# COURSE OF ALGORITHMS AND DATA STRUCTURE CHAPTER 1and 2



**L1 Mathematics**

**Semester 1**

**Prepared by ; Dr MAHMA OUAFA**

**2025/2026**

# CHAPTER 1-2

1. Brief history of computer science
2. Introduction to algorithms
3. Simple sequential algorithm

# History of computer science

- Computer science is the science that deals with the automatic processing of information using machines, specifically computers.
- Science: refers to the sum of theories and techniques.
- Automatic processing: is done by using precise and repeatable methods.
- Information: is the basic element of knowledge, which can be stored, processed and transformed.
- The computer basically consists of two parts: the hardware part and the software part

# History of computer science

- The discipline of computer science includes the study of algorithms and data structures, computer and network design, modeling data and information processes, and artificial intelligence.
- Computer science draws some of its foundations from mathematics and engineering and therefore incorporates techniques from areas such as probability and statistics, and electronic circuit design.
- Computer science also makes heavy use of hypothesis testing and experimentation during the conceptualization, design, measurement, and refinement of new algorithms, information structures, and computer architectures.

# History of computer science

- The history of computers goes back over 200 years.
- At first theorized by mathematicians and entrepreneurs, during the 19th century mechanical calculating machines were designed and built to solve the increasingly complex number-crunching challenges.
- The advancement of technology enabled ever more-complex computers by the early 20th century, and computers became larger and more powerful.
- the history of the computer science is related to the history of the computer basically.

# Introduction to algorithms

**Question** :

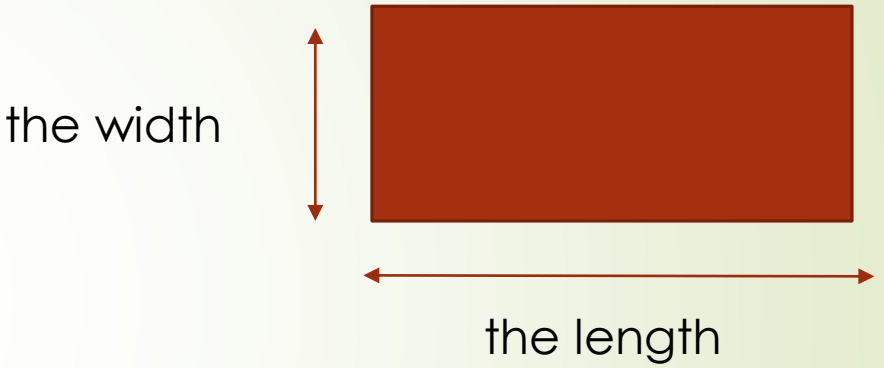
Let's assume that a student asked you,

***How to calculate the area of a rectangle?***

- How would you explain to him what must be done to calculate this area?

# Introduction to algorithms

***Answer***:




The following steps must be followed :

1. Knowing the length of the rectangle
2. Knowing the width of the rectangle
3. Calculating the area by multiplying the length by the width
4. Announcing and displaying the result (area of the rectangle)

These steps are called ***Algorithm***.

# Introduction to algorithms

**Question** :

Let's assume that we wrote an algorithm to solve a problem, can the computer understand the steps of the algorithm and implement it?

# Introduction to algorithms

**Answer** :

Before we answer this question, let's assume that we have two people, the first is an Arab who only speaks Arabic and the second is an Englishman who only speaks English.

The Arab person wrote a text in Arabic and sent it to the English person. It is clear that the English person will not be able to understand the text written in Arabic.

# Introduction to algorithms

**Answer** :

Likewise, the computer cannot understand the steps of the algorithm because they are written in a language it does not understand.

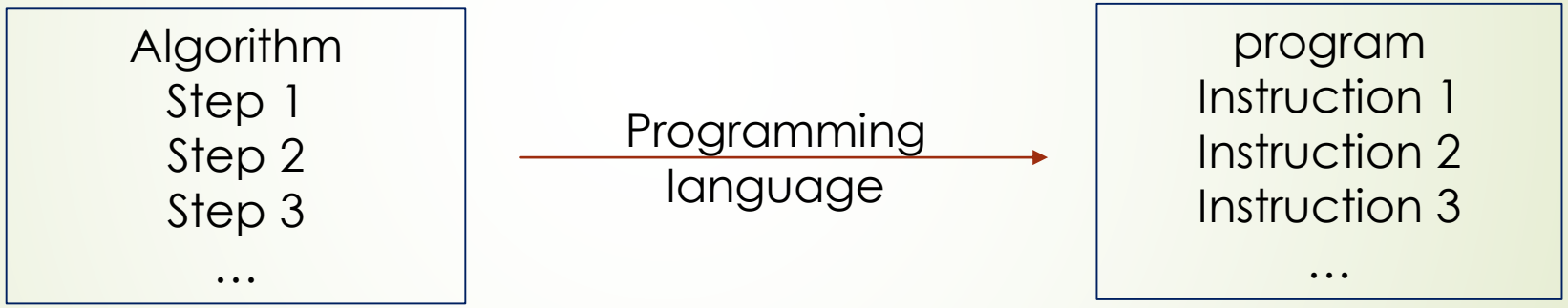
In order for the computer to be able to understand these steps and implement them, the algorithm must first be translated into one of the programming languages Python, C++ , C JAVA, Pascal....

After that, the computer itself translates the program into the language it directly understands (machine language)

# CHAPTER 2

# Algorithms ‘sequential algorithm’

All program is a result of translating the algorithm into instructions directed to be executed using the computer.



# Algorithms ‘sequential algorithm’

**Algorithm properties :** The algorithm has the following characteristics

1. ***Generality*** **:** we note that the previous steps are suitable for calculating the area of any rectangle .
2. ***Inputs:*** The number and nature of inputs vary from one algorithm to another. The inputs in the previous example are length and width.
3. ***Outputs:*** The results of executing the algorithm. The outputs in the previous example are area.
4. ***Clarity (the opposite of vague):*** Each step of the algorithm must be clear and unambiguous, meaning that each step must be clear in a way that allows it to be executed correctly.
5. ***Solvability:*** It is possible for a certain step in the algorithm to be clear (unambiguous) but not solvable. For example, we cannot find $\sqrt{-4}$.
6. ***Time limitation:*** It is possible for a certain step in the algorithm to be clear (unambiguous) and solvable but not solvable in a specific time frame. For example, calculating the final value of the number π.

# Algorithms ‘sequential algorithm’

- **Machine language (binary):** This is what is stored on your disks (the .exe for example), and brought into memory to be executed, is a sequence of 1 and 0 (eg: 101000100).
- For a human being, writing sequences of 0s and 1s is not very fun, not very readable, and causes many errors.
- So they invented **programming languages**,

# Algorithms ‘sequential algorithm’

How does the machine "**understand"** a programming language?

You need a program that translates the text of your program into binary code, we are dealing with **a compiler**,

A **computer program** is a list of commands that tell a computer what to do. It is presented in the form of one or **more sequences of instructions**

# Algorithms ‘sequential algorithm’

So ,

- An algorithm is a method for solving a particular problem.
- A programming language that describes an algorithm
- Program is Séquence ***of instructions*** which specifies step by step the operations to be carried out to obtain a result.

# Algorithms ‘sequential algorithm’

- Algorithm Structure ( General form )

```
Algorithm          Title
Variables          ……….
Begin
Instruction 1
Instruction 2
Instruction 3
 ……….
End
```

# Algorithms ‘sequential algorithm’

After taking a general overview of what an algorithm, a program, and programming languages , we return below to detail the basic elements that make up an algorithm.

# Algorithms ‘sequential algorithm’

- Basic instructions in the algorithm

1. Assignment process : Give a value to a variable **A** ← **5**

   It means giving the variable **A** a numerical value 5 .

2. Reading process: A process in which the user assigns a value to a variable via the keyboard. We express it with the keyword “ **READ**”

   **read (A)**

It means read the value that the user will enter and put it in the variable **A** .

# Algorithms ‘sequential algorithm’

3. Writing process:

It is the process that shows the value of a variable by printing it on the screen.

It also prints text sentences on the screen and expresses it using the keyword (**write**) , exemples :

**A ← 5**

**Write(A)**

It means showing the value of the variable **A** on the screen

**Write (‘I am an IM student’)**

It means showing the sentence between the two signs on the screen

# Algorithms ‘sequential algorithm’

- Example An algorithm to solve a first -class equation

**Algorithm** Eq1;

**Variables** a, b: Integer**;**

X: **real;**

**Begin**

Read(a);

Read(b);

X ← (-1*b) / a;

write(X);

**end**

# Algorithms ‘sequential algorithm’

Algorithm Elements : Variables

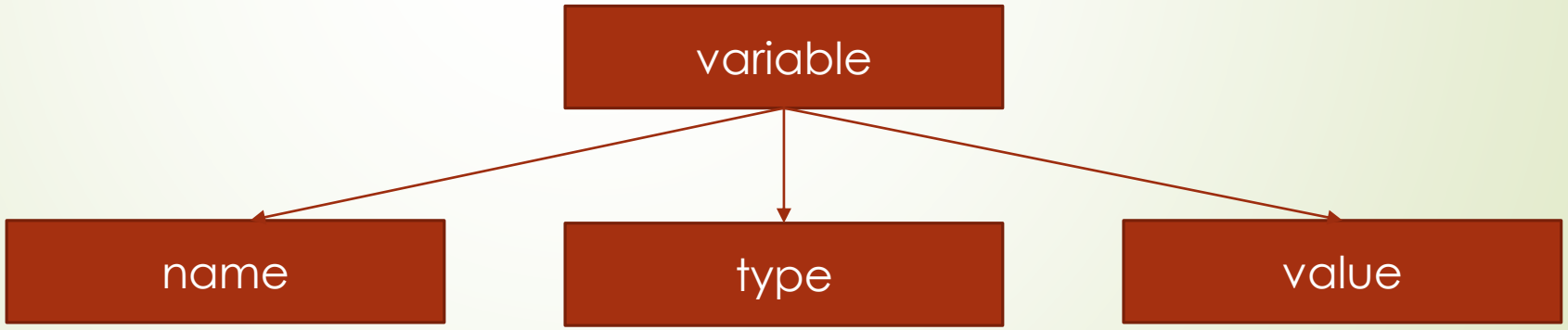
A variable is used to store the data needed by the algorithm. It is simply called a variable because its value changes.

# Algorithms ‘sequential algorithm’

Example

In the algorithm for calculating the area of a rectangle, we took the length and width as variables because the length and width differ from one rectangle to another.

The variable is characterized by 3 properties: name, type and value



# Algorithms ‘sequential algorithm’

- Example:

A: integer;

A ← 11;

A is an integer variable with a value of 11.

# Algorithms ‘sequential algorithm’

**Types of variables:**

Variables are divided into 3 types:

- Numeric
- Character
- Boolean

# Algorithms ‘sequential algorithm’

1. **Numerical variables:**

They are divided into two sections:

- **Integer variables** (Entier صحيحة ): They represent variables whose values belong to the set of integers, for example ..... -5.... -3....0.....1....2....10.....
- **Real variables** (Réel حقيقية ): They represent variables whose values belong to the set of real numbers, for example ..... -3,7 .....2,1....5, 6....

# Algorithms ‘sequential algorithm’

- Exemple

What are the final values of variables A, B, C after executing the algorithm?

```
Algorithm Test;
Variables  A,B,C: integer;
Begin
A ← 1;
B ← A + 5;
A ← B;
C ← A+B;
B ← C+A;
A ← 8;
End
```

# Algorithms ‘sequential algorithm’

## 2. Character variables

They are divided into two sections:

- **Single-character variables (**Caractère): are variables whose value is one character, and the character is either a number, an alphabetical letter, or a symbol. For example ....’A’.. ’ #’.....’2’.....
- **String variables** (Chaine de Caractères): represent variables whose value is a set of characters, for example .....’’783#’’.....’’148A’’……’’Ali45’’

# Algorithms ‘sequential algorithm’

3. **Boolean variables**

They are variables that take only two values,

- **true (Vrai)**
- **false (Faux)**.

# Algorithms ‘sequential algorithm’

- Exercise : What are the final values of variables A, B, C after executing the algorithm?

**Algorithm** Test;
**Variables** A,B,C: **boolean;**
**Begin**
A ← true;
B ← false;
A ← 5>2;
C ← B;
**End**

# Algorithms ‘sequential algorithm’

- Exercise: Write an algorithm that reads the student’s name, then his grades in two subjects, and then displays his name and grade point average.
- تمرين : أكتب خوارزمية تقوم بقراءة اسم الطالب ثم نقاطه في مادتين ثم تقوم بعد ذلك بإظهار اسمه و معدله

# Algorithms ‘sequential algorithm’

**Algorithm** Moyenne;
**Variables** Note1, Note2: **Integer;**
Nom: **String ;** Moy: **Real;**
**Begin**
Read (Nom);
Read (Note1);
Read (Note2);
Moy ← (Note1+Note2) / 2;
Write (Nom);
Write (Moy);
**End**

# Algorithms ‘sequential algorithm’

Algorithm Elements : Constants

Constants are the opposite of variables in that their value is known at the beginning of the algorithm and remains constant throughout execution( تبقي ثابتة طوال التنفيذ).

Example : The number π is considered a constant in the algorithm for calculating the area of a circle and its declaration is as follows:

# Algorithms ‘sequential algorithm’

**Algorithm** Cercle;
**Const** Pi=3.14;
**Variables** R,S: **integer;**
**Begin**
Read (R);
S ← Pi * R * R;
Write (S);
**End**

# Algorithms ‘sequential algorithm’

- **Question**:

Why do we need to declare the number as a constant at the beginning of the algorithm when we can use this number without declaring it as a constant?

- **Answer**:

If the constant is used in multiple places in the algorithm, declaring the constant makes the modification process easy.

# Algorithms ‘sequential algorithm’

- Example: Let the algorithm calculate the average of two students in two different subjects where the coefficient of the first subject is 3 and the coefficient of the second subject is 5

```
/*كتابة الخوارزمية مع التصريـــح بالثـــــوابـــــت */
Algorithm Moy2;
Const  Coff1=3; Coff2=5;
Variables  Note1,Note2: integer ; Moy: Real;
Begin
Read (Note1,Note2);
M ← ( (Note1 x Coff1) + (Note2 x Coff2) ) / (Coff1+ Coff2);
Write (M);
Read (Note1,Note2);
M ← ( (Note1 x Coff1) + (Note2 x Coff2) ) / (Coff1+ Coff2);
Write (M);
End
```

# Algorithms ‘sequential algorithm’

/*كتابة الخوارزمية من دون التصريح بالثـــــوابـــــت */

**Algorithm** Moy2;
**Variables** Note1,Note2: **integer ;** Moy: **Real;**
**Begin**
Read (Note1,Note2);
M ← ( (Note1 x 3) + (Note2 x 5) ) / (3+ 5);
Write (M);
Read (Note1,Note2);
M ← ( (Note1 x 3) + (Note2 x 5) ) / (3+ 5);
Write (M);
**End**

# Algorithms ‘sequential algorithm’

Suppose the material coefficients are changed to 1 and 2 respectively, how do we modify the two algorithms?

- In the first algorithm where we declared the constants, we change in one place, which is the line where we declared the constants Coff1 and Coff2

Const Coff1=3; Coff2=5;

It becomes

Const Coff1=1; Coff2=2;

# Algorithms ‘sequential algorithm’

- As for the second algorithm where we did not declare the constants, we will have to change it at the level of all the lines where we used the operators.

For example, M ← ( (Note1 x 3) + (Note2 x 5) ) / (3+ 5);

becomes

M ← ( (Note1 x 1) + (Note2 x 2) ) / (1+ 2);

And so on for all the lines where we used the operators

(هكذا بالنسبة لجميع الأسطر التي استخدمنا فيها المعاملات).

# Algorithms ‘sequential algorithm’

Algorithm Elements: Operators

Operators are what we can use to perform operations on variables.

1. **Arithmetic operators:**

We can place them between numerical variables and they are addition (+), subtraction (-), multiplication (x), division (/), integer division (Div), exponent (^) and remainder (mod).

Example: 5 mod 2=1(باقي القسمة الصحيحة)  11 mod 3= 2  12 Div 5 =2( القسمة الصحيحة)

# Algorithms ‘sequential algorithm’

- Priority of arithmetic operations:

exponents and parentheses, then multiplication and division, then addition and subtraction.

الأولوية : تكون للأسس و الأقواس ثم الضرب و القسمة ثم الجمع و الطرح

Example: What are the final values of variables A, B, C after executing the algorithm?

# Algorithms ‘sequential algorithm’

**Algorithm** Test;
**Variables** A,B,C: **integer ;**
**Begin**
A ← 1 ;
B ← (A + 5) x A^2;
A ← B-A*(5 mod A);
C ← A+B-3;
C ← C+A^(B+B);
**End**

# Algorithms ‘sequential algorithm’

Algorithm Elements: Comparison operators

- Equality (=)
- Inequality (<>)
- Greater than or equal to (>=)
- Less than or equal to (<=)
- Exactly greater than (>)
- Exactly less than (<)

# Algorithms ‘sequential algorithm’

Algorithm Elements: Logical operators

They are placed between logical variables and are represented by the operator (ET), the operator (OU), the operator (NON), or the operator (XOR).
Truth tables:

| A | B | A ET B |
|---|---|---|
| V | V | V |
| V | F | F |
| F | V | F |
| F | F | F |

| A | B | A OU B |
|---|---|---|
| V | V | V |
| V | F | V |
| F | V | V |
| F | F | F |

| A | B | A XOR B |
|---|---|---|
| V | V | F |
| V | F | V |
| F | V | V |
| F | F | F |

| A | NON A |
|---|---|
| V | F |
| F | V |

# CONSTRUCTING OF SIMPLE ALGORITHM with flowchart (Graphical method)

# algorithms 'sequential algorithm'

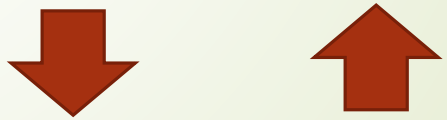
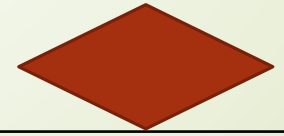
We mentioned earlier that the algorithm consists of two main parts: the ***declarations*** or ***data part*** and the other is the ***body*** of the algorithm.

1. As for the **data part**, it contains the **declarations of variables** and **constants**.
2. As for **the body part** , it is the **processing** part that contains the basic operations: **assignment** operations and **the input** and **output operations** that **allow the sequential execution** of the algorithm so that the operations are executed one after the other. That is why the algorithm here is called **a sequential algorithm.**

# Graphical method (flowchart)

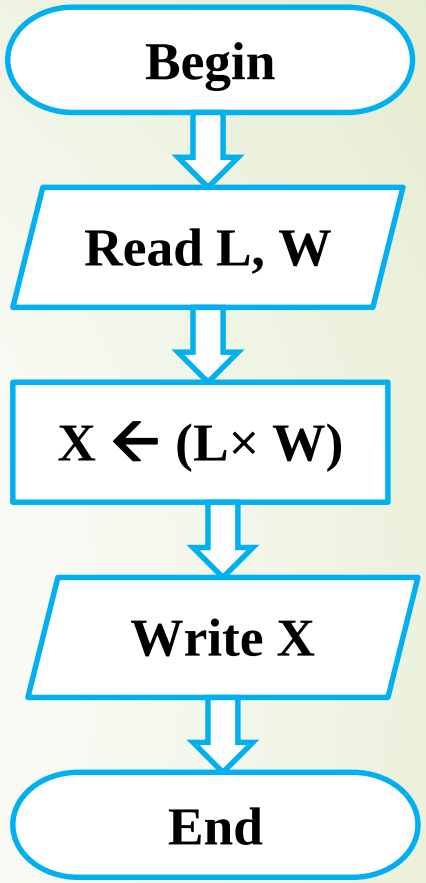
We can formulate the algorithm using the forms shown in the following table:



| | |
|---|---|
| Begin / end |  |
| Output / input |  |
| Calculate operations / assignment |  |
| Program direction |  |
| Conditional Formula |  |

# *How to calculate the area of a rectangle?*

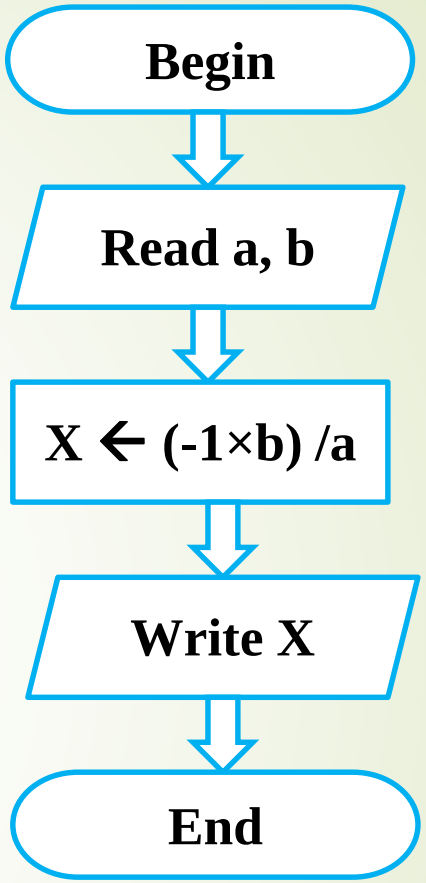
**Algorithm** rectangle;
**Variables** X, W, X: **real** ;
**Begin**
Read(L);
Read(W);
X ⟵ ( L *W);
write(X);
**end**




Algorithm to to calculate the area of a rectangle

# *How to solve a first -class equation ?*

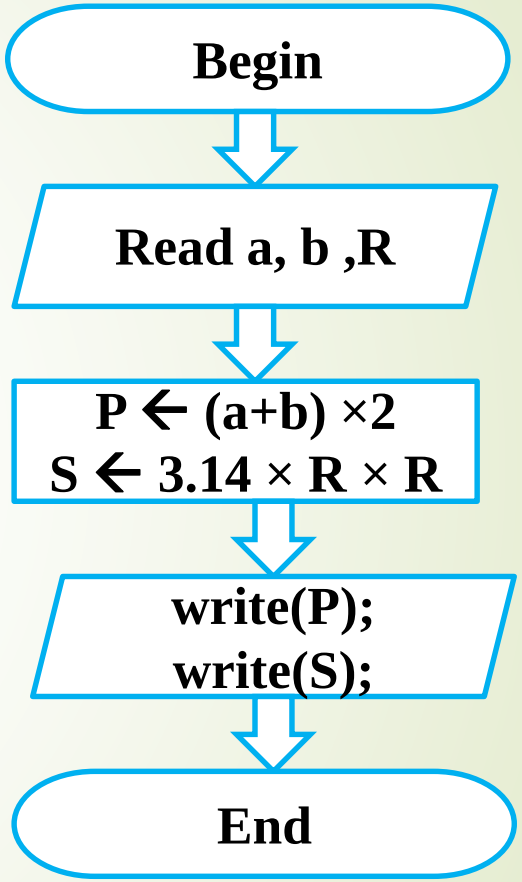
```
Algorithm    Eq1;
Variables a, b:
Integer;
X: real;
Begin
Read(a);
Read(b);
X  ←  (-1*b) / a;
write(X);
end
```




# Algorithm to solve a first -class equation

# *How to calculate the perimeter of a rectangle and the area of a circle?*

**Algorithm** Rectangle-Circle;
**Variables** a, b, R,P,S : Integer ;
**Begin**
Read(a);
Read(b);
Read(R)
P ← (a+b) ×2;
S ← 3.14 × R × R;
write(P);
write(S);
**end**




Algorithm to calculate the perimeter of a rectangle and the area of a circle

# CHAPTER 3

# CHAPTER 3 CONDITIONAL STRUCTURE

1. INTRODUCTION
2. SIMPLE CONDITIONAL STRUCTURE
3. COMPOUND CONDITIONAL STRUCTURE
4. MULTIPLE CHOICE CONDITIONAL STRUCTURE
5. BRANCHING

# INTRODUCTION

Write an algorithm that reads two numbers A and B and returns the result of dividing A by B?

- The solution :

  First :

  What are the inputs and outputs for this algorithm?

  - Algorithm inputs: A, B
  - Algorithm outputs:  A/B

# INTRODUCTION

Secondly :

**Algorithm** Dividing ;
**Variables**
A,B: **integer ;**
C**: Real;**
**Begin**
Read (A);
Read (B);
C ← A / B;
Write (C);
**End**

# INTRODUCTION

- Thirdly :the final value of all instructions in body of the algorithm after the executing ?

| | |
|---|---|
| READ (A) | 15 |
| READ (B) | 3 |
| C←A/B | 5 |
| WRITE C | 5 |

# INTRODUCTION

- Question: What we do if the value of B is equal to 0?
  - It is known that division by the number 0 is not possible.
  - So how do we change the algorithm in this case to avoid making mistakes?
  - The value of B must be different from 0. In other words, we must stipulate that the value of B must be different from 0.
  - To do this, we need the **conditional formula.**

# Simple structure of the conditional formula:

If <**Logic expression**> Then

    Instructions

Else

    Instructions

Endif

# Simple structure of the conditional formula:

- Modification of previous algorithm

**Algorithm** Dividing ;
**Variables**
A,B: **integer ;**
C**: Real;**
**Begin**
Read (A);
Read (B);
If (B=0) then
Write ( b must be different 0) ;
Else
C ← A / B;
Write (C);
**Endif** ;
**End**

# Simple structure of the conditional formula:

- Write an algorithm that reads a number A and then tells us whether it is negative or positive?

```
Algorithm Posit_Negat;
Variables A : integer ;
Begin
read(A);
if (A>0) then
Write (‘ A is positive ‘);
Else
Write (‘ A is Négative‘);
endif;
end.
```

# Simple structure of the conditional formula:

The previous algorithm takes into account negative and positive numbers but does not take into account zero.

What will be the output of the algorithm if **A=0**?

Answer: ????

So we need to add a third condition to handle the case of **A=0**

# Simple structure of the conditional formula:

- Modification of previous algorithm

```
Algorithm Posit_Negat;
Variables A : integer ;
Begin
 Lire(A);
if (A>0) then
      Write (‘ A is positive ‘);
else
      if  (A>0) then
            Write (‘ A is Négative‘);
       else
            Write (‘ A is Null‘);
endif;
end.
```

# Compound conditional structure :

- Write an algorithm that reads 3 numbers A, B, C and tells us whether they are arranged in ascending order or not?

  We can write the conditional formula in this case in two ways:

# Compound conditional structure :

- The first method

```
Algorithm sort ;
Variables A ,B,C: integer ;
begin
Read (A,B,C);
if  (A<B) then
    if (B<C) then
        Write (' The numbers are in ascending order ');
          endif;
else
            write (' The numbers are not in ascending order ');
        endif ;
End .
```

**Note: (else) follows the nearest (if) above it**

# Compound conditional structure :

- The second method

```
Algorithm sort ;
Variables A ,B,C: integer ;
Begin
read(A,B,C);
If  ( (A>B) and  (B>C) )then
        Write (‘ The numbers are in ascending order ‘);
Else
        write (‘ The numbers are not in ascending order ‘);
endif ;
end.
```

# Compound conditional structure :

Write an algorithm that reads 3 (unequal) numbers A, B, C and then arranges them in descending order.

- Method 1:

```
Algorithm SORT ;
Variables A ,B,C: integer;
Begin
read(A,B,C);
If  ( (A>B) and  (B>C) ) then
Write (A,B,C);
Else
if  ( (A>B) and  (C>B) ) then
Write (A,C,B);
Else
if  ( (B>A) and  (A>C) ) then
Write (B,A,C);
Else
if ( (B>C) and  (C>A) )
Write (B,C,A);
Else if ( (C>A) and (A>B) )
write(C,A,B);
Else
Write (C,B,A);
Endif ;
Endif;
Endif;
Endif;
Endif;
End .
```

# Compound conditional structure :

Exercise: Suppose the password for an application is "**ST2018**". Write an algorithm that reads a string and tells us whether the entered string matches the password or not.

**Algorithm** Password;

**Variables :** word,Pass:String ;

**Begin** Pass ⟵ "ST2018";

Read (word);

**if** (Pass = word) **then**

Write ('Identical password…');

**Else**

**Write** ('Wrong password…');

**Endif** ;

**End** .

# Multiple choice conditional structure

Write an algorithm that reads two numbers, then reads a third number.

If the third number is equal to 1, the algorithm adds the two numbers.

If the third number is equal to 2, the algorithm subtracts the two numbers.

If the third number is equal to 3, the algorithm multiplies the two numbers.

If the third number is equal to 4, the algorithm divides the first number by the second.

**Algorithme** Test_ACCORDING TO;

**Variables** A ,B,C : **Integer ;** D: **Real;**

**Begin**

Read (A,B,C);

Case of C

1: D ⟵ A +B; write (D);

2: D ⟵ A -B; write (D);

3: D ⟵ A xB; write (D);

4: D ⟵ A /B; write (D) ;

Else : write ('The operation code is wrong');

**End caseof**

**End .**

# Chapter 4

# CHAPTER 4 :LOOPS

1. INTRODUCTION
2. WHILE LOOP
3. REPEAT LOOP
4. FOR LOOP
5. NESTED LOOPS

# Introduction

Write an algorithm that prints numbers from 1 to 10

The solution

**Algorithm** Writing;

**Begin**

Write (‘1’) ;

Write (‘2’) ;

Write (‘3’) ;

Write (‘4’) ;

Write (‘5’) ;

Write (‘6’) ;

Write (‘7’) ;

Write (‘8’) ;

Write (‘9’) ;

Write (‘10’) ;

**End**

# Introduction

So far, it's normal,

but if we wanted to print numbers from 1 to 100 on the screen?

we will have to write the print instruction 100 times, which is naturally boring and contradicts the nature of the algorithms that were originally created to facilitate such things.

What is the solution then?

The solution is **iterative loops.**

# Introduction

Loops allow a specific part of the algorithm to be repeated multiple times. There are several types of loops:

- - For loop
- - wihle loop
- - Repeat loop

# FOR LOOP

- It is a **repetitive loop** that repeats the instructions inside it a **specific number of times** that is **known in advance**.
- This loop **uses a variable** that monitors the number of repetitions (**counter**).
- This variable is characterized **by three elements**:

  - Its **initial value**
  - **Final value**
  - the **value** by which **it increases** or **decreases** from **one iteration** to the next.

# FOR LOOP

- The for loop takes the following form:

**For variable** from **start to end step N**

Instructions

**End**

# FOR LOOP

- EXAMPLE :

  Write an algorithm that reads a number N and then calculates N!

# FOR LOOP

- SOLUTION

```
Algorithm  Fact;
Variables N, F,i : integer ;
Begin
Read(N);
F ← 1;
for i =1 to N
F  ← F * i ;
End
Write (F) ;
End .
```

# FOR LOOP

- EXERCICE :

Write an algorithm that reads two numbers A and B and then calculates $A^B$

# WHILE LOOP

It is a recursive loop that repeats the instructions inside it if the entry condition for the loop is met.

If the condition is not met, the instructions inside the loop are not executed.

The loop condition is a logical statement that can be either true or false.

The instructions inside the loop are repeated as long as the condition is true (continuation condition), and the repetition stops when the condition is not met.

The condition is checked before entering the loop.

# WHILE LOOP

The general form of the WHILE loop is given as follows:

**While** Condition **do**

Instructions

**End while**

# WHILE LOOP

- Example :

Write an algorithm that reads a number N and then calculates N!

# WHILE LOOP

**Algorithm** Fact;
**Variables** N, F, i : **integer ;**
**Begin**
Read (N);
F ← 1;
i ← 1;
While (i <= N) do
F ← F * i ;
i ← i + 1 ;
Endwhile
Write (F) ;
**End .**

# Repeat Loop

- The repeat loop is a recursive loop that repeats the instructions inside it until the stopping condition at the end of the loop is met.
- The loop condition is a logical statement that can be either true or false.
- The instructions inside the loop are repeated as long as the stopping condition is met, and the repetition stops when this condition is met.
- The condition is checked after each iteration (at the end of the loop).

# Repeat Loop

- The general form of the repeat loop is given as follows:

Repeat

Instructions

Up to Condition

# Repeat Loop

- Example :

Write an algorithm that reads a number N and then calculates N!

# Repeat Loop

**Algorithm** Fact;
**Variables** N, F, i : **integer ;**
**Begin**
Read (N);
F⟵1;
i ⟵ 1;
Repeat
F⟵F * i ;
i⟵ i + 1 ;
Up to (i > N)
Write (F) ;
**END.**

# Differences between the three loops

- The difference between the FOR loop and the WHILE and REPEAT loops is that the variable responsible for monitoring the number of repetitions (counter) changes automatically in the FOR loop, while this is not the case for the other two loops, as it is necessary to add an instruction that performs this task.
- In the FOR loop, the counter must be used, while this is not the case for the WHILE and REPEAT loops.
- In the case of using the counter in the WHILE and REPEAT loops, it must be given an initial value before the loop.
- As for the WHILE loop condition, it is called a continuation condition because the loop continues as long as this condition is met, unlike the REPEAT loop condition, which is called a stop condition, because when it is met, the loop stops.
- The continuation condition is checked before the start of each iteration, while the stop condition is checked at the end of each iteration.
- The FOR loop is used exclusively in cases where the number of repetitions is known in advance, while we can use the other two loops in both cases (Knowing or not knowing the number of repetitions in advance).

# NESTED loop

# INFINITE LOOP

# CHAPTER 5 AND 6