# CS224 Object Oriented Programming and Design Methodologies Lab Manual

## Lab 1- Introduction to C++

Department of Computer Science

Dhanani School of Science and Engineering

Habib University

Fall 2025

# Contents

# Lab 1

# Introduction to C++

## 1.1 Guidelines

1. Use of AI is strictly prohibited. This is not limited to the use of AI tools for code generation, debugging, or any form of assistance. If detected, it will result in immediate failure of the lab, student will be awarded 0 marks, reported to the academic integrity board and appropriate disciplinary action will be taken.
2. Absence in lab regardless of the submission status will result in 0 marks.
3. All assignments and lab work must be submitted by the specified deadline on Canvas. Late submissions will not be accepted.

## 1.2 Objectives

Following are the lab objectives of this lab:

1. Understand and run a basic C++ program
   - Write, compile, and execute a simple program.
   - Explain the role of the `main()` function and preprocessor directives like `#include`.
2. Use C++ input/output operations
   - Apply the stream insertion and extraction operators for console I/O.
   - Format output using `std::endl` and escape sequences.
3. Work with C++ fundamentals
   - Use comments for documentation.
   - Understand the purpose of the C++ Standard Library and the std namespace.
   - Declare and initialize variables of various data types (`int`, `float`, `double`, `char`, `bool`, etc.).
   - Use `sizeof()` to determine memory usage of data types.
4. Apply C++ operators
   - Perform arithmetic, assignment, relational, and logical operations.
   - Differentiate between simple and compound assignment operators.
5. Implement control structures using if, if-else, and if-else if decision-making statements.
6. Solve basic computational problems in C++ by implementing small programs for arithmetic calculations, unit conversions, and basic algorithms based on given problem statements.

## 1.3 Directory Structure

Labs will have following directory structure:

```
lab
├── manual
│   └── lab.pdf
├── src
│   ├── *.hpp
│   └── *.cpp
├── tests (optional)
│   └── *.cpp
└── makefile (optional)
```

`manual` will contain the lab manual pdf. `src` will contain the source code files, and `tests` (if present) will contain the test files. `makefile` (if present) will contain the makefile for testing and running.

## 1.4 C++ Basics

Before we start with our first program, let's take a look at some basic concepts in C++.

### 1.4.1 Comments

Comments are used to explain the code and make it more readable. In C++, comments can be written in two ways:

```cpp
// This is a single-line comment
/* This is a multi-line comment
    that spans multiple lines */
```

### 1.4.2 Standard Library

The word library refers to a collection of software resources (e.g., functions), usually written by others, that we can use in our programs. The standard library occupies a large part of the C++ standard and provides a large collection of components that facilitate the work of the programmer. The standard library is defined in a separate namespace (see 1.4.4 for namespace), which is called `std` [1].

### 1.4.3 Preprocessor

C++ uses a software program called *preprocessor*. The preprocessor is typically a part of the compiler and its role is to process the program before it is compiled. The preprocessor communicates with the compiler through directives. A preprocessor directive instructs the compiler to act accordingly. For example, with the `#include` directive, the preprocessor instructs the compiler to include the contents of the iostream file in the program before it is compiled. Regarding syntax, directives always begin with the `#` character and do not end with a semicolon (`;`) or some other special marker [1].

### 1.4.4 Namespace

C++ allows the grouping of data (e.g., classes, functions, variables, ...) in a common namespace. Thus, the namespace is a part of the program, in which certain names (e.g., variables) are declared. These names are not known outside of this area. For example, all the names of the standard library are defined in a namespace called `std`. We must write the namespace followed by the scope resolution operator `::`. For example, when we write `std::cout` we access the `cout` object that is declared in the `std` namespace. If we want to make available all the names of the `std` namespace, we write:

```cpp
using namespace std;
```

Now, we don't need to add the prefix `std::` before the names we use, that is, we can write:

```cpp
cout << "Hey Ho, Let's Go\n";
```

Alternatively, we can make available only the names that we use. This is achieved by using corresponding using declarations. For example:

```cpp
using std::cout;
```

Now, we don't need to add the prefix `std::` when we use the `cout`, while we have to add it when using other `std` names [1]. For small programs, it is common to use the `using namespace std;` directive to avoid writing the `std::` prefix repeatedly. But it is considered a bad practice. In larger programs, it is better to use the `using` declaration for specific names or to use the `std::` prefix to avoid name conflicts and improve code clarity.

### 1.4.5 The `main()` Function

Each C++ program must contain a function named `main()`. The word `main()` must be written in lowercase characters. The code of the program, or else the body of the function, must be enclosed in braces ({}). A statement is a command that will be executed when the program runs. Statements are typically written in separate lines and, almost always, each statement ends with a semicolon. Although the compiler does not care about the layout of the program, proper indentation and spacing make your program easier to read. Braces are used to group declarations and statements into a block or else a compound statement that the compiler treats as one. Besides functions, we'll use braces in control statements and loops [1].

### 1.4.6 Hello World

Your first C++ program is a simple "Hello, World!" program.

```cpp
#include <iostream>

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

Listing 1.1: `hello_world.cpp`

The first line of this source code is a preprocessor directive (see 1.4.3) that tells the C++ compiler where to find the definition of the `std::cout` object that is used on the fourth line. The `iostream` file, where the letter `i` corresponds to input and `o` to output, contains information about classes and functions that are necessary in order to read and display data. If we do not include this file, the compiler will not recognize the `cout` and the compilation will fail [2]. Every C++ program that has standard input and output must include this preprocessor directive.

The third line is also required in every C++ program, one and only one time [2]. It tells where the program begins. The identifier `main` is the name of a function (see 1.4.5), called the main function of the program. The required parentheses that follow the word "main" indicate that it is a function. The keyword `int` is the name of a data type in C++. It stands for "integer". It is used here to indicate the return type for the `main()` function. When the program has finished running, it can return an integer value to the operating system to signal some resulting status.

The fourth and fifth lines constitute the actual body of the program. A program body is a sequence of program statements enclosed in braces {}. In this example there are two statement:

```cpp
    std::cout << "Hello, World!" << std::endl;
    return 0;
```

It says to send the string `"Hello, World!"` to the standard output stream object `std::cout`. The single symbol `<<` represents the C++ output operator. When this statement executes, the characters enclosed in quotation marks `" "` are sent to the standard output device which is usually the computer screen. `std::endl` represents the end of the line and flushes the output buffer. Finally, note that every program statement must end with a semicolon (`;`).

Another valid version of the same program is shown below.

```cpp
#include <iostream>

using namespace std;

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

Listing 1.2: `hello_world_with_namespace.cpp`

C++ uses compiler to translate the source code into machine code. The compiler is a program that translates the source code into machine code, which is a set of instructions that the computer can understand. The compiler also checks for syntax errors and other issues in the code. If there are no errors, the compiler generates an executable file that can be run on the computer.

`g++` is the most popular C++ compiler. We can use it to compile our C++ programs.

```
>>> g++ hello_world.cpp -o hello_world
```

In above command, `g++` is the name of the compiler, `hello_world.cpp` is the name of the source file, and `-o` is an option that specifies the name of the output file. In this case, the output file will be named `hello_world`. We can run the program by typing the following command in the terminal:

```
>>> ./hello_world
```

### 1.4.7 Data Types

C++ provides a set of data types. Each variable must have a type. The type determines the amount of memory allocated to the variable, the range of values that can be assigned to it, and the kind of operations that can be applied to it. The size of the types is implementation dependent, that is, it can vary among different systems.

| Type | Typical Bit Width | Typical Range |
|---|---|---|
| `char` | 1 byte | -127 to 127 or 0 to 255 |
| `unsigned char` | 1 byte | 0 to 255 |
| `signed char` | 1 byte | -127 to 127 |
| `int` | 4 bytes | -2147483648 to 2147483647 |
| `unsigned int` | 4 bytes | 0 to 4294967295 |
| `signed int` | 4 bytes | -2147483648 to 2147483647 |
| `short int` | 2 bytes | -32768 to 32767 |
| `unsigned short int` | 2 bytes | 0 to 65,535 |
| `signed short int` | 2 bytes | -32768 to 32767 |
| `long int` | 8 bytes | -9223372036854775808 to 9223372036854775807 |
| `signed long int` | 8 bytes | same as long int |
| `unsigned long int` | 8 bytes | 0 to 18446744073709551615 |
| `long long int` | 8 bytes | -$2^{63}$ to $2^{63}$-1 |
| `unsigned long long int` | 8 bytes | 0 to 18,446,744,073,709,551,615 |
| `float` | 4 bytes | |
| `double` | 8 bytes | |
| `long double` | 12 bytes | |
| `wchar_t` | 2 or 4 bytes | 1 wide character |

### 1.4.8 Variables

*A variable is a memory location with a given name and the value of a variable is the content of its memory location [1].* There are some basic rules for naming variables. These rules also apply for function names. Be sure to follow them or your code won't compile:

1. The name or else the identifier can contain letters, digits, and underscore characters _. The language does not set a limit on the length of the name.

2. The name must begin with either a letter or the underscore character.

3. C++ is case sensitive, meaning that it distinguishes between uppercase and lowercase letters. For example, the variable `sum` is different from the variables `Sum` or `sUM`.

4. The following keywords cannot be used as variable names because they have special significance to the C++ compiler.

| | | | | |
|---|---|---|---|---|
| alignas | const | friend | protected | true |
| alignof | const_cast | goto | public | try |
| and | constexpr | if | register | typedef |
| and_eq | continue | inline | reinterpret_cast | typeid |
| asm | decltype | int | return | typename |
| auto | default | long | short | union |
| bitand | delete | mutable | signed | unsigned |
| bitor | do | namespace | sizeof | using |
| bool | double | new | static | virtual |
| break | dynamic_cast | noexcept | static_assert | void |
| case | else | not | static_cast | volatile |
| catch | enum | not_eq | struct | wchar_t |
| char | explicit | nullptr | switch | while |
| char16_t | extern | operator | template | xor |
| char32_t | false | or | this | xor_eq |
| class | float | or_eq | thread_local | |
| compl | for | private | throw | |

5. In order to avoid name conflicts, don't choose names that begin with one or two _ characters, because these choices are reserved for use in the standard library. Also, don't use names that the compiler uses, such as names of library functions or variables (e.g., `cout`). Their use is allowed, but it is confusing and dangerous. Therefore, it is safer to handle predefined names as if they were reserved words.

Variables must be declared before used in the program. The declaration informs the compiler about the name of the variable and its type.

```
data_type variable_name;
```

The `variable_name` is the name of the variable and the `data_type` its type. As we said, C++ is a statically typed language, in the sense that the compiler must know the type of every entity (e.g., variable) at the point of its use. The type of the entity determines the operations we can perform on it. For example, to declare an integer variable named `i` and a floating-point variable named `j`, we write:

```
int i;
float j;
```

The traditional way to assign a value to a variable is by using the assignment operator `=`. For example, the following code assigns the value 2 to `a`:

```
int a;
a = 2;
```

Alternatively, a variable can be initialized together with its declaration:

```
int a = 2;
```

We can also initialize more than one variable of the same type when declared. For example, the following statement declares the `a`, `b`, `c`, and `d` variables and initializes the first three with the values `2`, `3`, and `4`, respectively. The variable `d` is initialized with a garbage value.

```cpp
int a = 2, b = 3, d, c = 4;
```

We could even write:

```cpp
int a = 2, b = a+1, d, c = b+1;
```

The assignments take place from left to right, meaning that first `a` becomes `2`, then `b` becomes `3`, and then `c` becomes `4`. Here is another way:

```cpp
int a(2), b(a+1), c(a+2);
```

### 1.4.9 Operators

#### Insertion/Extraction Operators

In C++, data input and output is accomplished via streams. The term stream refers to a data source or destination. For example, an output stream may be associated with the screen, a file, or an output device (e.g., printer). C++ provides a set of classes declared in the `iostream` and `fstream` header files to support data input/output operations.

`cout` is an object of the `ostream` class. By default, the `cout` object is associated with the predefined standard output (e.g., screen). The `cout` is followed by the `<<` operator and the expression we want to display. By default, the values written in `cout` are converted to a sequence of characters. For example, if we write:

```cpp
cout << 35;
```

the characters `3` and `5` are inserted into the output stream. The `<<` operator sends the data on its right to `cout`. The `<<` visually suggests that the direction of the data flow is from right to left.

An input stream may be associated with the keyboard, a file on the hard disk, or a peripheral device (e.g., modem). `cin` is an object of the `istream` class. By default, the `cin` object is associated with the standard input device (e.g., keyboard). The type of the variable after the `>>` determines the type of the input data that will be read and stored in that variable. Just as the `ostream` class overloads the `<<` operator, the `istream` class overloads the `>>` right shift operator to read the data. When used in this way, the `>>` operator extracts the data from the input stream and stores it into the indicated variable. The `>>` visually suggests that the direction of the data flow is from left to right. Both `istream` and ostream classes are defined in the iostream file. The `istream` class provides several functions to get the input data. For now, we'll read numeric values and store them in appropriate variables.

```cpp
#include <iostream>

int main() {
    int    i;
    double j;

    std::cout << "Enter number: ";
    std::cin >> i;  // Read an integer and store it into i.
    std::cout << "Enter number: ";
    std::cin >> j;
    std::cout << i << '\t' << j << '\n';

    return 0;
}
```

When the `cin` statement is executed, the program waits for the user to enter a value. Usually, a `cout` statement precedes `cin` to indicate to the user what kind of data to enter. Once the user enters an integer and presses Enter, this value will be stored into `i`. Because the value is not stored before pressing Enter, the user can change it.

#### Assignment Operator

The = operator is used to assign a value to a variable. For example, the statement `a = 20;` assigns the value 20 to the variable `a`, while the statement `a = b;` assigns the value of the variable `b` to `a`. When used in a chained assignment, the assigned value is stored in all variables in the chain. For example:

```cpp
int a, b, c;
a = b = c = 20;
```

the values of `a`, `b`, and `c` become 20.

#### Arithmetic Operators

The arithmetic operators `+`, `-`, `*`, `/`, and `%` are used to perform addition, subtraction, multiplication, division, and modulo operations, respectively.

#### Compound Assignment Operators

The compound assignment operators `+=`, `-=`, `*=`, `/=`, and `%=` are used to perform an operation and assignment in a single step. For example:

```cpp
#include <iostream>

int main() {
    int a = 1, b = 2;

    a -= 2;       // compound subtraction
    a *= 1 - b;   // compound multiplication
    a += b + 3;   // compound addition
    a /= b + 2;   // compound division
    a %= b;       // compound modulus

    std::cout << a << std::endl;

    return 0;
}
```

#### Relational Operators

The relational operators `>`, `>=`, `<`, `<=`, `!=`, and `==` are used to compare two operands and determine their relationship.

#### Logical Operators

The logical operators `!`, `&&`, and `||` are used to form logical expressions. The `!` operator is unary, while `&&` and `||` are both binary. `!` is logical not, `&&` is logical and, and `||` is logical or. The logical operators produce either `true` or `false`, just like the relational operators. The logical operators are left associative, so the compiler evaluates the operands from left to right.

### 1.4.10 Control Structures

#### `if` statement

The `if` statement controls program flow based on whether a condition evaluates to `true` or `false`. The simplest form of the `if` statement is:

```cpp
if (condition) {
    ... // block of statements
}
```

If the value of the condition is `true`, the block of statements between the braces will be executed.

#### `if-else` Statement

As we saw in the previous example, an `if` statement may have an `else` clause:

```
if (condition) {
    ... // block of statements A
} else {
    ... // block of statements B
}
```

If the condition is `true`, the first block of statements is executed; otherwise, the second one.

#### Nested `if-else if-else` Statements

If a program requires multiple conditions to be checked, we can use nested `if-else if-else` statements:

```
if (condition_A) {
    ... // block of statements A
} else if (condition_B) {
    ... // block of statements B
}
.
.
.
else {
    ... // block of statements N
}
```

There can be any number of `else if` clauses between the first `if` and the last `else`. The conditions are evaluated in order starting from the top. Once a `true` condition is found, the corresponding block of statements is executed and the remaining `else if` statements are ignored.

## 1.5 Exercises

1. [11 points] Write a C++ program `greetings.cpp` that greets on your behalf. The program should print your name and a short message, such as "Hello, my name is John Doe!". *Note that this program does not require any input from the user.*

2. [11 points] Write a program `mini_calculator.cpp` that takes two integers as inputs from the user and prints their sum, difference, product, and quotient. See below for an example of how the program should interact with the user.

   ```
   >>> Enter first number: 10
   >>> Enter second number: 5
   Sum: 15
   Difference: 5
   Product: 50
   Quotient: 2
   ```

3. [11 points] Write a program `case_converter.cpp` that takes a lowercase character from user and converts it into uppercase character.

   ```
   >>> Enter a lowercase character: a
   A
   ```

   *Hint: use ASCII codes.*

4. [11 points] You have been given a task as a programmer on a Habib Super Computer. In order to do some calculations, you need to know how many bytes the following data types use: `char`, `int`, `float`, `double`, `long`, `short` and `bool`. You don't have any manuals so you can't look this information up. Write a C++ program `data_types_sizes.cpp` that will determine the amount of memory used by these types in bytes and display the information on the screen. *Hint: Use `sizeof()` operator.*

```
The size of char is: 1 bytes
The size of int is: 4 bytes
The size of float is: 4 bytes
The size of double is: 8 bytes
The size of long is: 8 bytes
The size of short is: 2 bytes
The size of bool is: 1 bytes
```

5. [11 points] Write a program `currency_converter.cpp` that takes an `amount` as input and find the minimum number of notes of different denominations that sum up to the given amount. Starting from the highest denomination note, try to accommodate as many notes possible for given amount. We may assume that we have infinite supply of notes of values: 5000, 1000, 500, 100, 50, 20, 10, 5, 2 and 1. See below for an example of how the program should interact with the user.

```
>>> Enter amount: 10200
5000 : 2
1000 : 0
500 : 0
100 : 2
50 : 0
20 : 0
10 : 0
5 : 0
2 : 0
1 : 0
```

```
>>> Enter amount: 2769
5000 : 0
1000 : 2
500 : 1
100 : 2
50 : 1
20 : 0
10 : 1
5 : 1
2 : 2
1 : 0
```

6. [15 points] Asim is playing cricket and hits six and the ball lands on the 19th floor of a building. Hania, Asim's friend lives on 19th floor of that building. Asim asks for the ball to be returned and Hania drops the ball from the balcony of the 19th floor. Each floor has a height of 18 feets. Write a program `free_fall.cpp` that calculates the time it takes for the ball to hit the ground floor in seconds. You can use second equation of free-fall motion,

$$h = v_0 t + \frac{1}{2} g t^2$$

where, $h$ is the height, $v_0$ is the initial velocity, and $g$ is the acceleration due to gravity (approximately $9.81\,\mathrm{m/s^2}$). *Note: Hania drops the ball not throws it. Make sure units are consistent (1 m = 3.28084 fts).*

7. [15 points] Talha is singing in concert and while singing he felt thirsty because its a hot summer night. We don't know what was the temperature that night, because the thermometer was using the Fahrenheit scale. Write a program `temperature_converter.cpp` that converts temperatures between Celsius and Fahrenheit. The program should prompt the user for a temperature and the unit (C or F), perform the conversion, and print the result. Celsius and Fahrenheit are related by the following formula,

$$5F - 9C = 160$$

where, $F$ is the temperature in Fahrenheit and $C$ is the temperature in Celsius. See below for examples of how the program should interact with the user.

```
>>> Enter temperature: 100
>>> Enter unit (C/F): C
Temperature in Fahrenheit: 212
```

```
>>> Enter temperature: 32
>>> Enter unit (C/F): F
Temperature in Celsius: 0
```

8. [15 points] Quadratic Polynomials are defined as $p(x) = ax^2 + bx + c$, where $a$, $b$, and $c$ are real numbers. Solution of a Quadratic equation is defined as some value of $x$ such that $p(x) = 0$. Luckily there is a formula that gives us the solution of a Quadratic equation. The solutions are given by the formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Write a program `quadratic_solver.cpp` that takes as input the coefficients $a$, $b$, and $c$ of a Quadratic polynomial and prints the solutions of the corresponding Quadratic equation. The program should also handle the case where there are no real solutions (i.e., when $b^2 - 4ac < 0$). See below for an example of how the program should interact with the user.

```
>>> Enter coefficient a: 1
>>> Enter coefficient b: -3
>>> Enter coefficient c: 2
Solutions: x1 = 2, x2 = 1
```

```
>>> Enter coefficient a: 1
>>> Enter coefficient b: 1
>>> Enter coefficient c: 1
Solutions: x1 = -0.5 + 0.866025i, x2 = -0.5 - 0.866025i
```

*Note: ignore the rounding of the floating point numbers.*

# References

[1] G.S. Tselikis. *Introduction to C++*. CRC Press, 2022. ISBN: 9781000635744. URL: https://books.google.com.pk/books?id=LNx1EAAAQBAJ.

[2] J.R. Hubbard. *Schaum's Outline of Programming with C++*. Schaum's Outline Series. McGraw Hill LLC, 1996. ISBN: 9780071368117. URL: https://books.google.com.pk/books?id=KtKsBAAAQBAJ.