



PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH ECHAHID HAMMA LAKHDAR
UNIVERSITY OF EL-OUED

FACULTY OF TECHNOLOGY
1ST YEAR LMD SCIENCES AND TECHNIQUES

Chapter III :

Concept of Algorithm and Program

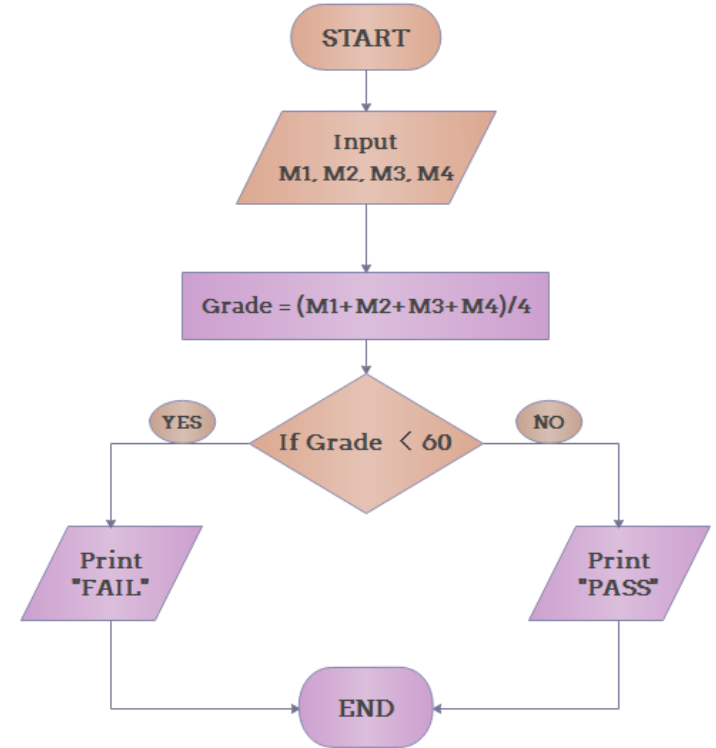
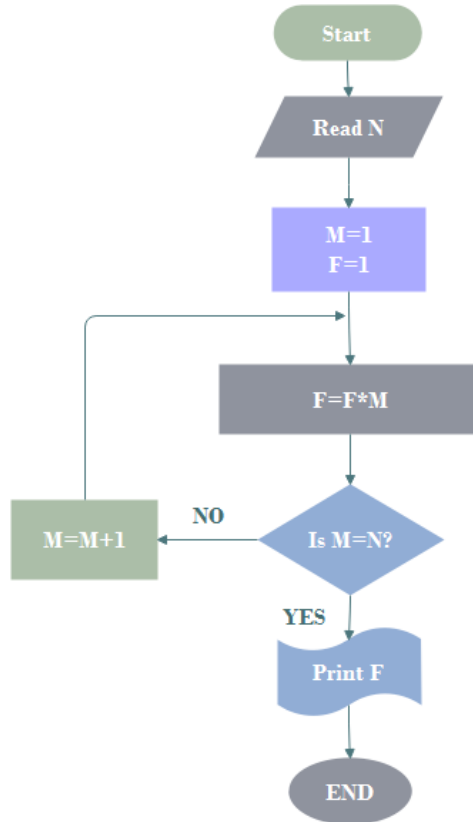
مفهوم الخوارزمية والبرنامج

Dr . BERHOUM Adel

Professor at
University of El

4 - 2023 UNIVERSITY YEAR 202

Outline



1. Introduction

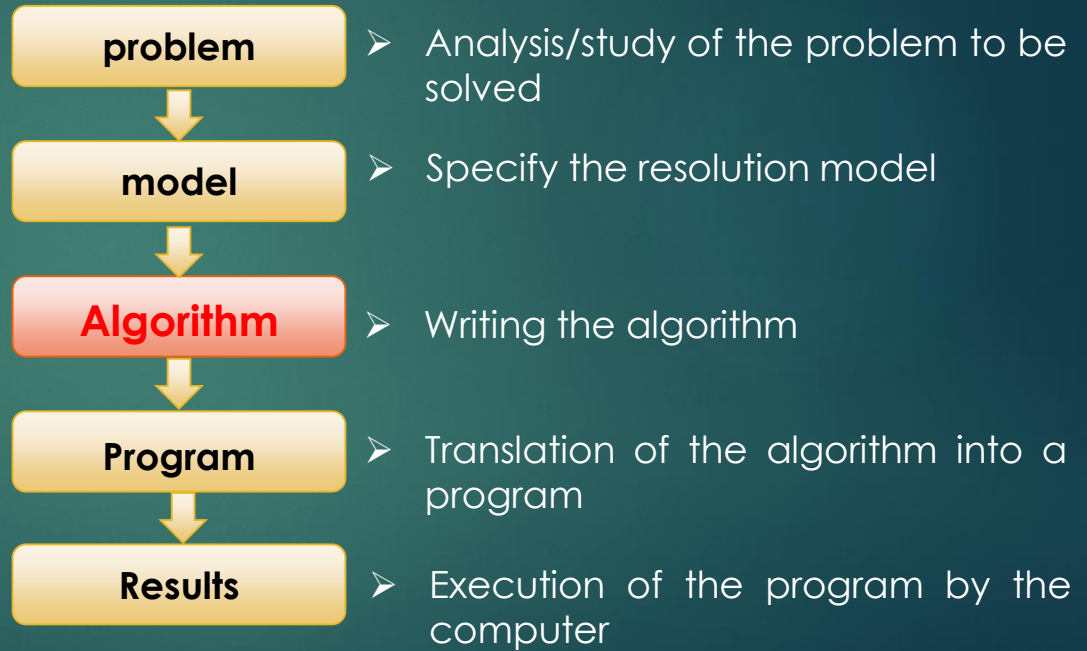
- The word “Algorithm” is invented by the mathematician “**ALKHAWARISMI**”. A Algorithm is the statement of a **sequence of primitive actions** carrying out processing. It describes the plan or sequences of actions to resolve a given problem.
- An algorithm is **a set of sequential and logically ordered actions** (or instructions), making it possible to transform input data (**Inputs**) into output data (**outputs or results**), in order **to solve a problem**.
- So, an algorithm **represents a solution for a given problem**. This solution is specified through a set of instructions (sequential with logical order) that manipulate data. Once the algorithm is written (with any language: French, English, Arabic, etc.), it will be transformed, after choosing a programming language, into a source code program which will be compiled (translated) and executed by the computer.

For the programming language that will be used, it will be the PASCAL language.

1. Concept of an algorithm

The approach and analysis of a problem

An algorithm represents a solution to a given problem. To achieve this algorithmic solution, an analysis and resolution process will be applied. This process consists of the following steps:



2. Data Structures

1. input output Data

An algorithm allows processing on a set of input data to produce output data. The output data represents the solution of the problem treated by the algorithm.

An algorithm can be schematized as follows:

All data in a program are objects in RAM (a reserved space in RAM). Each object (memory space) is designated by a name called: identifier.

Input data (variables)



Algorithm



Output data (result)

2. Data Structures

2. Concept of identifier

An identifier is a character string containing only alphanumeric characters (alphabetical of [az] and [AZ] and numeric [09]) and drawn 8 '_' (underline), and which must begin with either an alphabetic letter Or _.

An identifier allows you to uniquely identify an algorithm (or a program), a variable, a constant, a procedure or a function.

In a given programming language, we are not allowed to use the reserved words (keywords) of the language as identifiers. Among the key words of the **PASCAL** language:

- **program, begin, end,**
- **if, else, then, while, for, do, to, downto, repeat, until, goto,**
- **procedure, function, label, var, const, type, uses, array, of,**
- **real, integer, boolean, char, string, ...**

2. Data Structures

2. Concept of identifier

Examples:

a1: is a valid identifier.

a_1: is a valid identifier.

A_1: is a valid identifier.

x12y: is a valid identifier.

x1 y: is an invalid identifier (because of blank or space).

Examples:

x1-y: is an invalid identifier (because of the sign).

x1_y: is a valid identifier.

1xy: is an invalid identifier (starts a numeric character).

_x1 : ?

2. Data Structures

3. Constants and variables

The data manipulated by an algorithm (or a program) are either constants or variables:

- Constants: a constant is an object containing a value that can never be modified. Its goal is to avoid using a value in a direct way. Let's imagine that an algorithm uses the value 3.14 ten times (the number of occurrences of the value 3.14 is for example 15) and that we want to modify this value by another more precise value: 3.14159. In this case we are required to modify all occurrences of 3.14.
- On the other hand, if we use a constant $\text{PI} = 3.14$ we modify this constant only once.
- Variables: a variable is an object containing a value that can be modified. All data (variable or constant) in an algorithm has a data type (domain of possible values).

2. Data Structures

4. Data types

In algorithms, we have five basic types:

- Integers: represents the set {..., 4, 3, 2, 1, 0, 1, 2, 3, 4, ...}
- Real: represents fractional numeric values and with fixed (or floating) points
- Characters: represents all printable characters.
- Character strings: a sequence of one or more characters
- Booleans (**logical**): represents the two values **TRUE** and **FALSE**.

Algorithm	PASCAL
Entire	Integer
Real	Real
Boolean	Boolean
Character	Char
chain	String

3. Structure of an algorithm / program

An algorithm manipulates data, the data before using it must be identified and declared using the identifier.

An algorithm is made up of three parts:

- **Header** : in this part we declare the name of the algorithm through an identifier.
- **Declarations**: in this part we declare all the data used by the algorithm.
- **Body** : represents the sequence of actions (instructions)

To write an algorithm, you must follow the following structure

3. Structure of an algorithm / program

To write an algorithm/ program, you must follow the following structure

Algorithm <_Algo_Identifier>

<Declarations>

Star

<Body of algo>

<instruction1>

<instruction2>

End.

```
Program <name>;  
Uses Crt; - output to screen  
Const  
<list of constants>  
Var  
<list of variables>  
<list of subprograms>  
BEGIN  
  
END.
```

Program Name



Declaration
section



Executable
section

3. Structure of an algorithm / program

1. Declarations

In the declaration part, we declare all the input and output data in the form of constants and variables.

- **Constants**: are objects containing non-modifiable values. Constants are declared as follows: **<identifier> = <value>;**

Examples:

- PI = 3.14; Real constant.
- MAX = 10; Integer constant.
- cc = 'a'; Constant character.
- ss = 'algo'; Character string constant.
- b1 = true; Boolean constant.
- b2 = false; Boolean constant.

3. Structure of an algorithm / program

1. Declarations

- **Variables**: are objects containing values that can be modified. Variables are declared as follows: **<identifier>: <type>;**

A variable belongs to a data type. We have five basic types of data:

- Integers – Real – Characters - String of characters – Booleans, containing two values: True or False;

Examples:

x: real; variable. n, m:integer; two integer variables. s:String; character string variables.

b1, b2, b3:boolean; 3 Boolean variables. c1: character c1:char; character variable.

N.B.: In addition to constants and variables, we can declare new types, labels and (in a PASCAL program) functions and procedures.

3. Structure of an algorithm / program

2. Body

The body of an algorithm consists of a set of sequentially and logically ordered actions/instructions. The instructions are of five types, namely:

- **Read:** The operation of inputting data to the algorithm. A reading consists of giving an arbitrary value to a variable.
- **Write:** The operation of displaying data. It is used to display results or messages.
- **Assignment:** this is used to modify the values of the variables.
- **Control structure:** It allows modifying the sequentiality of the algorithm, to choose an execution path or repeat a process.
- **Alternative Single /Double Test Structure – Repetitive structure (iterative) – the For loop – the Tantque loop**

The Repeat loop in the PASCAL language, each statement ends with a semicolon. Except at the end of the program, we put a period.

3. Instruction Types

2. Body

All of a program's instructions are written in its body. (between Start and End, i.e. Begin and End.). These instructions can be grouped into three types: inputs/outputs (entry of values and display of results), assignment and control structures (tests and loops)

Algorithm <_Algo_Identifier>

<Declarations>

Star

<Body of algo>

<instruction1>

<instruction2>

End.

```

Program <name>;
Uses Crt; - output to screen
Const
<list of constants>
Var
<list of variables>
<list of subprograms>
BEGIN
END.
    
```

Program Name



Declaration section



Executable section

3. Instruction Types

2. Input/Output Instructions (Read/Write)

Input (Read): An input instruction allows us in a program to give any value to a variable. This is achieved through the read operation. The syntax and semantics of a reading is as follows:

Algorithm	PASCAL	Meaning
read(<id_var>)	read(<id_var>; readln(<id_var>);	Give any value to the variable including the identifier <id_var>.
read(<iv1>, <iv2>, ...);	read(<iv1>, <iv2>, ...);	Give values to variables <iv1>, <iv2>, etc.

It should be noted that the read instruction only concerns variables, we cannot read constants or values. When reading a variable in a PASCAL program, the program hangs waiting for a value to be entered via the keyboard. Once the value is entered, validate with the enter key, and the program resumes execution with the following instruction.

Examples: read(a, b, c); read(height);

3. Instruction Types

2. Input/Output Instructions (Read/Write)

Outputs (Write): An output instruction allows us in a program to display a result (processed data) or a message (character string). This is achieved through the writing operation. The syntax and semantics of a script is as follows: reading is as follows:

Algorithm	PASCAL	Meaning
write (<id_var> <is_const> <value>, <expression>)	write (<id_var> <id_const> <valeur>, <expression>); writeln (<id_var> <id_const> <valeur>, < expression>);	Display a value of a variable, constant, immediate or calculated value through an expression.

It should be noted that the writing instruction does not only concern variables, we can write constants, values or expressions (arithmetic or logical). We can display a value and skip the line immediately after using the instruction: writeln.

3. Instruction Types

2. Input/Output Instructions (Read/Write)

Examples:

<code>write ('Bonjour');</code>	<code>{show Hello message}</code>
<code>write(a, b, c);</code>	<code>{show the values of variables a, b and c}</code>
<code>write(5+2);</code>	<code>{display the result of the sum of 5 and 2: display 7}</code>
<code>write(a+b-c);</code>	<code>{display the result of the arithmetic expression: a+b-c}</code>
<code>write(5<2);</code>	<code>{display the result of the comparison 5 < 2, the result is the boolean value FALSE}</code>
<code>write('The value of x : ', x);</code>	

3. Structure of an algorithm / program

1. Instruction of assignment

–An assignment consists of giving a value (immediate, constant, variable or calculated through an expression) to a variable. The syntax for an assignment is:

Algo : <id_varialbe> = OR ← < valeur> | <id_variable> | <expression >

PASCAL : <id_varialbe> := <valeur> | <id_variable> | <expression> ;

An assignment has two parts: the left part which always represents a variable, and the right part which can be: a value, variable or an expression. The condition for an assignment to be correct is that: the right part must be of the same type (or compatible type) with the left part.

3. Instruction Types

1. Instruction of assignment

Examples:

a = 5 a:=5;	{ put the value 5 in the variable a }
a=5 b:=a+5; B	{ put the value of the expression a+5 into the variable B }
sup a>b sup:=a>b;	{a>b gives a Boolean result, so sup is a Boolean variable}

3. Structure of an algorithm / program

1. Control structures

In general, the instructions in a program are executed in a sequential manner: the first instruction, then the second, after the third and so on. However, in several cases, we are required either to choose between two or more execution paths (a choice between two or more options), or to repeat the execution of a set of instructions, for this we need control structures to control and choose execution paths or redo processing several times. Control structures are of two types: Conditional control structures and repetitive (iterative) control structures.

3. Structure of an algorithm / program

1. Control structures

1. Conditional control structures:

These structures are used to decide the execution of an instruction block: is this block executed or not. Or to choose between executing two different blocks. We have two types of conditional structures:

A. Simple alternative test (if **<cond>** then)

B. Double alternative test (if **<cond>** then **else**)

3. Structure of an algorithm / program

1. Control structures

1. Conditional control structures:

A. Simple alternative test (if <cond> then)

A simple test contains a single block of instructions. Depending on a condition (logical expression), we decide whether the block of instructions is executed or not. If the condition is true, we execute the block, otherwise we do not execute it.

The syntax for a simple alternative test is as follows:

Algorithm: **if** <condition (s)> **then** <instruction (s)>; **Endif**

PASCAL : **if** <condition (s)> **then begin** <instruction (s)>; **end;**

3. Structure of an algorithm / program

1. Control structures

1. Conditional control structures:

A. Simple alternative test (if <cond> then)

```
Read(x);  
    if (x>2) then  
        x:= x + 3;  
    endif  
write('x=', x);
```

```
Read(x);  
    if (x>2) then  
        x:= x + 3;  
    writeln('x=', x);
```

Note: In the PASCAL language, a block is delimited by the two keywords begin and end. If the block contains a single instruction, begin and end are optional (they can be removed).

3. Structure of an algorithm / program

1. Control structures

1. Conditional control structures:

B. Double alternative test (if **<cond>** then **else**)

A double test contains two blocks of instructions: we have to decide between the first block or the second. This decision is made according to a condition (logical or Boolean expression) which can be true or false. If the condition is true we execute the first block, otherwise we execute the second.

The syntax for a simple alternative test is:

Algorithm: **if** <condition (s)> **then** <instruction (s)>; **else** <instruction (s)>; **Endif**

PASCAL : **if** <condition (s)> **then begin** <instruction (s)>; **end; else begin** <instruction (s)>; **end;**

3. Structure of an algorithm / program

1. Control structures

1. Conditional control structures:

B. Double alternative test (if **<cond>** then **else**)

```
Read(x);  
    if (x>2) then  
        x = x + 3;  
    else  
        x = x * 3;  
    endif  
write('x=', x);
```

```
Read(x);  
    if (x>2) then  
        x := x + 3;  
    else  
        x := x * 3;  
writeln('x=', x);
```

3. Structure of an algorithm / program

1. Control structures

1. Conditional control structures:

B. Double alternative test (if **<cond>** then **else**)

– In PASCAL language, you must never put a semicolon before else.

– In the previous example, we can remove begin end from the if and those from the else since there is only one instruction in the two blocks.

Examples:

Write an algorithm (and a PASCAL program) which allows you to indicate whether an integer is even or not.

3. Structure of an algorithm / program

1. Control structures

2. Repetitive control structures:

Repetitive structures allow us to repeat a treatment a finite number of times. For example, we want to display all the prime numbers between 1 and N (N given positive integer). We have three types of iterative structures (loops):

A. For Loop

B. While loop

C. Repeat Loop

3. Structure of an algorithm / program

1. Control structures

2. Repetitive control structures:

A. For Loop

The repetitive control structure for uses an integer index which varies (with increment = 1) from an initial value to a final value. At the end of each iteration, the index is incremented by 1 automatically (implicitly).

The syntax of the for loop is as follows:

```
For <index> = <vi> To <vf> Do  
    <instruction (s)>  
endfor;
```

```
For <index> := <vi> To <vf> Do  
    begin  
        <instruction (s)>  
    end;
```

<index>: integer variable <vi>: initial value <vf>: final value

3. Structure of an algorithm / program

1. Control structures

2. Repetitive control structures:

A. For Loop

- The **for loop** contains a block of instructions (the instructions to repeat). If the block contains a single instruction, the begin and end are optional.
- The block will be repeated a number of times = $(\langle vf \rangle = \langle vi \rangle + 1)$ if the final value is greater than or equal to the initial value.
- The block will be executed for $\langle index \rangle = \langle vi \rangle$, for $\langle index \rangle = \langle vi \rangle + 1$, for $\langle index \rangle = \langle vi \rangle + 2$, ..., for $\langle index \rangle = \langle vf \rangle$. You should never put a semicolon after the do keyword. (logical error)

3. Structure of an algorithm / program

1. Control structures

2. Repetitive control structures:

B. While loop

The repetitive control structure while uses a logical or Boolean expression as a condition for access to the loop: if the condition is verified (it gives a true result: TRUE) then we enter the loop, otherwise we left him.**The syntax of the repeat loop is as follows:**

```
While <condition (s)> do
```

```
<instruction (s)>
```

```
While <condition (s)> do
```

```
<instruction (s)>
```

3. Structure of an algorithm / program

1. Control structures

2. Repetitive control structures:

B. While loop

- The block of instructions is executed as long as the condition is **true**. Once the condition is **false**, we stop the loop, and we continue executing the instruction that comes after end As long as (after end).
- Like the for loop, you must never put a semicolon after do.
- Any for loop can be replaced by a while loop, however the reverse is not always possible.

3. Structure of an algorithm / program

1. Control structures

2. Repetitive control structures:

C. Repeat loop

The repetitive control structure repeat uses a logical or Boolean expression as an exit condition from the loop: if the condition is verified (it gives a true result: TRUE) we exit the loop, otherwise we access it (we repeat the execution of the block).

The syntax of the repeat loop is as follows:

```
repet
  <instruction (s)>
until <condition (s)>
```

```
repet
  <instruction (s)>
until <condition (s)>
```

<condition>: logical expression that can be **true** or **false**.

3. Structure of an algorithm / program

1. Control structures

2. Repetitive control structures: C. Repeat loop

We execute the block of instructions until we have the correct condition. Once the condition is verified, we stop the loop, and we continue executing the instruction that comes after until (after until). In the repeat loop we do not use begin and end to delimit the block of instructions (the block is already delimited by repeat and until).

The difference between the repeat loop and the while loop is:

- The condition to repeat and always the opposite of the condition while: to repeat is the exit condition of the loop, and for while it is the condition to enter.
- The condition test is at the end of the loop (the end of the iteration) to repeat. On the other hand, it is at the start of the iteration for the while loop. That is to say, in as long as we test the condition before entering the iteration, and in repeat we do the iteration after we test the condition.

3. Structure of an algorithm / program

1. Control structures

3. Branch/Jump control structure (the Goto instruction):

A branch statement allows us to jump to a location in the program and continue execution from that location. To make a connection, you must first indicate the target of the connection via a <num_etiq> label:. Then we jump to this place with the instruction go to <num_etiq> (in Pascal: **goto <num_etiq>**).

The syntax for a branch is as follows:

```
Goto <num_etiq>
```

```
.....
```

```
.....
```

```
<num_etiq>
```

```
Goto <num_etiq>
```

```
.....
```

```
.....
```

```
<num_etiq>
```

3. Structure of an algorithm / program

1. Control structures

3. Branch/Jump control structure (the Goto instruction):

N.B.:-

A label represents a number (integer), example: 1, 2, 3, etc.

- In a PASCAL program, you must declare the labels in the declaration part with the keyword label. (we saw const for constants var for variables)
- A label designates a single place in the program, you can never indicate two places with the same label.
- On the other hand, you can make several connections to the same label.- A jump or branch can be to an earlier or later instruction (before or after the jump).

3. Structure of an algorithm / program

1. Control structures

3. Branch/Jump control structure (the Goto instruction):

Example:-

```
Algorithme Branching;  
variable a, b, c:integer;  
label 1 , 2;
```

Star

```
    read(a,b);
```

```
    2: c:=a;
```

```
    if (a>b) then
```

```
        goto 1;
```

```
    Endif;
```

```
        a := a + 5;
```

```
    goto 2;
```

```
    1: write(c);
```

```
end.
```

```
program Branching;  
uses wincrt;  
var a, b, c:integer;  
label 1 , 2;
```

```
begin
```

```
    read(a,b);
```

```
    2: c:=a;
```

```
        if (a>b) then
```

```
            goto 1;
```

```
            a := a + 5;
```

```
        goto 2;
```

```
        1: write(c);
```

```
End.
```

3. Structure of an algorithm / program

1. Control structures

3. Branch/Jump control structure (the Goto instruction):

In the example above, there are two labels: 1 and 2.

Label 1 refers to the last instruction of the algorithm/program (write(c) / write(c) ;),

label 2 refers to references the third instruction of the algorithm/program (c a; / c := a;).

To run the algorithm, we use the following table (a = 2 and b = 5):

Instructions	Variables	a	b	c
Read (a, b): Give any two values to a and b		2	5	?
c = a ;		2	5	2
a > b → false if a = 2 and b =5 we do not enter the block if a = a + 5;		7	5	2
goto 2 c = a;		7	5	7
a > b → true since a = 7 and b =5 we enter the if block go to 1 => write (c)		7	5	7 (result displayed)

3. Structure of an algorithm / program

1. Control structures

3. Branch/Jump control structure (the Goto instruction):

- A. There are two types of connection: has. unconditional connection: it is an unconditional connection, it does not belong to an if block or an otherwise block. In the previous example, the go to 2 instruction is an unconditional jump.
- B. Conditional branch: On the other hand, a conditional branch is a jump which belongs to an if block or an else block. The go to 1 (goto 1) instruction in the previous example is a conditional jump since it belongs to the if block.

Instructions	Variables	a	b	c
Read (a, b): Give any two values to a and b		2	5	?
c = a;		2	5	2
a > b → false if a = 2 and b =5 we do not enter the block if a = a + 5;		7	5	2
go to 2 c = a;		7	5	7
a > b → true since a = 7 and b =5 we enter the if block go to 1 => write (c)		7	5	7 (result displayed)

4. Organization chart representation







A flowchart is the graphical representation of solving a problem. It is similar to an algorithm. Each type of action in the algorithm has a representation in the flowchart.

It is better to use the algorithmic representation than the flowchart representation, especially when the problem is complex. The disadvantages that may be encountered when using flowcharts are:

- When the organization chart is long and takes up more than one page,
- arrow overlap problem,
- more difficult to read and understand than an algorithm.

4. Organization chart representation

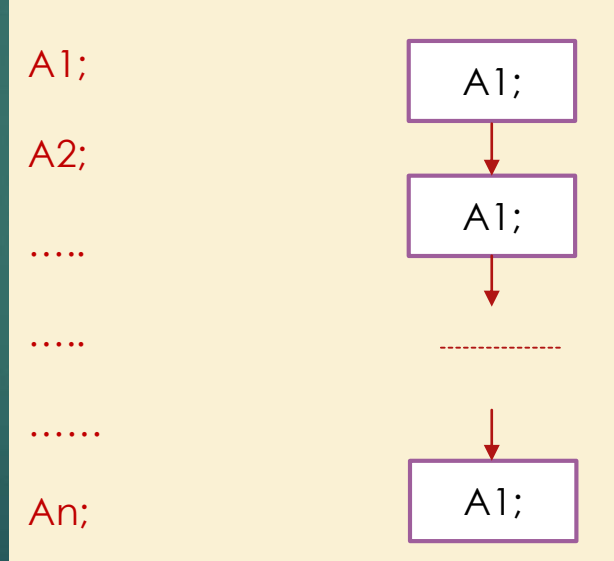
1. Organization chart symbols

	Represents the start and end of Algorithm
	Inputs/Outputs: Reading data and writing results.
	Calculations, Treatments
	Tests and decision: we write the test inside the diamond
	Order of execution of operations (Sequencing)
	Connector

4. Organization chart representation

1. Representation of algorithmic primitives

Sequencing allows you to execute a series of actions in the order in which they appear. Let A_1, A_2, \dots , be a series of actions, their sequence is represented as follows:



A_1, A_2, \dots, A_n : can be elementary or complex actions.

4. Organization chart representation

1. Representation of algorithmic primitives

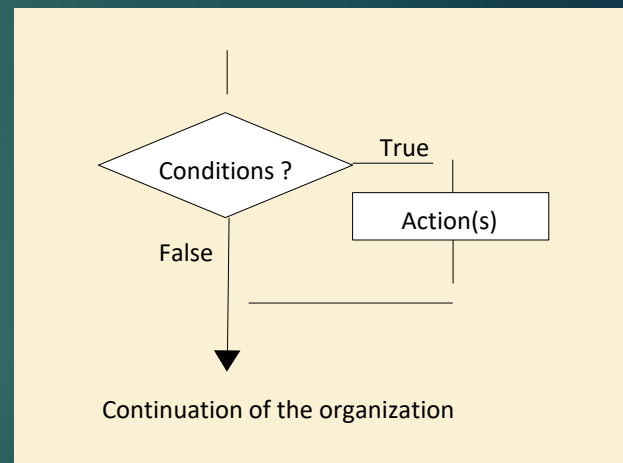
A. The simple alternative structure

If < condition (s) > **then**

<Action (s)>

Endif

If the condition is verified, the <action(s)> block will be executed, otherwise nothing, and we continue executing the instruction after end if.



4. Organization chart representation

1. Representation of algorithmic primitives

A. The simple alternative structure

The conditions used for testing (single or double) are logical or Boolean expressions, meaning expressions whose evaluation gives either **TRUE** or **FALSE** . Any comparison between two numbers represents a logical expression. We can form logical expressions from other logical expressions using the following operators: **Not**, **Or** and **And**.

Examples: $(x \geq 5)$: is a logical expression, it is true if the value of x is greater than or equal to 5. it is false other wise. **Not** $(x \geq 5)$: E.L. which is true only if the value of x is less than 5. $(x \geq 5)$ **And** $(y \leq 0)$: E.L. which is true if x is greater than or equal to 5 and y less than or equal to 0.

4. Organization chart representation

1. Representation of algorithmic primitives

B. The double alternative structure

Algorithmic representation	Représentation sous forme d'organigramme
<pre> if <condition (s)> then <action1 (s)> else <action2 (s)> endif </pre> <p>If the condition is verified, the <action1(s)> block will be executed, otherwise (if it is false) we execute <action2(s)>.</p>	<pre> graph TD Start(()) --> Cond{Conditions?} Cond -- FALSE --> Act2[Action2(s)] Cond -- TRUE --> Act1[Action1(s)] Act2 --> Join(()) Act1 --> Join Join --> End[Continuation of the organization] </pre>

4. Organization chart representation

1. Representation of algorithmic primitives

C. The iterative FOR structure (FOR Loop)

Algorithmic representation	Représentation sous forme d'organigramme
<pre> For<cpt> =<iv> To <fv> then <action1 (s)> endFor </pre> <p>In the FOR loop, we execute the <actions> block (<vf> <vi> + 1) times. This in the case where <vf> is greater than or equal to <vi>. Otherwise, the action block will never be executed</p>	<pre> graph TD Start(()) --> Decision{<math>cpt \leq iv</math>} Decision -- TRUE --> Action[Action(s)] Action --> Increment[<math>cpt = cpt + 1</math>] Increment --> Decision Decision -- FALSE --> End((Continuation of the organization)) </pre>

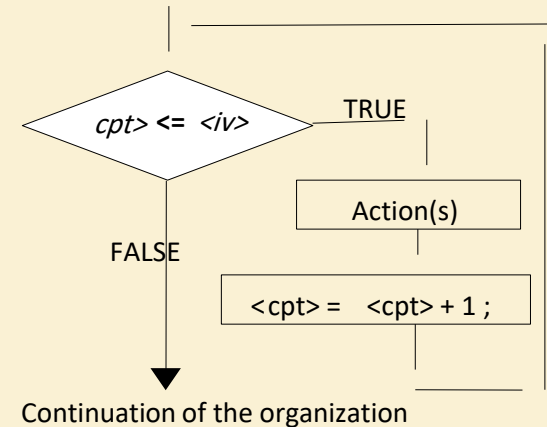
4. Organization chart representation

1. Representation of algorithmic primitives

C. The iterative FOR structure (FOR Loop)

The flow of the FOR loop is expressed as follows:

- 1** – the integer variable $\langle \text{cpt} \rangle$ (the counter) takes the initial value $\langle \text{vi} \rangle$;
- 2** – we compare the value of $\langle \text{cpt} \rangle$ to that of $\langle \text{vf} \rangle$; if $\langle \text{cpt} \rangle$ is greater than $\langle \text{vf} \rangle$ we exit the loop;
- 3** – if $\langle \text{cpt} \rangle$ is less than or equal to $\langle \text{vf} \rangle$ we execute the $\langle \text{action(s)} \rangle$ block;
- 4** – the FOR loop automatically increments the counter $\langle \text{cpt} \rangle$, that is to say it adds one ($\langle \text{cpt} \rangle \leftarrow \langle \text{cpt} \rangle + 1$);



4. Organization chart representation

1. Representation of algorithmic primitives

D. The While Iterative Structure (While Loop)

Algorithmic representation	Représentation sous forme d'organigramme
<p>While <cpt> =<iv> To <fv> do</p> <p style="padding-left: 20px;"> <action1 (s)></p> <p>endwhile</p>	<pre> graph TD Start(()) --> Cond{Condition?} Cond -- True --> Act[<Action(s)>] Act --> Cond Cond -- False --> Cont[Continuation of the organization] </pre>

We execute the <actions> instruction block as long as the <condition> is verified (i.e. it is true). The sequence of the loop is as follows:

- 1 – We evaluate the condition: if the condition is false we exit the loop;
- 2 – If the condition is true, we execute the <actions> block; otherwise go to 4.
- 3 – We return to 1; 4 – We continue with the rest of the algorithm

4. Organization chart representation

1. Representation of algorithmic primitives

E. The iterative structure Repeat (Repeat Loop)

Algorithmic representation	Représentation sous forme d'organigramme
<p>Repet</p> <p> <action1 (s)>;</p> <p>until <condition (s)></p>	<pre> graph TD Start(()) --> Action[<Action(s)>] Action --> Condition{Condition?} Condition -- True --> End[Continuation of the organization] Condition -- Fals e --> Action </pre>

We repeat the execution of the <action(s)> **block until** we have the correct condition. The procedure is as follows:

- 1 – We execute the <action(s)> block;
- 2 – We evaluate the condition: if the condition is verified (it is true) we exit the loop (we continue the rest of the algorithm);
- 3 if the condition is not verified (it is false) we return to 1.

*Thank
you*



For listening and following the course

Dr. BERHOUM Adel