Chapter 3 :

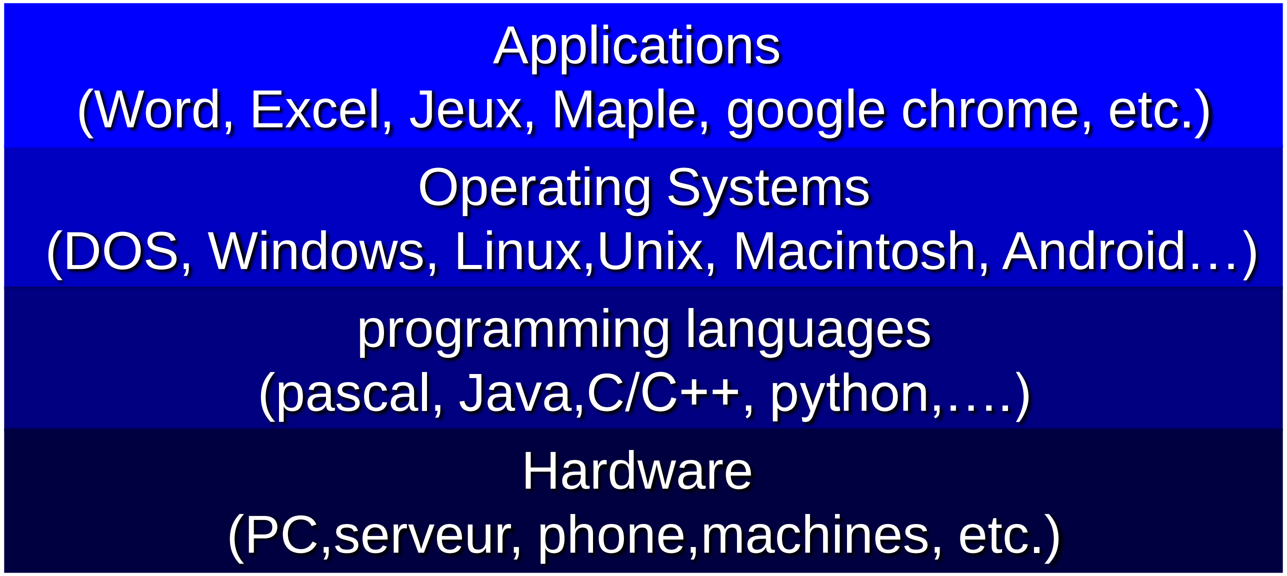
programming algorithm

Table of contents

- Intoduction
- General information about programs and software
- Algorithm Definition & Meaning
- Variables in an algorithm
- Expressions and operators in an algorithm
- Basic Instructions
- Building an Algorithm
- Conditional statements/control structures
- Iteration statements/Loop structures
- Activity 3

Introduction

- Computer system components



Applications
(Word, Excel, Jeux, Maple, google chrome, etc.)

Operating Systems
(DOS, Windows, Linux,Unix, Macintosh, Android…)

programming languages
(pascal, Java,C/C++, python,….)

Hardware
(PC,serveur, phone,machines, etc.)

General information about programs and software

Operating system OS

- **Process Management**: The OS manages processes, which are instances of executing programs. It allocates CPU time, memory, and other resources to different processes.
- **Memory Management**: The OS controls the allocation and deallocation of memory for various processes.
- **File System Management**: It provides a way to organize and store files and data on storage devices.
- **Device Management**: The OS handles communication with hardware devices.
- **User Interface**: Many operating systems provide user interfaces, including command-line interfaces (CLI) and graphical user interfaces (GUI).

General information about programs and software (1)

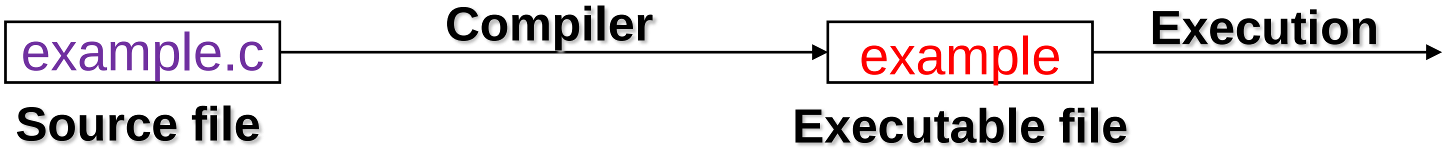
programming language

- They provide a way for humans to communicate with computers and give them instructions to perform specific tasks.
- There are many programming languages available, each with its own syntax, semantics, and areas of application.
- the most commonly used programming languages:
  - **Python**
  - **JavaScript**:.
  - **Java**:
  - **C++**:
  - **C**:
  - **C#**:

General information about programs and software (2)

Compiler

Is a software tool used in computer programming to transform source code written in a high-level programming language into a lower-level code or binary code that can be executed directly by a computer's central processing unit (CPU).

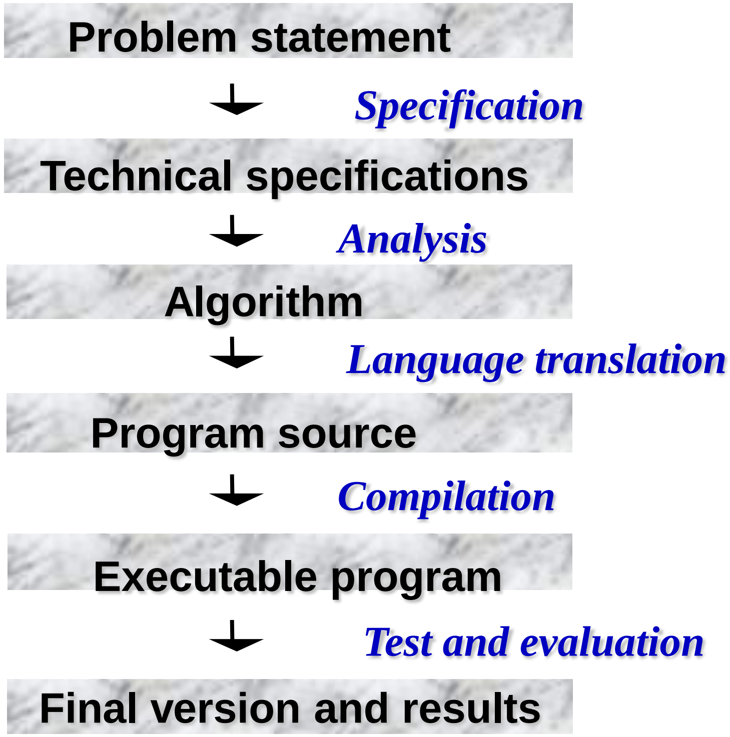


Interpreter vs. Compiler:

Interpreters execute code line by line, translating and executing each line as it is encountered, while compilers translate the entire source code into machine code or an intermediate representation before execution.

General information about programs and software (3)

Program



Problem statement

Specification

Technical specifications

Analysis

Algorithm

Language translation

Program source

Compilation

Executable program

Test and evaluation

Final version and results

Programming involves the writing of algorithms

⇒ This explains why algorithms matter.

Algorithm Definition & Meaning

The term "algorithm" is the technique of performing arithmetic with Arabic numerals developed by **al-Khwārizmī**.

An algorithm is a step-by-step procedure or a set of well-defined rules for solving a specific problem or completing a specific task.

Pseudo-code: To describe an algorithm, it's common to use a mix of natural language and pseudo-code, which is a high-level, human-readable representation of code that describes the steps without adhering to a specific programming language.

Variables in an algorithm

Variables are an essential concept in programming and algorithm design. They are used to store and manipulate data.

Data Storage: Variables are like containers that hold data. They can store various types of data, such as numbers, text, or more complex structures like arrays and objects.

Naming: Variables have names that programmers assign to them. These names are used to refer to the data stored in the variable.

Data Types: Variables have data types that determine the kind of data they can store. Common data types include integers, floating-point numbers, strings, and boolean values.

Declaration: Before using a variable, it must be declared. This informs the computer's memory system to allocate space for the variable. The syntax for declaring a variable varies from one programming language to another.

Variables in an algorithm (2)

Variables Name

Avoid Reserved Words: Don't use reserved words or keywords in the programming language as variable names. For example, in C, you can't use the name **for** it's a keyword used for loops.

Start with a Letter: Variable names should start with a letter (a-z, A-Z).

Alphanumeric and Underscores: Variable names can contain letters, numbers, and underscores. Avoid special characters and spaces.

Descriptive Names: Variable names should be chosen to be descriptive and indicative of the data they store. This makes the code more readable and understandable. For example, instead of using a generic name like a or x, use names like total_sum, user_avreag.

Variables in an algorithm (3)

Variables Type

In algorithms, variables can have different data types, which determine the kind of data they can store and how that data is manipulated.

- **Integer (int)**: Integer variables store whole numbers, both positive and negative, without fractional parts.
- **Floating-Point (float)**: Floating-point variables are used to store real numbers with decimal points.
- **Character(char)**: Character variables hold sequences of characters, such as text or words.
- **Boolean (bool)**: Boolean variables have only two possible values: True and False.
- **Array**: Arrays are used to store collections of elements of the same data type.
- **Pointers** : In low-level languages like C and C++, variables can also store memory addresses (pointers or references) to other data in memory.

Variables in an algorithm (4)

Variables designantion

- **Input Variable or Data:** refer to the values or information that a program receive from external sources, such as a user, other software/hardware components, or data files.

- **Output Variable or Results:** refer to the values, data, or information that a program or function generates or produces as a result of its execution.

- **Intermediate Variable :**Typically refers to a variable that is used to store an intermediate result or temporary value during a computation or operation.

Expressions and operators in an algorithm

In algorithms and programming, expressions and operators are fundamental concepts used for performing operations on data.

Expressions are combinations of values, variables, and operators that can be evaluated to produce a result. Operators are symbols or keywords that perform specific operations on the operands.

Expressions and operators in an algorithm (1)

Expressions

❑ **Arithmetic Expressions**: These expressions involve mathematical operations, such as addition, subtraction, multiplication, and division.

result = 5 + 3

area = length * width

average = (x + y) / 2

❑ **Relational Expressions**: Relational expressions are used to compare values and produce a Boolean result (True or False).

is_equal : x == 5

is_greater: a > b

❑ **Logical Expressions**: Logical expressions involve logical operators and are used for making decisions or combining conditions.

is_valid = (age >= 18) and (age<30)

❑ **Bitwise Expressions**: In low-level programming or when dealing with binary data, bitwise expressions can be used to manipulate individual bits within values.

a=3&7

Expressions and operators in an algorithm (2)

Operators

❑ **Arithmetic Operators**:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo, for finding the remainder)

❑ **Assignment Operator**:

- = (assigns a value to a variable)

❑ **Relational Operators** (used in comparisons):

- == (equal to)
- != (not equal to)
- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)

Expressions and operators in an algorithm (3)

Operators

❑ **Logical Operators** (used in logical expressions):

- && (logical AND)
- | |(logical OR)
- !! (logical NOT)

❑ **Bitwise Operators** (used for binary manipulation):

- & (bitwise AND)
- | (bitwise OR)
- ^ (bitwise XOR)
- ! (bitwise NOT)
- << (left shift)
- >> (right shift)

❑**Increment and Decrement Operators**:

- ++ (increment by 1)
- -- (decrement by 1)

Basic instructions

Assignment

- assignment is a fundamental operation that involves storing a value in a variable. It is used to initialize a variable, update its value, or modify the content of a data structure. Assignment is typically represented by the ‘=‘ operator, which assigns a value to a variable.

```
Example :             i =1            j =i           k =i+j
                     x =10,3                          ch1 = "SMI"
                     ch2 = ch1        x =4             x =j
```

Updating Variables: Variables can be updated by assigning new values to them. This is a common operation when variables need to change over time. The assignment replaces the current value with the new value.

Basic instructions (1)

Input Instructions (read())

Rading input is a common operation that allows a program to obtain data from an external source, typically a numerical date, a user, a file, or another system. **The specific way to read input depends on the programming language and the source of the input.**

In algorithm.

read(X) , X is a input/data variable

Basic instructions (2)

Output Instructions (write())

Is instruction refer to operations that output or display information, typically to the user via a screen, console, file, or another output medium. Writing instructions allow a program to communicate results, messages, and data to the user or store information for future reference. **The specific way to write output depends on the programming language and the output destination.**

In algorithm.

write(Y) , Y is a output/result variable

write(“Hello all”), displaying a message

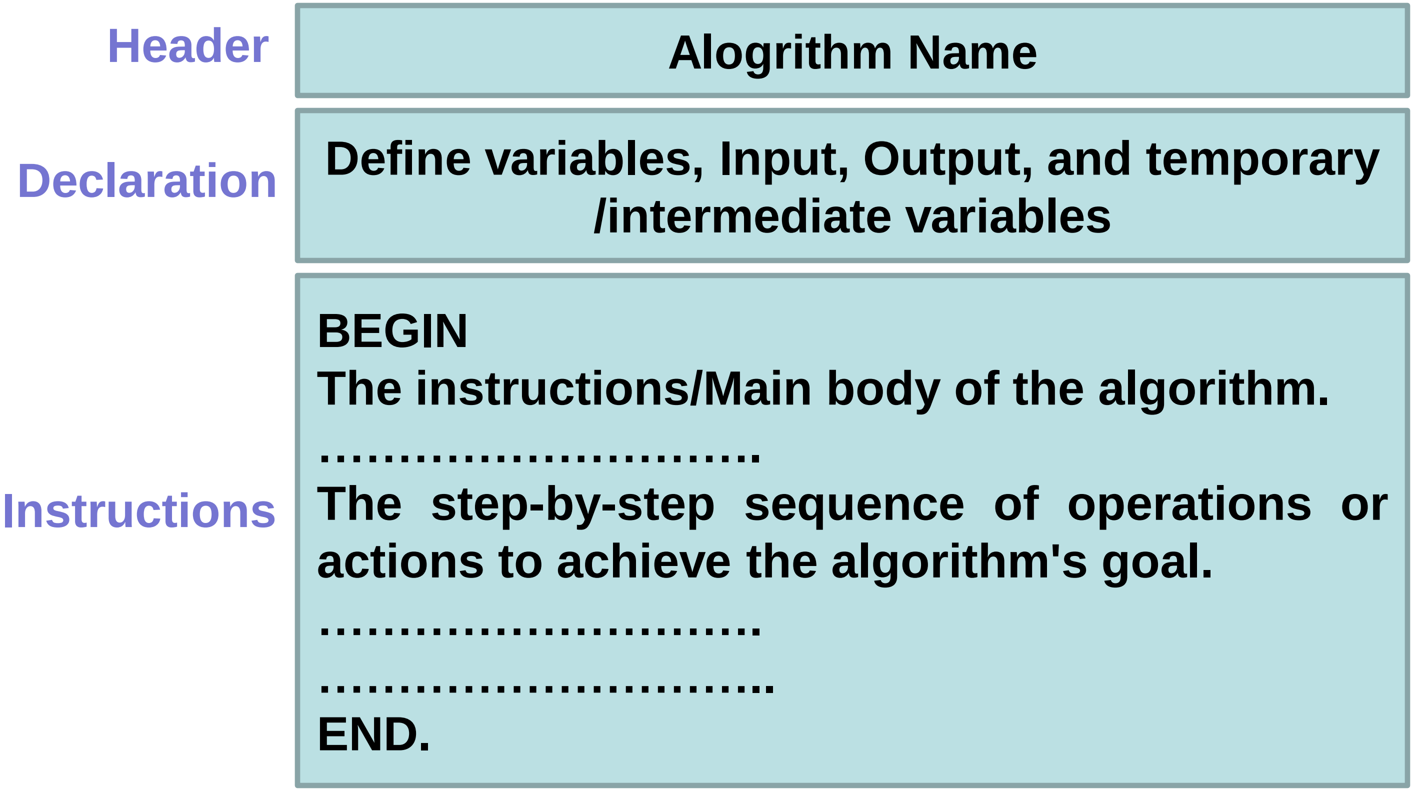
Basic instructions (3)

Example

```
Algorithm : Calcul_double
Input variable, A : integer
 Output variable, B : integer
BEGIN
            read (A)
            B = 2*A
            write(B)
END.
```

Building an Algorithm

It appears you're describing the general structure of an algorithm as consisting of three main parts: the header, declaration, and block of instructions.



Header

Alogrithm Name

Declaration

Define variables, Input, Output, and temporary /intermediate variables

Instructions

BEGIN
The instructions/Main body of the algorithm.
………………………..
The step-by-step sequence of operations or actions to achieve the algorithm's goal.
………………………..
…………………………..
END.

Building an Algorithm

My First Algorithm

Name: sum of two integer

```
Name: sum of two integer
Input Variables: A,B integer
Output Variable: C integer
intermediate variables:______________
BEGIN
        A=23
        B=10
        C=A+B
        Write (C)
END.
```

Building an Algorithm

My First Algorithm

```
Name: sum of two integer (best solution)
Input Variables: A,B integer
Output Variable: C integer
intermediate variables:______________
BEGIN
        read(A)
        read(B)
        C=A+B
        Write (C)
END.
```

Conditional statements/control structures

- Conditional instructions or control structures, are a fundamental part of programming and algorithm design.
- They allow you to make decisions and execute different blocks of code based on specified conditions.
- Conditional statements enable your program to take different actions depending on whether a condition is true or false.

There are typically several types of conditional statements in algorithms:

- ❑ **if Statements**
- ❑ **If-else Statements**
- ❑ **else-if Statements**
- ❑ **Switch Statements**

Conditional statements/control structures

if statements

An "if" statement is used to execute a block of code if a condition is true.

Example:

```
Name: Sum Positive number
Input Variables: A,B integer
Output Variable: C integer
BEGIN
        read(A)
        read(B)
        if(A>0) begin_if
                C=A+B
                end_if
        Write (C)
END.
```

Conditional statements/control structures

if-else statements

An "if-else" statement allows to execute one block of code if a condition is true and another block if the same condition is false. **Example:**

```
Name: Sum Positive number
Input Variables: A,B integer
Output Variable: C integer
BEGIN
          read(A),read(B)
          if(A>0) begin_if
                    C=A+B
                    end_if
          else begin_else
                    C=A-B
                end_else
          Write (C)
END.
```

Conditional statements/control structures

else-if statements

These are used when you have multiple conditions to check and execute different code blocks based on which condition is true.**Example:**

```
Name: Sum Positive number
Input Variables: A,B integer
Output Variable: C integer
BEGIN
        read(A),read(B)
        if(A>0) begin_if
                C=A+B
                end_if
        else begin_else
                if(A==0) begin_if
                        C=A*B
                        end_if
                else begin_else
                        C=A-B
                        end_else
                end_else
        Write (C)
END.
```

Conditional statements/control structures

Switch Statements

hese are used to select one of many code blocks to execute, based on the value of an expression.**Example:**

```
Name: Sum Positive number
Input Variables: A,B integer
Output Variable: C integer
BEGIN
        read(A),read(B)
        switch begin_switch
                case (A>0): C=A+B
                case (A<0): C=A-B
                default: C=A*B
               end_switch
        Write (C)
END.
```

Conditional statements/control structures

Algorithm example 1

Write an algorithm that asks the user for an integer and tests whether it's divisible by 3:

```
Name: Divisible by 3
Input Variables: A integer
Output Variable: ___________
BEGIN
             read(A)

              if(A%3==0) begin_if
                          write("The entered integer is divisible by 3.“)
                          end_if
             else begin_else
                           write(" The entered integer is not divisible by 3.“)
                     end_else
END.
```

Conditional statements/control structures

Algorithm example 2

Name: Positive_negative_null
Input Variables: A integer
Output Variable: ____________

```
BEGIN
        read(A)
        if(A>0) begin_if
                 write("The entered integer is positive.“)
                end_if
        else begin_else
                if(A<0) begin_if
                         write("The entered integer is negative.“)
                        end_if
                else begin_else
                        write("The entered integer is zero.“)
                        end_else
                end_else
END.
```

Conditional statements/control structures

Algorithm example 3

```
Name: Calculator
Input Variables: A,B integer, op char
Output Variable: C
BEGIN
        read(A), read(B), read(op)
        switch begin_switch
                case (op=‘+’): C=A+B
                case (op=‘-’): C=A-B
                case (op=‘*’): C=A*B
                case (op=‘/’): if(B==0) begin_if
                                              write(“error“)
                                                end_if
                                        else begin_else
                                             C=A/B
                                             end_else
               default: write(“error“)
               end_switch
        Write (C)
END
```

Iteration statement/Loop structures

Loops or iteration constructs, are fundamental components of programming that allow you to repeat a block of code multiple times. Iteration is essential for automating repetitive tasks, processing collections of data, and implementing algorithms that involve repeating actions.

- **For Loop**
- **While Loop**
- **Repeat Loop**

Iteration statement/Loop structures(1)

For Loop

A for loop is a fundamental control structure in programming that allows you to execute a block of code a specific number of times. It is commonly used when you know in advance how many times you want to repeat a certain operation.

```
for (initialization/start; stop; iteration/step)
```

Example:

```
intermediate variables:i integer
BEGIN
For (i=0 to 10 with step 1)
          begin_for
            instructions
           end_for
END.
```

Iteration statement/Loop structures(2)

For Loop

- Calculating x to the n power, where x is a real number and n is a positive integer.

```
Name: Calculate power
Input Variables: x,n integer
Output Variable: P integer
 intermediate variables: i integer
BEGIN
        write(" Enter the value of x ");
        write(x);
        écrire (" Enter the value of n ");
        read(n);
        P=1;
        For (i=1 to n with step 1) begin_for
        P= P* x ;
        end_for
        write(P);
END.
```

Iteration statement/Loop structures(3)

While Loop

while loop is a control structure in programming that allows you to repeatedly execute a block of code as long as a specified condition remains true. It continues iterating as long as the condition is true.

```
while(condition) begin_while
           Instructions
                  end_while
```

Example:

```
intermediate variables:i integer
BEGIN
i=0
while(i<=10) begin_while
           instructions
            i=i+1
          end_while
END.
```

Iteration statement/Loop structures(4)

While Loop

- Calculating x to the n power, where x is a real number and n is a positive integer.

```
Name: Calculate power
Input Variables: x,n integer
Output Variable: P integer
 intermediate variables: i integer
BEGIN
        write(" Enter the value of x ");
        write(x);
        écrire (" Enter the value of n ");
        read(n);
        P=1; i=1
        while(i<10) begin_while
        P= P* x ;
        end_while
        write(P);
END.
```

Iteration statement/Loop structures(3)

Repeat Loop

That is similar to a "while" loop, but it guarantees that the code block will be executed at least once, even if the condition is initially false.

```
Repeat begin_repeat
    Instructions
while(condition) end_repeat
```

Example:

```
intermediate variables:i integer
BEGIN
i=0
Repeat begin_repeat
          instructions
          i=i+1
          while(i<=10) end_repeat
END.
```

Iteration statement/Loop structures(5)

Repeat Loop

- Calculating x to the n power, where x is a real number and n is a positive integer.

```
Name: Calculate power
Input Variables: x,n integer
Output Variable: P integer
 intermediate variables: i integer
BEGIN
        write(" Enter the value of x ");
        write(x);
        écrire (" Enter the value of n ");
        read(n);
        P=1; i=1
        Repeat begin_repeat
        P= P* x ;
         while(i<10) end_repeat
        write(P);
END.
```

Activity chapter 3

Write an algorithm that prompts the user to enter any time in the form of three variables: Hour, Minute, and Second, and then displays a message indicating whether the time entered is valid or not. (assuming all three variables are non-negative).

Examples :
Hour =**28**   Minute = **31**   Second =**51**   is an invalid time.
Hour = **16**   Minute= **3**   Second = **55**   is a valid time.

Algorithm

Name: validated time

Input variables: m, h, s : Integer

Output variables:

Intermediate variables :

```
BEGIN
   read(h) ;
   read(m) ;
   read(s) ;
if(h<24 ) begin-if
          if(m<60) begin-if
                         if(s<60) begin-if
                         Ecrire ("is a valid time ")
                          end_si
                         else begin_else
                         Ecrire("is a valid time ")
                         end_else
           end_if
           else begin-if
           Ecrire("is a valid time ")
           end-if
end-if
else begin-if
           Ecrire("is an invalid time") ;
end-if
END.
```