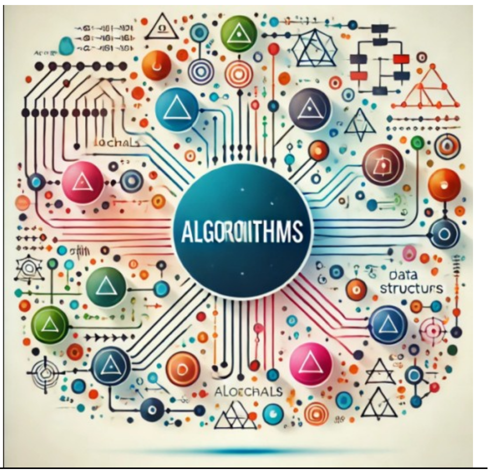
Chapter III:

Algorithms and Flowcharts



Introduction

An algorithm is a set of ordered instructions that enable solving a problem or accomplishing a specific task. The idea is to define a sequence of clear and unambiguous steps to reach a final result. Algorithms are fundamental to many fields, such as programming, mathematics, and computer science. They help break down a complex problem into simpler and more manageable sub-problems.

Origin of the Word "Algorithm"

The word *algorithm* comes from the name of the Arab mathematician **Abu Ja'far Mohammed ben Musa Al-Khwarizmi**, who lived around 780 to 850. Al-Khwarizmi played a fundamental role in the development of mathematics, especially in algebra, which is also derived from his work (from *al-jabr*).

He was from the ancient city of **Khawarizm** (modern-day *Khiva*), located in what is now Uzbekistan, formerly part of the Soviet Union (ex-U.S.S.R.).

Al-Khwarizmi’s writings greatly influenced European mathematicians and introduced key mathematical concepts to the Western world, including the

Arabic numeral system and methods of calculation. His name was Latinized as "Algoritmi," from which the word *algorithm* is derived.

The term *algorithmics* was conceptualized in 1958 by university researchers to refer to the systematic study of methods for solving problems using defined procedures.

1.1 Definition of an Algorithm

An *algorithm* is an ordered and finite sequence of instructions designed to solve a given problem or accomplish a specific task. The purpose of an algorithm is to provide a solution by following a series of defined steps, leading to an expected result.

1.2 Definition and Characteristics of an Algorithm

- **Finite**: It must terminate after a finite number of steps.
- **Precise**: Each step must be clearly defined and unambiguous.
- **Efficient**: It should solve the problem optimally, using minimal resources.
- **General**: It should be applicable to a set of similar cases.

1.3 Structure of an Algorithm

a. Algorithm Name:

- The name identifies the purpose of the algorithm, making its function clear.
- Example: Algorithm: Sum

b. Declarative Part:

- This is where variables and constants used in the algorithm are declared. It specifies the data types.
- It sets up the resources needed for the algorithm's execution.

c. . Executable Part:

- This is the main section where the instructions are written to solve the problem.
- It includes steps, conditions (if, else), loops (for, while), and calculations.

<u>Example:</u>

```
Begin
 Read a, b
 sum ← a + b
 Display "The sum is:", sum
```

d. End of the Algorithm:

- This marks the end of the algorithm.
- It's common to restate the algorithm's name for clarity.
- Example:

```
End of the algorithm CalculateSum
```

Complete Example:

Here's a complete example in pseudocode with the structured parts:

```
Algorithm: CalculateSum
        Variables:
        a, b: Integers
        sum: Integer
```

```
Begin
        Read a, b
        sum ← a + b
        Display "The sum is:", sum
End of the algorithm CalculateSum
```

This structure makes it easier to write, read, and understand algorithms, ensuring that each part has a specific role in solving the problem.

Explanation of the Structure:

- **Algorithm: name of the algorithm** Identifies the algorithm and its purpose.
- **Constants** Lists the constants used in the algorithm (fixed values that do not change).
- **Variables** Declares the variables needed for processing (such as a, b, and sum).
- **Begin and Instructions** The instructions represent the steps to perform the task. Comments or descriptions can be included between the instructions to make the code more readable.
- **End of the algorithm** Marks the end of the algorithm, reminding the name.

1.4. APPROACH AND ANALYSIS OF A PROBLEM

An algorithm is often written in pseudocode to facilitate its later translation into a programming language. Developing a program involves several steps:

1. **Problem Analysis**: This is the first step, where the problem at hand is thoroughly analyzed.
2. **Algorithm Design**: The second step is to create an algorithm that outlines the solution.
3. **Translation into Code**: The third step is to translate the algorithm into a program using a chosen programming language.

Essentially, an algorithm represents the raw solution to a computational problem and is independent of any specific programming language. For example, the same algorithm can be implemented in Java, C++, or any other language.

Writing an algorithm is a thoughtful exercise typically done on paper. The creation of an algorithm comes before the programming phase; it is a problem-solving approach that requires careful planning and precision.

1.5. STRUCTURE OF DATA

In computer science, a data structure is a way of organizing data to facilitate processing. Different data structures exist for different types of data: constants, variables, records, finite compound structures, arrays (over [1..n]), lists, trees, and graphs.

5.1 CONSTANTS AND VARIABLES

Constants are entities that reserve memory space to store data whose values cannot be modified during the execution of the algorithm (i.e., they remain fixed throughout the algorithm). Constants can be of various types: integer, real, boolean, character, etc.

➢ **Declaring Variables:**

Variables are used to store data that can be manipulated within an algorithm. Here are some common types:

- **Integer:** Stores whole numbers without decimals (e.g., 1, -5, 42).
- **Boolean:** Stores binary values, typically *true* or *false*.
- **Character (CHAR):** Stores a single character, such as 'A' or '#'.
- **String (STRING):** Stores a sequence of characters, representing text (e.g., "Hello, World!").

➢ **Declaring Constants:**

Constants are values that remain unchanged throughout the program. They are usually declared at the beginning of a program with a fixed value.

- **Real:** Stores real numbers (decimal values), such as 3.14 or -7.89.

While writing algorithms we will use following symbol for different operations:

Operation	**Symbol**	**Description**
Addition	+	Adds two numbers
Subtraction	-	Subtracts one number from another
Multiplication	*	Multiplies two numbers
Division	/	Divides one number by another
Modulus	%	Remainder of division
Exponentiation	^ or **	Raises a number to the power of another
Floor Division	//	Division without remainder

Et pour les comparaisons :

Comparison	**Symbol**	**Description**
Greater than	>	Checks if left operand is greater than right
Less than	<	Checks if left operand is less than right
Greater than or equal to	>=	Checks if left operand is greater than or equal to right
Less than or equal to	<=	Checks if left operand is less than or equal to right
Equal to	==	Checks if two operands are equal
Not equal to	!=	Checks if two operands are not equal

Ces symboles sont largement utilisés dans les algorithmes pour effectuer des calculs et des

Example of an Algorithm:

- Problem: Calculate the area of a rectangle.

Algorithm:
Bigin
Input length (L) and width (W).
Calculate area = L * W.
Output the area.
End.

1.6 . Control Structures

(a) Iterative Control Structures:

These structures allow certain parts of an algorithm to repeat based on a specified condition. Common iterative structures include:

- **For Loop:** Executes a block of code a specified number of times.
- **While Loop:** Repeats as long as a condition is true.
- **Do-While Loop:** Similar to a while loop, but the code runs at least once, as the condition is checked after the first execution.

(b) Conditional Control Structures:

These structures allow decisions within an algorithm, executing certain actions based on conditions.

- **If-Else:** Executes one block of code if a condition is true, and another if it's false.
- **Switch Case:** Checks a variable against multiple cases and executes the matching case block.

Let me know if you need more examples or further details on any specific part

Example 1: Determine if a Number is Even or Odd

Problem:

Write an algorithm to check if a number is even or odd.

Algorithm:

```
Start
Input a number (num)
If num % 2 == 0
    o  Output "The number is even"
Else
```

```
        o   Output "The number is odd"
    End
```

Example 2: Find the Largest of Three Numbers

Problem: Write an algorithm to find the largest of three numbers.

```
Algorithm:
  1. Start
  2. Input three numbers (a, b, c)
  3. If a > b and a > c
        o   Output "a is the largest"
  4. Else If b > a and b > c
        o   Output "b is the largest"
  5. Else
        o   Output "c is the largest"
  6. End
```

a.1. The "For Loop"

The **For Loop** is used when you know the exact number of times you want a piece of code to repeat. It is ideal for iterating over a sequence, such as a range of numbers.

Structure of a For Loop:

```
For (initialization; condition; update) {   // Code to execute       }
```

Explanation of Components:

- **Initialization**: Sets the starting value for the loop counter.
- **Condition**: Determines whether the loop should continue running. If the condition is true, the loop runs; if false, it stops.
- **Update**: Changes the counter variable (usually an increment or decrement) each time the loop executes.

Example of a For Loop: Print numbers from 1 to 5.

```
For (i = 1; i <= 5; i++) {        Print I        }
For (initialization; condition; increment/decrement)
```

```
Program sum
Real sum,s
```

```
Integer i,n
Read(*,) n
S=0
Do I = 1,n,1
Sum= sum+sqrt(i)
S=sum
Enddo
Write(*,*) sum
end
```

a 2. The "While Loop"

The **While Loop** is used when you want to repeat a block of code as long as a certain condition is true. Unlike the For Loop, it's generally used when the number of iterations is unknown.

Structure of a While Loop:

```
While (condition) {
// Code to execute}
```

Explanation of Components:

- **Condition**: Checked before each iteration. If the condition is true, the loop runs; if false, it stops.

Example of a While Loop: Print numbers from 1 to 5.

```
i = 1
While (i <= 5) {
Print i
i = i + 1
}
```

Explanation:

- Here, i starts at 1. The loop checks if i <= 5, and if true, prints i.
- After each iteration, i is incremented by 1.
- The loop stops when i exceeds 5.

a.3. The "Do-While Loop"

The **Do-While Loop** is similar to the While Loop but guarantees that the code block runs at least once because the condition is checked after the code block executes.

Structure of a Do-While Loop:

```
Do {   // Code to execute } While (condition);
```

Explanation of Components:

- **Code to execute**: The loop body is executed once before the condition is checked.

- **Condition**: Evaluated after each iteration. If true, the loop runs again; if false, it stops.

Example of a Do-While Loop: Print numbers from 1 to 5.

```
i = 1
Do {
Print i
i = i + 1
} While (i <= 5)
```

Explanation:

- Here, i starts at 1. The loop first prints i and increments it by 1.
- Then it checks if i <= 5. If true, the loop repeats.
- The loop stops when i exceeds 5.

Each of these iterative structures has its ideal use cases:

- Use a **For Loop** when you know the exact number of repetitions.
- Use a **While Loop** when the loop should continue until a specific condition is met.
- Use a **Do-While Loop** when the code should execute at least once regardless of the condition.

b: **Conditional Control Structures**

Conditional control structures enable algorithms to make decisions and execute specific blocks of code based on whether certain conditions are met. They are essential for implementing logic in programming and are commonly used in all high-level programming languages.

b .1 If-Else Statement

- **Definition:** The if-else statement allows the program to evaluate a condition. If the condition is true, one block of code is executed; if the condition is false, an alternative block is executed.
- **Syntax :**

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

Algorithm to Solve a Quadratic Equation

Problem:

Find the roots of the quadratic equation $ax^2+bx+c=$, where a, b, and c are constants, and $a\neq0$

Steps:

```
1. Start
2. Input the coefficients a, b, and c.
3. Calculate the Discriminant D=b**2−4*a*c
4. Check the value of the Discriminant (D):
    o If D>0:
        ▪ The equation has two distinct real roots.
        ▪ Calculate:
            ▪ x1== (−b+sqrt(D))/2a
            ▪ x2=(b+sqrt(D))/2a
        ▪ Output x1 and x2.
    o Else If D=0
        ▪ The equation has one real root (double root).
        ▪ Calculate:
        ▪ x==−b/2a
        ▪ Output x.
    o Else (If D<0D < 0D<0):
        ▪ The equation has two complex roots.
        ▪ Output two complex roots
5. End
```

Write and Run a Program (Flow Chart)

It determines the nature of the problem, prepares calculation methods, and organizes the required solution steps accurately. It is preferable to create a flow chart that accurately describes the method of solving the problem, and converts the flowchart into a program written in the Fortran language, and this program is the source program, and the calculator translates the source program and is done This is done by a special program known as the Fortran interpreter:

What is Flow Chart?

A flowchart is a diagram that depicts a process, system, or computer algorithm. They are widely used in multiple fields to document, study, and plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts, sometimes spelled as flow charts, use rectangles, ovals, diamonds, and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence. They can range from simple, hand-drawn charts to comprehensive computer-drawn diagrams depicting multiple steps and routes. If we consider all the various forms of flowcharts, they are one of the most common diagrams on the planet, used by both technical and non-technical people in numerous fields.

The Main Steps for Solving Problems

- ✓ 1-Identify and analyze the issue
- ✓ 2- Writing the algorithm
- ✓ 3- Draw the flowchart
- ✓ 4- Writing the program
- ✓ 5- Test the program
- ✓ 6- Documenting the program

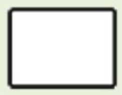
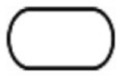
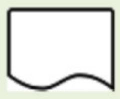
Flowchart Symbols

Symbols in Flowcharts:

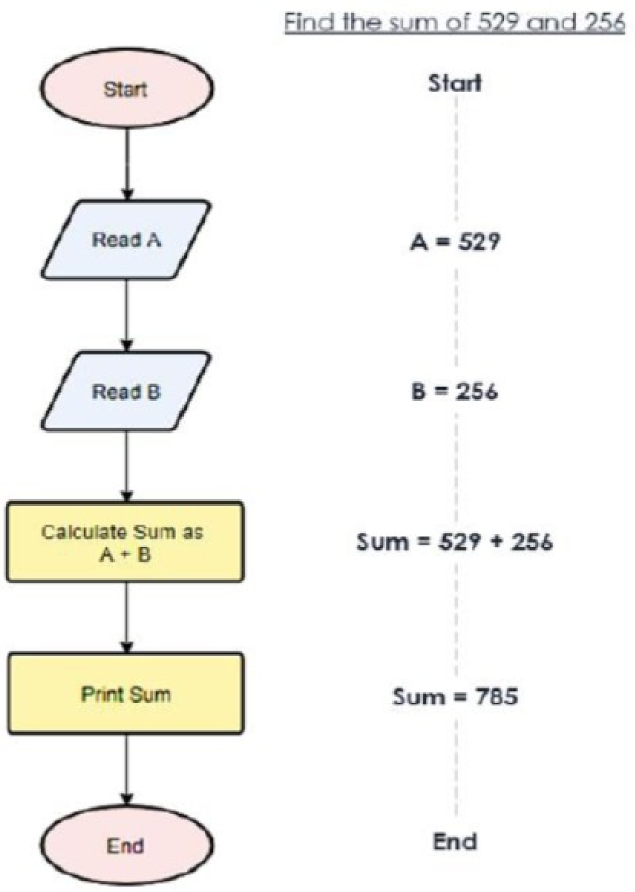
- **Oval:** Used for the *Start* and *End* of a process.
- **Rectangle:** Represents a *Process* or an action, such as calculations or assignments.
- **Parallelogram:** Used for *Input* or *Output*, where data is either received or displayed.
- **Diamond:** Denotes a *Decision* point, where a choice is made (e.g., yes/no or true/false).
- **Arrow (Connector):** Shows the *Flow of Control*, guiding the sequence from one step to the next.

These symbols make flowcharts highly effective for analyzing and designing complex systems or algorithms.

These flowchart shapes and symbols are some of the most common types you will find in most flowchart diagrams.

Flowchart Symbol	Name	Description
	Process symbol	Also known as, an "Action Symbol," this shape represents a process, action, or function. It is the most widely used symbol in flowcharting.
	Start/End symbol	Also known as, the "Terminator Symbol," this symbol represents the start points, ends, and potential outcomes of a path. Often contains "Start" or "End" within the shape.
	Document symbol	Represents the input or output of a document, specifically. Examples of and input are receiving a report, email, or order. Examples of output using a document symbol include generating a presentation, memo, or letter.

◇	Decision symbol	Indicates a question to be answered — usually yes/no or true/false. The flowchart path may then split off into different branches depending on the answer or consequences thereafter.
○	Connector symbol	Usually used within more complex charts, this symbol connects separate elements across one page.
▱	Input/output symbol	Also referred to as the "Data Symbol," this shape represents data that is available for input or output as well as representing resources used or generated. While the paper tape symbol also represents input/output, it is outdated and no longer in common use for flowchart diagramming.


Find the sum of 529 and 256
Start
Read A
Read B
Calculate Sum as A + B
Print Sum
End
Start
A = 529
B = 256
Sum = 529 + 256
Sum = 785
End