstanding system effectiveness
Practical application of programming concepts in C++, adhering to the defined algorithm and flowchart	Limited C++ application; struggles with algorithm	Basic C++ skills; some algorithm deviations	Effective C++ application; minor algorithm deviations	Advanced C++ application; close algorithm adherence	Exceptional C++ mastery; flawless algorithm execution
Creativity and problem-solving skills demonstrated during the coding and implementation phases	Minimal creativity and problem-solving evident	Basic demonstration of creativity and problem-solving	Effective demonstration of creativity and problem-solving	Advanced creativity and problem-solving skills showcased	Exceptional creativity and problem-solving mastery demonstrated
OEL Report	Unable to submit the report	Report is submitted but is incomplete and does not follow the prescribed format	Report is submitted and somewhat follows the prescribed format with major portions missing	Report is submitted and somewhat follows the prescribed format with few portions missing	Complete report with proper format is submitted

Student's Name: ___________________________________ Seat No.: ________________

Total Score = ____________________________ Instructor's Signature: ____________________