Chapter 2: Simple sequential algorithm

1. Concept of algorithmic language

Algorithmic language:
The algorithmic language is a compromise between natural language and a programming language, characterized by a list of keywords.

The key word:
A keyword is an identifier that has a specific meaning to the programming language (Begin, End, If, For, Repeat, While, etc.).

2. Structure of an algorithm

An algorithm can be presented as follows:

```
Algorithm name_of_algorithm;            // Header

Const   List_of_constants;
Var List_of_variables;                  // Declaration
Type List_of_types;

List of procedures and procedures/functions;

Begin
   Instruction 1 ;
   Instruction 2 ;                      // Body of algorithm
   …                                    // Initialization → processing → results display

   Instruction N ;
end.
```

✓**The header:** used to identify the algorithm by giving it a name.
✓ **Declarations**: This is a list of all objects (constants, variables, etc.) used and manipulated in the body of the algorithm.
✓ **The body**: Contains the operations and instructions to be executed.

Example :

The algorithm that calculates and displays the area and perimeter of a circle of radius R can be written as follows:

Analysis

Inputs: R
Ouputs : S and P
R is real so S and P are reals too.

Operations

1. Read the value of the radius R
2. Calculate the surface S according to the law S = $\pi * R^2$
3. Calculate the perimeter P according to the law P = $2* \pi * R$
4. Present (display) the values of: S and P

Formalism

```
Algorithm cercle ;
 Const
   Pi = 3,14 ;
 Var
   R, S, P : Real ;
Begin
        Read ( R ) ;
        S ← Pi x R x R;
        P ← 2 x Pi x R;
        Write ( S ) ;
        Write ( P ) ;
end.
```

3. Variables and constants

3.1. Definition of a variable

A variable is a memory space identified by a name (address), used to store a value, which can be modified during the execution of the algorithm.

3.2. Definition of a constant

A constant only takes on a single value during the execution of the algorithm.

3.3. Variable declaration

To declare a variable we define its name and its type.

Syntax

```
Var
   Name_Variable1: Type1 ;
   Name_Variable2,…, Name_VariableN : Type2 ;
```

Example

Algorithm	Program (C)
Var A, B : integer ;	int A, B ;
C : real ;	float C ;
X1 ,X2 : boolean ;	bool X1,X2 ;
id_1 : character ;	char id_1 ;

3.4. Constantes declaration

To declare a constant we define its name and its value.

Syntax

```
Const
   Name_constante1= value1;
   Name_constante2= value2;
```

Example

Algorithm	Program (C)
Const Pi = 3.14 ;	Const Float Pi=3.14 ;
resp = true ;	bool resp = true ;

A variable is identified by an **identifier** and a **type** and a **value**.

A. The identifier

An identifier is the name of a variable, a constant or the name of an algorithm (program). An identifier must respect a particular syntax:

- It is made up by a serie of letters, alphabets (in upper and lower case) and numbers or the underlined character "_" (the underscore).
- It must begin with a letter or "_". (The name must not start with a number).
- An identifier cannot be a keyword (Start, End, If, Then, Else, printf, if, else, while, for…..etc).
- An identifier must not contain special characters like: ?, !, *, …

Remarks :

- To facilitate the readability of an algorithm, it is preferable to use meaningful names.
- Characters of the Greek alphabet are prohibited. If we need to use α, β, or δ, we must write them in Latin alphabet like alpha, beta, delta and so on.
- It is forbidden to use accented characters (é, è, î, ç, ', ", …) in an identifier.
- It is forbidden to use indices or exponents (do not use x_1, x^2 but rather use x1 and x2)
- The C language is case sensitive (The variable Age ≠ age)
- *Examples*: x, y, PI, radius_of_the_circle, _x1, _y2, …

B. The type

A variable is assigned a type. This type indicates the memory space that can be reserved for this variable.

The standard (basic) types in algorithmic language are: Integer, Real, Boolean, Character.

- **The integer type:** designates a set of integer values. Examples: -10, -6, 0, 5, 1001...
- **The real type:** designates the set of real values. Examples: 1.55, -224.25 ...
- **The boolean type:** designates two values: TRUE or FALSE.
- **The character type:** This type includes all the characters used for writing text ('a', 'b 'c', 'A', 'B', '*', '+', ';'. ..).

Remarks :

- There are also composed types defined from basic types such as arrays, records, strings, etc. Also, we can define a new type and give it a new name.
- The representation of variables in memory depends on the systems and languages used.

C. The value

The value can change during the algorithm (program), but must respect the type. A variable whose type is integer can therefore never contain a value with comma.

4. Basic operations

4.1. Expressions

An algorithm can contain expressions.
An expression is a combination of operands (variables or values) and operators

Exp:

$$X^{\text{operand}} +_{\text{operator}} Y^{\text{operand}}$$

An OPERATOR: is a tool that allows performing calculations.
An OPERAND: can be a value, a constant, a variable or a function call which will be used by an operator.

4.2. Types of operators

A. Arithmetic operators

Addition +, Subtraction -, Division /, Multiplication *, Equal =, MOD and DIV

MOD : give the remainder of division (x MOD y).
DIV : give the quotient of division (x DIV y).

Algorithm	**Program C**
DIV	/
MOD	%

B. Comparaison operators

Strictement Inférieur <, Inférieur ou égale ≤, Strictement Supérieur >, Supérieur ou égale ≥, Différent ≠, Strictly inferior <, inferior or equal than ≤, Strictly superior >, superior or equal than ≥, Different ≠

Algorithm	**Program C**
≤	>=
≥	<=
=	==
≠	!=

Note: The result of a comparison will be TRUE (1)/FALSE (0)

C. Logical operations

AND : logical AND, **OR :** logical OR, **NOT** : negation

Algorithm	Program C
AND	&&
OR	| |
NOT	!

Example: Logical table

A	B	A AND B	A OR B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

A	NOT A
False	True
True	False

4.3. The rules for evaluating an expression

Calculating the value of an expression containing more than one operator depends on the meaning given to this expression. Example: X + Y * Z is an ambiguous expression.

Parentheses remove ambiguity: (X + Y) * Z or X + (Y * Z).

To avoid any ambiguity in the absence of parentheses, evaluation rules have been established. An order of priority between the operators is introduced as follows:

Level 1 : ( ) // ---- highest priority
Level 2 : NOT
Level 3 : * , / , MOD , DIV , AND
Level 4 : + , - , OR
Level 5 : = , < , >, ≥, ≤ // ---- least priority

Remarks:

As best practice in writing algorithms

- We must always add parentheses in the expressions to avoid ambiguities.
- The algorithmic expressions must be written in linear form.

Examples :

$\frac{X+Y}{2Z}$ → (X+Y)/ (2*Z) ;

B^2 - 4 AC → (B * B) – (4 * A * C)

NOT A AND B → NOT (A AND B) // (NOT A) AND B

A ∈ [0,255] → (A >=0) AND (A<=255)

4.4. Basic (elementary) instructions

In general, there are three elementary instructions: Read, Write and assignment ←

A. Assignments

The assignment is an instruction that stores the value of an expression in a variable.

Syntax

```
    VariableName ← expression
```

Examples

```
X ← 7                 // Assign a value to the variable X (X receives 7)
Y ← X                 // Copy the value of variable X into variable Y (Y receives X)
Z ← (2 * X + T) / Y   // An expression formed from several operations ( Z receives (2 * X + T) / Y )
```

Remarks

- If the variable already contained a value, it would be replaced by the value of the expression. The old value of the variable will be lost and there is no way to recover it!

- A type check operation is performed before assignment. It is an error if the value of the expression does not belong to the type of the variable, unless it is an integer value and the variable is of real type. In this case the integer value is converted to real.

Exercises

Exercise 1

What will be the values of variables a and b after performing the following actions?

```
Algorithm Ex1 ;
 var
    a,b : integer ;
 Begin
    a ← 1 ;
    b ← a + 3 ;
    a ← 3 ;
 end.
```

Exercise 2

What will be the values of variables a, b and c after executing the following instructions?

```
Algorithm Ex2;
  var
    a, b, c : integer ;
 Begin
   a ← 5 ;
   a ← a – 1 ;
   b ← 3 ;
   c ← a +  b ;
   a ← 2 ;
   c ← b – a ;
   c ← c +1 ;
end.
```

B. Input-output operations

The reading instruction (READ)

The READ instruction allows assigning a value to a variable from the keyboard.

Syntax: read(VariableName);

Examples:

```
        read(A);
        read(B);
```

- Several successive reading instructions can be grouped into a single instruction. For the previous example, we can replace the three reading instructions with:
  read(A, B).
- On a computer, when the processor receives the read(VariableName) order, it stops execution of the program and waits for a value. The user must then enter a value from the keyboard. As soon as the entry is validated (by pressing the Enter key ↲), execution of the program continues. The value passed by the user is assigned to the variable and overwrites its previous value.
- The read(VariableName) instruction causes an error if the value entered does not belong to the type of the variable, unless it is an integer value and the variable is of real type. In this case the integer value is converted to a real value.

The writing instruction (write)

The write instruction allows display on screen. There are two types of display:

1. Display the value of a variable or an expression.

   Syntax: Write (VariableName);

   Example : Write(X) ; or Write (2 * X + Y) ;

// If X = 10 and Y= 5 then, the first instruction display 10 and the second display 25 .

2. Display a text message.

   Syntax : Write ("a message") ;
   Example : Write(" The result is = ") ;

Different successive write instructions can be grouped into a single instruction. For example, the sequence of instructions:

```
Write ( "The result is =" );
Write (X) ;
Can be replaced by: Write (“The result is =”, X);
```

This instruction displays : "The result is = 10".

Note: Message communication is very useful in programming. It offers the possibility:

- To guide the user by telling him what the machine expects from him following a reading order.
- To explain the results of a treatment.

C. Comments

A comment is optional text that has no effect on the instructions of the algorithm. It is used to explain why certain instructions are used in a program.

Syntax of a comment:

/* comment */ or // comment

Example :

```
Algorithm Sum;
 var
   A, B, C: integer;
 begin
    /* this program calculates the sum of two numbers*/
   Read (A);
   Read (B);
   C ← A+B; //The variable C will contain the result
   Write (''the sum ='', C);
 end.
```

D. Inputs and Outputs in C language

Syntax :

Algorithm	**Program C**
←	=
read (A)	scanf('' %d'', &A) ;
write (A)	printf(''%d'', A) ;

E. Variable content format

The following table represents the main formats that can be used with input-output functions in C language:

Format	**Description**
%d	Data with Integer type
%f	Data with real type
%c	Data with character type
%s	Data with string type

F. Exercises

Exercise 1

Write an algorithm that calculates the perimeter and area of a rectangle. Translate into C language.

Exercise 2

Write an algorithm that calculates the sum and average of 5 exam notes.

Exercise 3

Write an algorithm that calculates the sum and average of 5 exam scores, using only two variables.

Exercise 4

Write an algorithm that reads two integer values A, B, then swap them and display the result.

exp: if A=12 and B = 23 after swapping A= 23 and B = 12

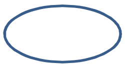
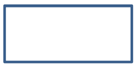
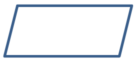
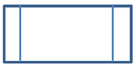
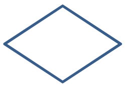
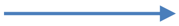
Translate into C language.

5. Flowchart representation

A flowchart is a diagram that illustrates the steps and decisions of a process. It can be used to graphically represent an algorithm.

The flowchart contains standard forms. Each form describes a specific operation.

The basic forms of a flowchart are represented in table below:

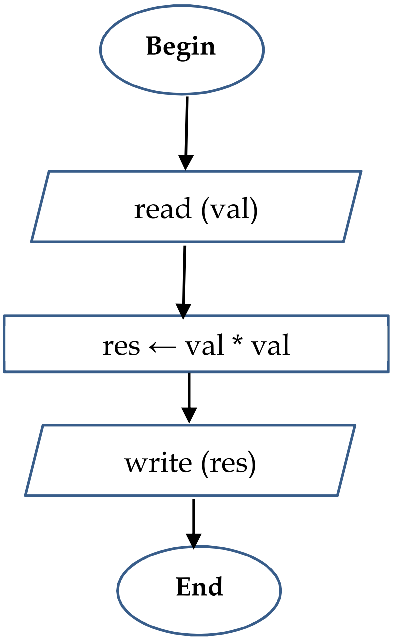
Form	Description
	Start(Begin)/end of the flowchart
	Processing (operation)
	Write, Read, Input, Output operations
	A subroutine (subprogram) : Call to a portion of an algorithm (program) considered as a simple operation
	Decision (based on condition)
	Flow lines : show the direction of the flow (process).

Example :

Consider the following algorithm which calculates the square power of a number Val.

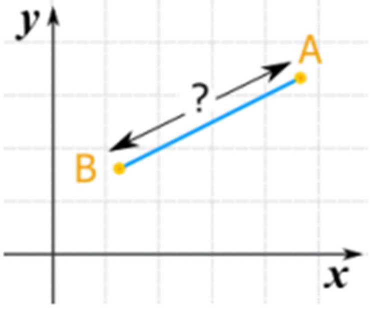
```
Algorithm square_pow;
Var
    val, res : integer;
Begin
    read(val);
    res ← val * val ;
    Write (“the result is: “, res);
End.
```

The corresponding flowchart is represented as follows:



Exercise:

1- Write an algorithm that calculates the distance between two points A and B in a plane (X, Y).



$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

2- Represent the algorithm using a flowchart.

3- Translate the algorithm to a program using C language.