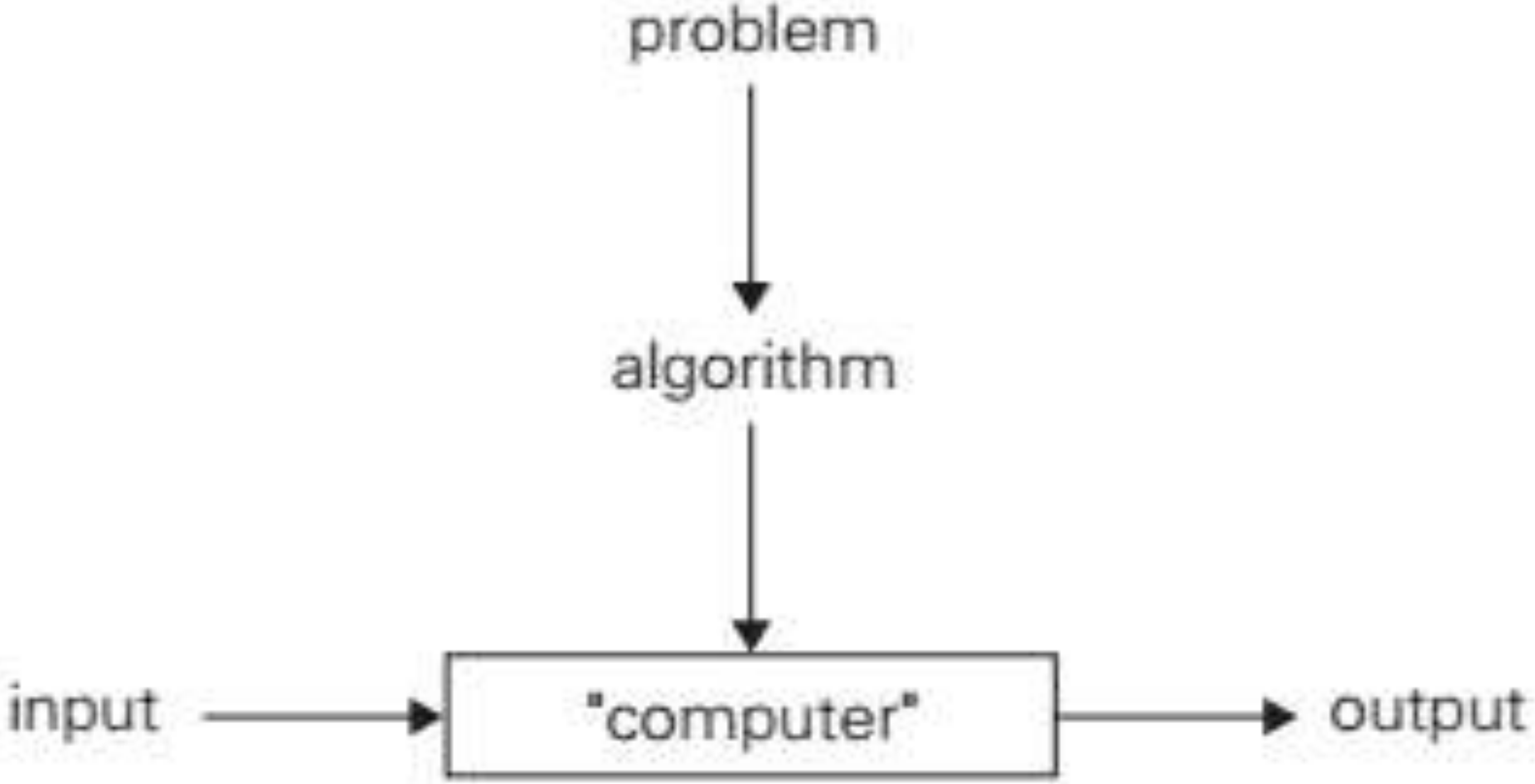
Introduction To Algorithms

CHAPTER 1

1.1 What is an Algorithm?

- An algorithm is a finite sequence of clear, step-by-step instructions, each of which can be carried out in a finite amount of time, to solve a given problem for all valid inputs.
- The concept of algorithms is fundamental in computer science and predates computers themselves.
- Historically, even before electronic computers existed, people (called “computers”) followed algorithms to perform calculations.


problem
algorithm
input
"computer"
output

1.2 Characteristics of Algorithms

For any process to qualify as an algorithm, it must have the following essential properties:

✓ **Input:**

The algorithm receives zero or more inputs from a specified set.

✓ **Output:**

It produces at least one output (result).

✓ **Definiteness:**

Each step must be precisely and clearly defined. There should be no ambiguity in what needs to be done.

✓ **Finiteness:**

The algorithm must terminate after a finite number of steps. It should not go into an infinite loop.

✓ **Effectiveness:**

All operations must be basic enough to be carried out exactly and within a finite amount of time, in principle even by a person with paper and pencil

1.3 Ways to Write and Represent Algorithms

There are several standard ways to write or represent algorithms. The choice depends on the audience, purpose, and the level of detail needed.

1- Natural Language Description

This involves writing the algorithm as simple, step-by-step instructions in plain English (or any language).

✓ Advantages: Easy for beginners to understand.

✓ Disadvantages: Can be ambiguous or too informal for precise communication.

Example:

1. Read two numbers.
2. Add them.
3. Print the result.

2- Pseudocode

Pseudocode uses a structured, programming-like style without worrying about exact syntax of a programming language. It focuses on the logic.

✓ Advantages: Easy to read and write; abstracts away programming details.

✓ Disadvantages: Needs to be later translated to real code.

Example:

```
Input A, B
Sum ← A + B
Output Sum
```

•

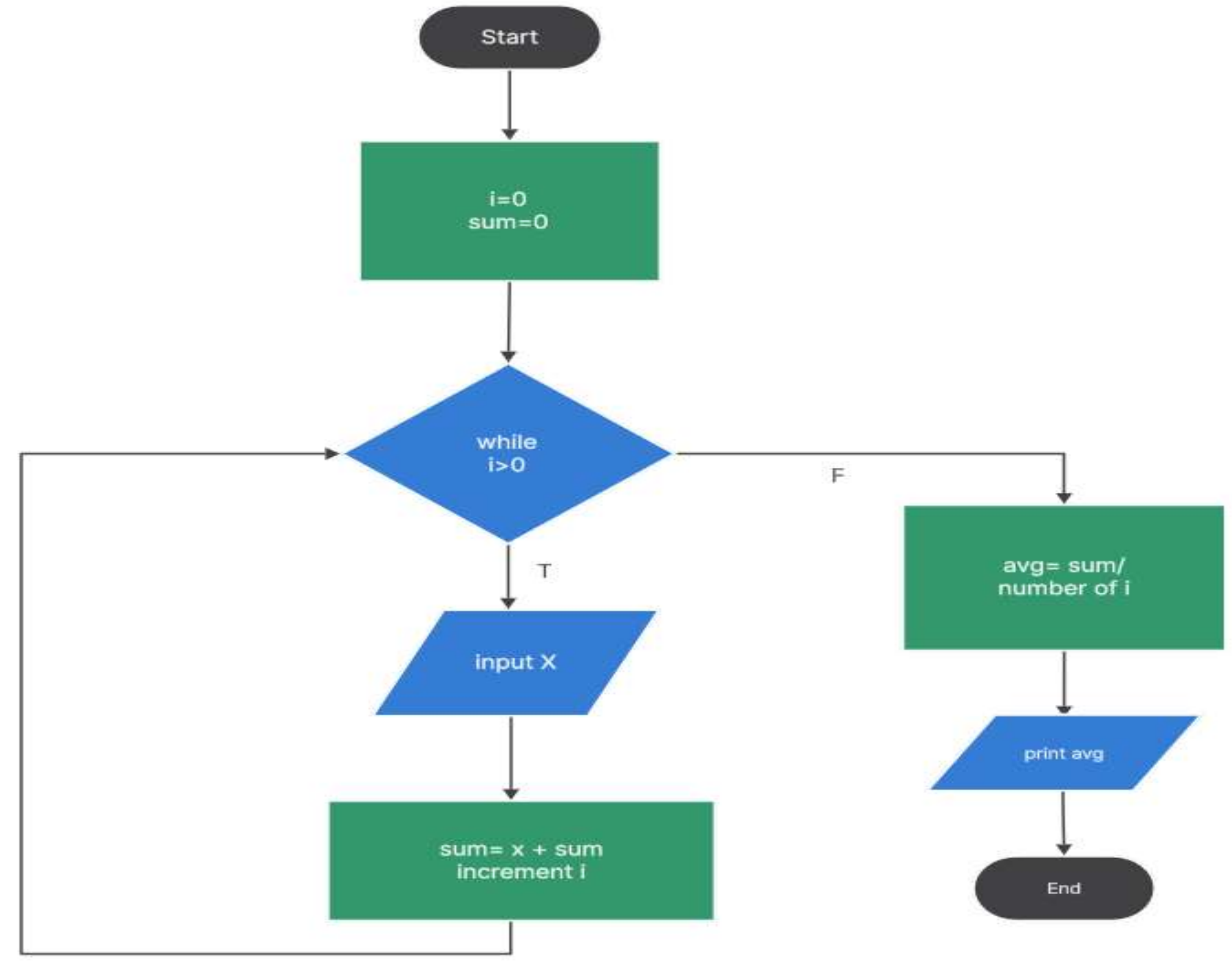
3-Flowcharts

Flowcharts are graphical diagrams that show the flow of control using standard shapes:

- Ovals for Start/End,
- Rectangles for Processes,
- Diamonds for Decisions,
- Arrows for flow direction.

✓ Advantages: Very intuitive; helps visualize the process.

✓ Disadvantages: Can get messy for complex algorithms.


Start
i=0
sum=0
while
i>0
F
T
input X
sum= x + sum
increment i
avg= sum/
number of i
print avg
End

4- Programming Code

Writing the algorithm directly in a programming language like Python, C++, or Java.

✓ Advantages: Can be executed immediately by a computer.

✓ Disadvantages: Less abstract; ties the algorithm to specific syntax and implementation details.

Best Practices for Students

1-Start Simple: Use pseudocode for design → code for implementation.

2-Validate with Math: Prove correctness/complexity formally.

3-Visualize Hard Parts: Draw flowcharts for recursion or tables for DP.

4-Target Your Audience:

□Exams/Textbooks: Pseudocode

□Developers: Executable Code

□Researchers: Math + Pseudocode

5.Use Tools:

□Flowcharts: Diagrams.net, Mermaid

□Pseudocode → Code: LeetCode Playground

□Visualization: VisuAlgo, Algorithm Visualizer

1.4 Algorithm vs. Program

An algorithm is a conceptual, step-by-step procedure to solve a problem.

It is language-independent — it does not depend on any specific programming language or machine.

✓ Example:

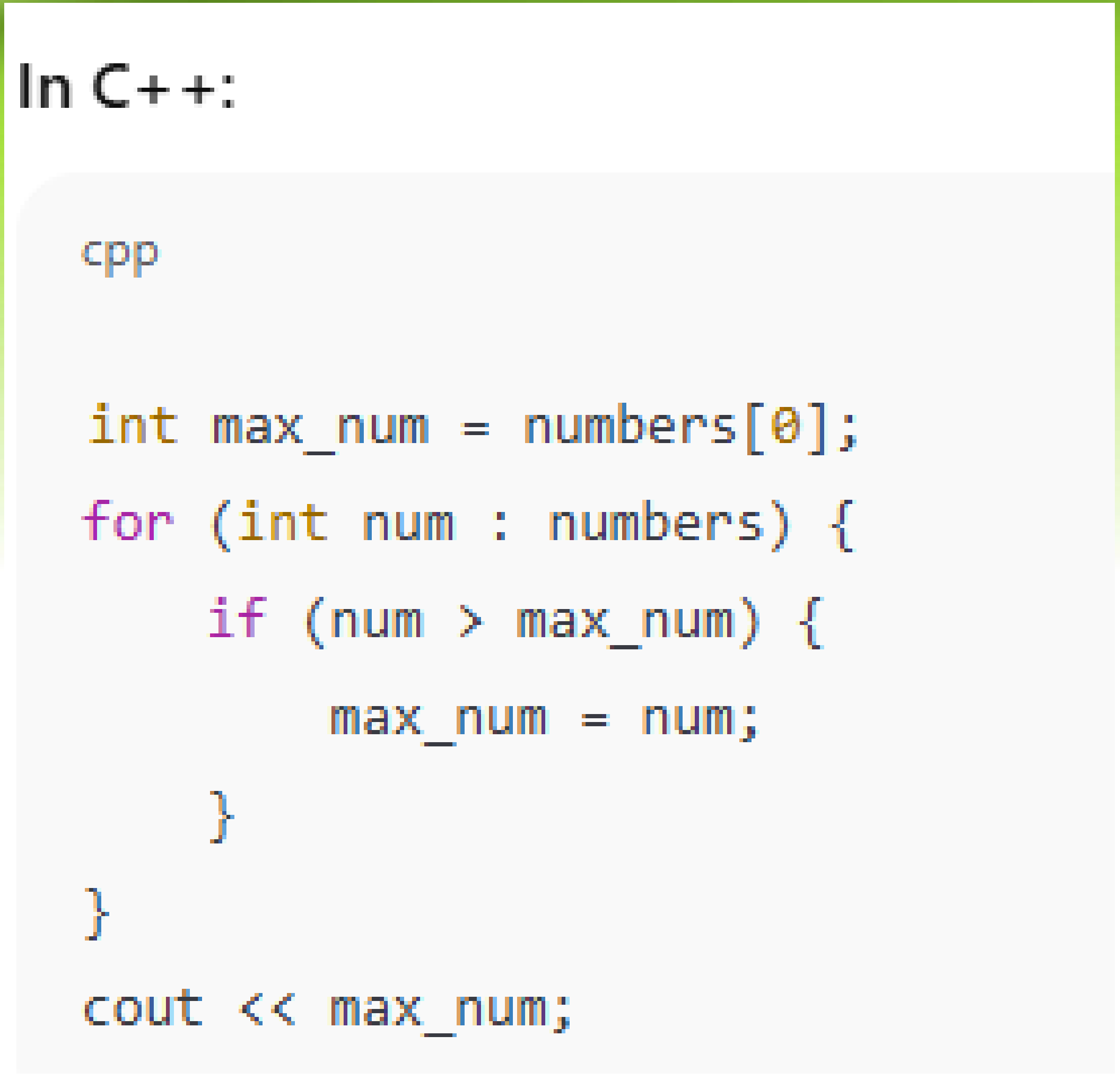
“To find the largest number in a list, go through each item and keep track of the largest one found so far.”

An algorithm only describes what needs to be done, not how to do it in a particular language.

A program is a specific implementation of an algorithm written in a programming language (like Python, C++, or Java), so that a computer can execute it.

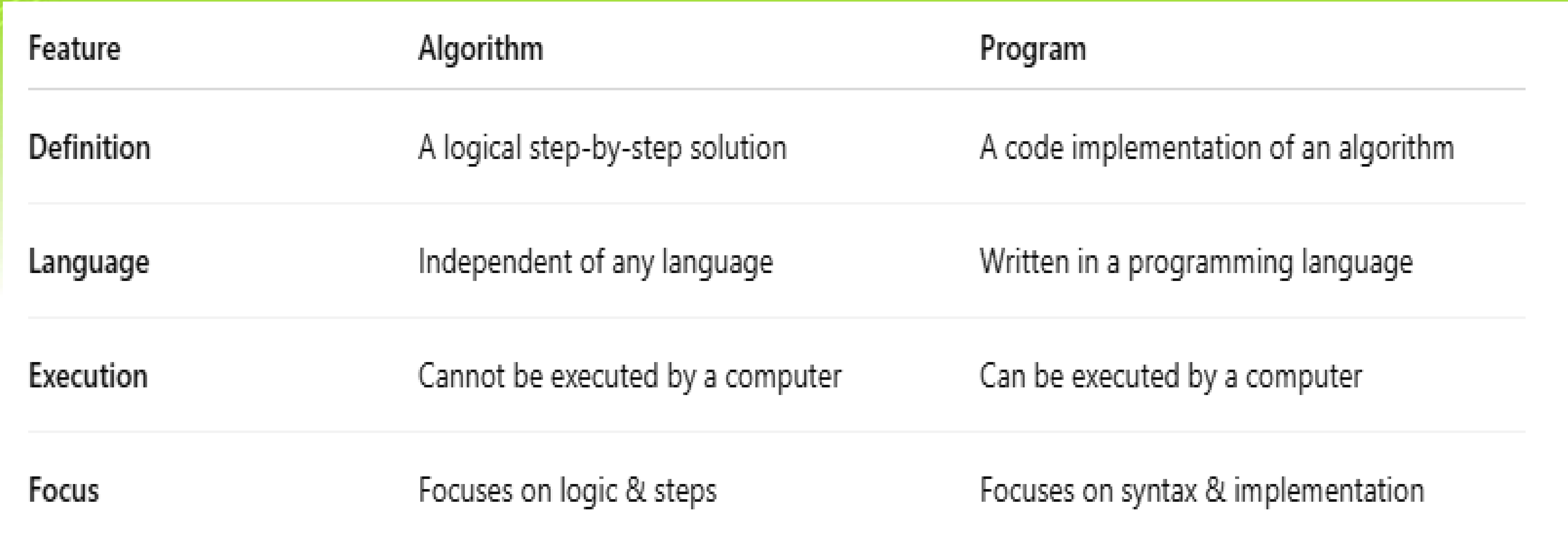
✓ Example:

The same algorithm to find the largest number can be written in different programming languages:

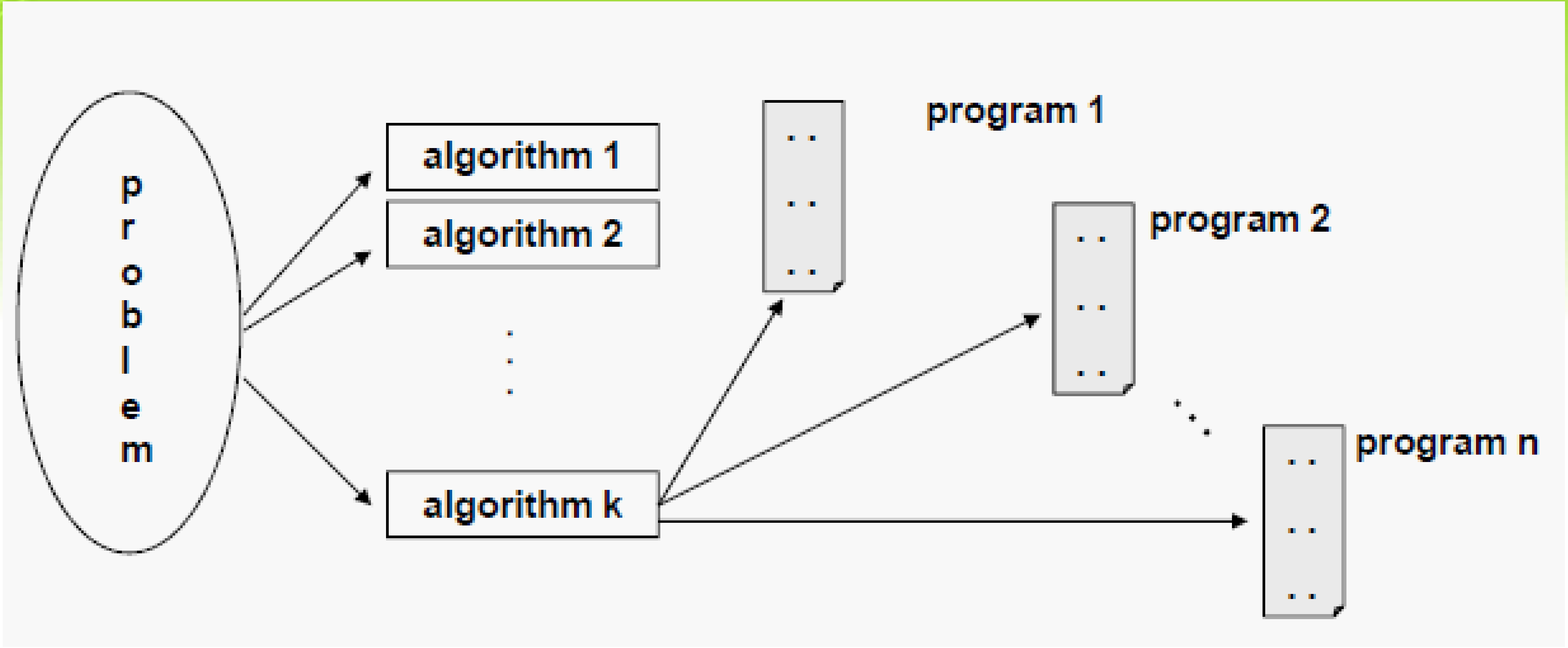


In C++:

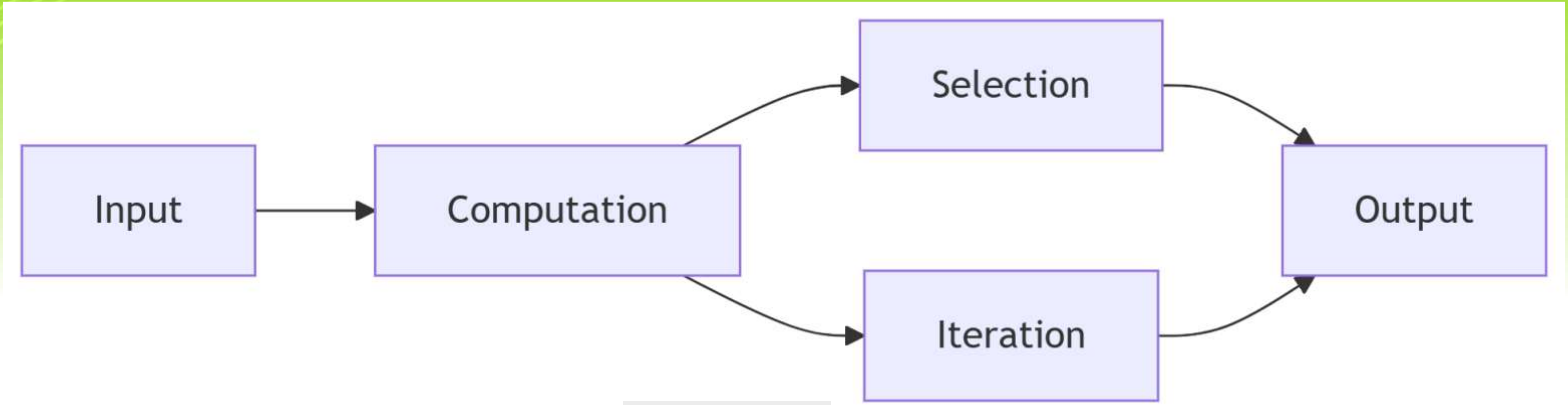
```cpp
int max_num = numbers[0];
for (int num : numbers) {
    if (num > max_num) {
        max_num = num;
    }
}
cout << max_num;
```



Feature	Algorithm	Program
Definition	A logical step-by-step solution	A code implementation of an algorithm
Language	Independent of any language	Written in a programming language
Execution	Cannot be executed by a computer	Can be executed by a computer
Focus	Focuses on logic & steps	Focuses on syntax & implementation


problem
algorithm 1
algorithm 2
algorithm k
program 1
program 2
program n

1.5 Core Algorithm Components



- Input: Acquire data (e.g., READ grade)
- Computation: Arithmetic/logical operations
- Selection: Decision-making (IF-THEN-ELSE)
- Iteration: Repetition (WHILE, FOR)
- Output: Report results (PRINT result)

1.6 Algorithm Design Process

- Designing an algorithm is a systematic process that involves several stages. It ensures that the solution is correct, efficient, and practical for implementation.
- Steps in the Algorithm Design Process

1- Understanding the Problem

- Carefully read and analyze the problem statement.
- Identify the inputs, required outputs, and any constraints.

Ask questions like:

- What exactly is being asked?
- What are the edge cases?
- Are there performance constraints?

2-Developing a Strategy

- Decide how to approach the problem.
- Choose an overall technique:

✓ Greedy algorithms

✓ Divide and conquer

✓ Dynamic programming

✓ Brute force

Sometimes this means considering multiple approaches and comparing them.

3-Designing the Algorithm

- Write a high-level sequence of steps that solves the problem.
- Represent it in pseudocode or a flowchart.
- Ensure each step is clear and unambiguous.

4- Proving Correctness

- Show that the algorithm works for all valid inputs.
- Use logical reasoning or formal proofs if necessary.
- This helps avoid hidden logical errors.

5-Analyzing Efficiency

- Determine the time complexity (how running time grows with input size).
- Determine the space complexity (how much memory it uses).
- Helps decide if the algorithm is practical.

6- Implementing the Algorithm

- Translate the pseudocode into an actual programming language (Python, Java, C++, etc).
- Pay attention to data structures and syntax.

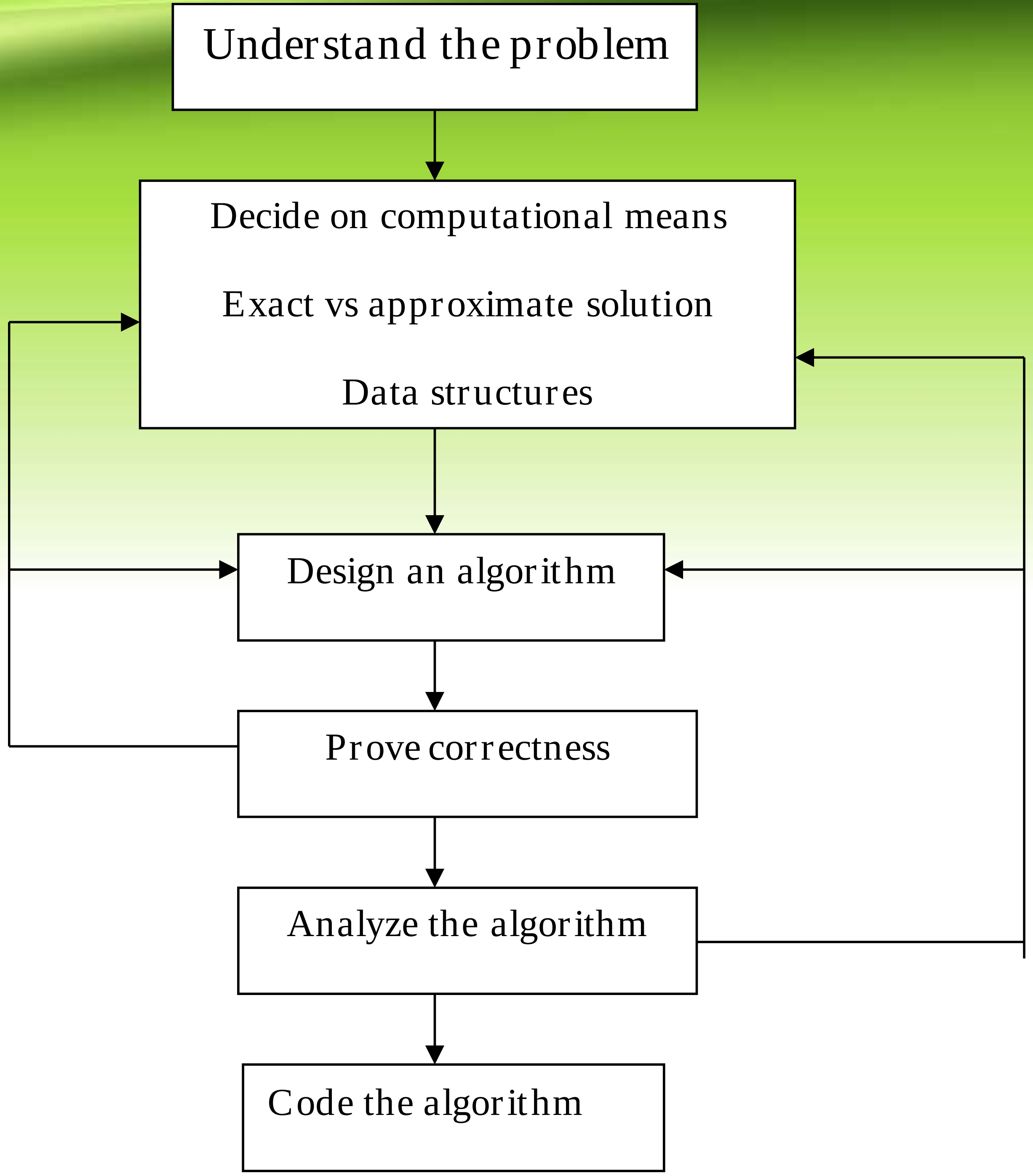
7- Testing and Debugging

- Test the program with typical cases, edge cases, and large inputs.
- Fix any bugs or inefficiencies.

Iteration and Improvement

Often, after implementation and testing, improvements are needed:

- Optimize performance.
- Handle additional edge cases.
- Make the code cleaner or more general


Understand the problem
Decide on computational means
Exact vs approximate solution
Data structures
Design an algorithm
Prove correctness
Analyze the algorithm
Code the algorithm

1.7 Algorithm Examples

1- Finding the Maximum Number in a List

- Goal: Given a list of numbers, find the largest one.
- Algorithm:

```
1. Set max ← first element in the list
2. For each element x in the list:
    if x > max then
        set max ← x
3. Return max
```

✓ Explanation:

- We start by assuming the first number is the largest, then compare each number in the list to update the maximum if we find a bigger one.

2- Calculating the Factorial of a Number

- Goal: Given a positive integer n, compute $n! = n \times (n-1) \times \ldots \times 2 \times 1$.
- Algorithm:

```
1. Set result ← 1
2. For i from 2 to n:
    set result ← result × i
3. Return result
```

✓ Example:

If $n = 5$, then result $= 1 \times 2 \times 3 \times 4 \times 5 = 120$.

3- Linear Search

- Goal: Find a target value in a list.
- Algorithm:

1.For each element x in the list:

If x equals the target, then

→ return "Found".

2.If we reach the end of the list without finding the target,

→ return "Not found".

- Explanation:

We check each element one by one until we find the target (or confirm it's not in the list).

Tasks

1- Euclidean Algorithm for GCD

Goal: Compute the greatest common divisor (GCD) of two integers a and b.

2- Describe the algorithm used by your favorite ATM machine in dispensing cash. (You may give your description in either English or pseudocode, whichever you find more convenient.)