Al-Mustaqbal University
College of Sciences
Intelligent Medical System Department

جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

# كلية العلوم
# قسم الانظمة الطبية الذكية

# Lecture: ( 1 )

*Introduction, Procedural Programming Principles, Introduction to algorithm, Algorithms example*

**Subject: Programming fundamental**
**Level: First**
**Lecturer: Dr. Maytham N. Meqdad**


Study Year: 2025-2024

## Introduction to C++ Programming Language

**C++** is a general-purpose programming language that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a **compiled** language.

1. C++ is a high-level, general-purpose programming language designed for system and application programming. It was developed by Bjarne Stroustrup at Bell Labs in 1983 as an extension of the C programming language. C++ is an object-oriented, multi-paradigm language that supports procedural, functional, and generic programming styles.

2. One of the key features of C++ is its ability to support low-level, system-level programming, making it suitable for developing operating systems, device drivers, and other system software. At the same time, C++ also provides a rich set of libraries and features for high-level application programming, making it a popular choice for developing desktop applications, video games, and other complex applications.

3. C++ has a large, active community of developers and users, and a wealth of resources and tools available for learning and using the language. Some of the key features of C++ include:

4. Object-Oriented Programming: C++ supports object-oriented programming, allowing developers to create classes and objects and to define methods and properties for these objects.

5. Templates: C++ templates allow developers to write generic code that can work with any data type, making it easier to write reusable and flexible code.

6. Standard Template Library (STL): The STL provides a wide range of containers and algorithms for working with data, making it easier to write efficient and effective code.

7. Exception Handling: C++ provides robust exception handling capabilities, making it easier to write code that can handle errors and unexpected situations.

Overall, C++ is a powerful and versatile programming language that is widely used for a range of applications and is well-suited for both low-level system programming and high-level application development.

## Here are some simple C++ code examples to help you understand the language:

### 1.Hello World:

```
#include <iostream>

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

- **Output**

**Hello, World!**

---

**Steps for C++ Program Development and Execution**

1. **Editing**
2. **Compiling**
3. **Linking Library files**
4. **Loading**
5. **Execution**

Editing:

Editing refers the typing or writing the program in any text editor. But we want all the things in one place like writing the program, compiling, and executing it. This is achieved with the help of software that is known as IDE (Integrated Development Environment). IDE integrated all the tasks that are required to run a program.

Examples of IDEs: Turbo C++, Devcpp, Xcode, Visual Studio Code, CodeBlocks, Eclipse, etc.

Compiling:

Consider a program first.cpp which is saved in Hard Disc. To compile the first.cpp file, we need an IDE that contains a compiler. The compiler converts the high-level code into machine-level language code and a new executable file with the name first.exe is generated and get stored in the hard disc. If the compiler finds any error in the code, it throws the error to the programmer else the code is successfully compiled.

Example: When first.cpp is compiled, the executable files are generated like max.exe and main.exe and get stored in the hard disc to get executed later.

**Linking Libraries:**

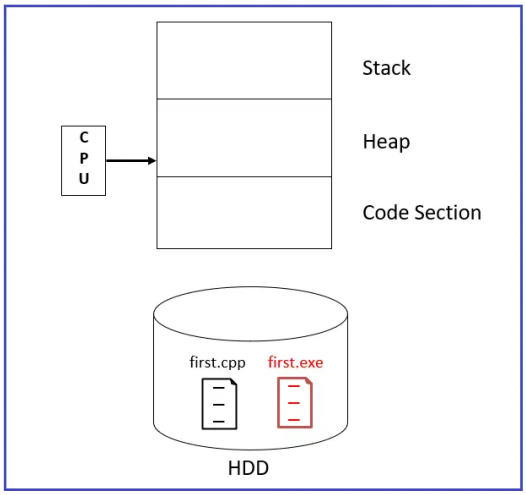
Every language has some built-in objects and functions that can be reused in any program. The built-in objects and functions are grouped inside libraries that can be included in programs as header files. These libraries and header files are linked with the code during compilation where the library code is also converted to an executable file along with the entire program.

Example: We included iostream which is a header file for cout and cin objects. The iostream is attached to the code during compilation where the header file code is also converted to executable code with .exe extension. This is called the linking of the library.

**Loading:** To execute the program code, the code must be brought to the main memory from the secondary memory.

Execution: As soon as the program gets loaded in the main memory in different sections as given below, the program execution starts. The execution of the program starts from the first line of the main function.

**Main Memory Management**




Main memory has different sections.

1. **Code Section**: The entire machine-level code is copied to the code section of the main memory. All the arrangements that are called relocations are done here and it is done by the operating system.
2. **Stack:** All the variables (that are used for storing the data values) are stored in the stack section of the code.
3. **Heap**: Heap memory stores the dynamically allocated variables, the variable that is allocated during the run time of the program (discussed later in detail).

**Example:** The variables in the program first.cpp are x and y is stored in the stack of main memory. The rest of the entire code is copied to the code section and the heap will be empty in this case as there are no dynamically allocated variables.

# Procedural Programming Principles C++

Procedural programming is a programming paradigm that focuses on defining procedures or routines that are executed sequentially. In C++, procedural programming involves organizing code into functions and structures. Here are some key principles of procedural programming in C++:

**Functions:**

- Break down your program into smaller functions. Each function should perform a specific task.
- Functions should have a clear purpose and should ideally perform a single, well-defined operation.

```
// Function declaration
void greetUser(std::string name);

int main() {
    // Function call
    greetUser("John");
    return 0;
}

// Function definition
void greetUser(std::string name) {
    std::cout << "Hello, " << name << "!\n";
}
```

**Modularity:**

- Organize your code into modules using functions.
- Use header files for function declarations and separate implementation files.

**Data Structures**

- Use structs to group related data together.

**Sequential Execution:**

- Code is executed sequentially, one statement after another.
- Use control structures for conditional and iterative execution.

```
// Example of sequential execution with if statement
int number = 5;
```

```
if (number > 0) {
    std::cout << "Number is positive.\n";
} else {
    std::cout << "Number is non-positive.\n";
}
```

**Variable Scope:**

- Variables have scope, meaning they are only accessible within a certain part of the program.
- Use local variables within functions to limit their scope.

# Algorithm

In mathematics and computer science, an algorithm is a finite sequence of rigorous instructions, typically used to solve a class of specific problems or to perform a computation. Algorithms are used as specifications for performing calculations and data processing. More advanced algorithms can use conditionals to divert the code execution through various routes (referred to as automated decision-making) and deduce valid inferences (referred to as automated reasoning), achieving automation eventually. Using human characteristics as descriptors of machines in metaphorical ways was already practiced by Alan Turing with terms such as "memory", "search" and "stimulus".

In contrast, a heuristic is an approach to problem solving that may not be fully specified or may not guarantee correct or optimal results, especially in problem domains where there is no well-defined correct or optimal result.

As an effective method, an algorithm can be expressed within a finite amount of space and time and in a well-defined formal language for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input.

**Programming Algorithm Defined**

A programming algorithm is a procedure or formula used for solving a problem. It is based on conducting a sequence of specified actions in which these actions describe how to do something, and your computer will do it exactly that way every time. An algorithm works by following a procedure, made up of inputs. Once it has followed all the inputs, it will see a result, also known as output.

Characteristics of an algorithm:

1. Precision – the steps are precisely stated.

2. Uniqueness – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
3. Finiteness – the algorithm stops after a finite number of instructions are executed.
4. Input – the algorithm receives input.
5. Output – the algorithm produces output.
6. Generality – the algorithm applies to a set of inputs.

There are seven different types of programming algorithms:

1. Sort algorithms
2. Search algorithms
3. Hashing
4. Dynamic Programming
5. Exponential by squaring
6. String matching and parsing
7. Primality testing algorithms

The advantages of programming algorithms include:

- A stepwise representation of a solution to a given problem, making it easy to understand.
- Uses a definite procedure.
- Not dependent on a particular programming language.
- Every step in an algorithm has its own logical sequence, making it easy to debug.

# Algorithm Examples

## Algorithm 1: Add two numbers entered by the user

```
Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
        sum←num1+num2
Step 5: Display sum
Step 6: Stop
```

## Algorithm 2: Find the largest number among three numbers

```
Step 1: Start
Step 2: Declare variables a,b and c.
Step 3: Read variables a,b and c.
Step 4: If a > b
           If a > c
              Display a is the largest number.
           Else
              Display c is the largest number.
```

```
        Else
           If b > c
              Display b is the largest number.
           Else
              Display c is the greatest number.
Step 5: Stop
```

## Algorithm 3: Find Roots of a Quadratic Equation $ax^2 + bx + c = 0$

```
Step 1: Start
Step 2: Declare variables a, b, c, D, x1, x2, rp and ip;
Step 3: Calculate discriminant
         D ← b2-4ac
Step 4: If D ≥ 0
              r1 ← (-b+√D)/2a
              r2 ← (-b-√D)/2a
              Display r1 and r2 as roots.
        Else
              Calculate real part and imaginary part
              rp ← -b/2a
              ip ← √(-D)/2a
              Display rp+j(ip) and rp-j(ip) as roots
Step 5: Stop
```

## Algorithm 4: Find the factorial of a number

```
Step 1: Start
Step 2: Declare variables n, factorial and i.
Step 3: Initialize variables
          factorial ← 1
          i ← 1
Step 4: Read value of n
Step 5: Repeat the steps until i = n
     5.1: factorial ← factorial*i
     5.2: i ← i+1
Step 6: Display factorial
Step 7: Stop
```

## Algorithm 5: Check whether a number is prime or not

```
Step 1: Start
Step 2: Declare variables n, i, flag.
Step 3: Initialize variables
        flag ← 1
        i ← 2
```

```
Step 4: Read n from the user.
Step 5: Repeat the steps until i=(n/2)
     5.1 If remainder of n÷i equals 0
            flag ← 0
            Go to step 6
     5.2 i ← i+1
Step 6: If flag = 0
           Display n is not prime
         else
           Display n is prime
Step 7: Stop
```

## Algorithm 6: Find the Fibonacci series till the term less than 1000

```
Step 1: Start
Step 2: Declare variables first_term,second_term and temp.
Step 3: Initialize variables first_term ← 0 second_term ← 1
Step 4: Display first_term and second_term
Step 5: Repeat the steps until second_term ≤ 1000
     5.1: temp ← second_term
     5.2: second_term ← second_term + first_term
     5.3: first_term ← temp
     5.4: Display second_term
Step 6: Stop
```