Chapter 2 :
Simple sequential Algorithms

1. Structure of an algorithm

Reminder : *An algorithm is a sequence of ordered and finite instructions (operations, actions or treatments) that allows solving a given problem.*

An algorithm consists of three parts:

- **The header part :** usually this part only serves to define what makes the algorithm.
- **The header part :** Typically, this part is only used to define what the algorithm does.
- **The Declarations section:** contains the declaration of the different objects (variables and constants) used in the problem-solving process.
- **The Instructions (actions, operations or treatments) section** or the body of the algorithm : it contains the different steps for solving the problem written in the form of instructions.

```
Algorithm algo_name ;
    Declaration of constants ;
    Declaration of variables ;
Begin
    Instruction_1 ;
    Instruction_2 ;
    …
    Instruction_n ;
End.
```

1.1. The header part: It starts with the reserved word "Algorithm" followed by the name of the algorithm.

1.2. The declarations part: Any object used or manipulated in the problem-solving process must be declared in this part. Two types of objects are distinguished: those that keep their values unchanged are called **<u>Constants</u>**, and those that can change their values during the execution of the algorithm are called **<u>Variables</u>**.

1.2.1. Constants: The syntax for declaring a constant is as follows:

Const const_name = const_value;
Where:
const_name is the name of the constant.
const_value is the value of the constant.

Example:
Const Pi = 3.14;

1.2.2. Variables: The syntax for declaring a variable is as follows:
Var var_name: var_type;
Where:
var_name is the name of the variable.
var_type is the type of the variable.

Variable Types:

The type of a variable indicates the set of values that can be taken by that variable. The types offered by algorithmic language and most programming languages (for our purposes, we will study the C language) are simple or composite:

Simple Types : A type is considered simple if it describes a single value. In turn, simple types can be classified into two categories:

1. **Numeric Types:**

Type	Meaning	Example	Operators for manipulating the type
Integer	Set of relative integer values	18, -75, +23	Basic arithmetic : +,-,*,/
			remainder of an integer division (modulo) : %
			Relational : <, >, <=, >=, =, ≠
Real	Set of real values	8.03,-1.6, +3.7	Basic arithmetic : +,-,*,/
			Relational : <, >, <=, >=, =, ≠

1. **Types symboliques :**

Type	Meaning	Example	Operators for manipulating the type
Boolean	Set of logical values	2 values : TRUE et FALSE	Basic logical : And, Or, Not
Character	Set of single-character values	's','!', '5', '%'	Relational : <, >, <=, >=, =, ≠
String	Set of multi-character values	"ab","+1"	Relational : <, >, <=, >=, =, ≠

Compound (Structured) Types : A type is considered compound if it describes composite information (several values). Among these types, we can mention : the array type, the string type, and the structure type, which will be studied in the following chapters.

Examples :

```
var x, y: integer;        // Declaration of variables x and y of type integer.
    z: real;              // Declaration of variable z of type real.
    c: char;              // Declaration of variable c of type character.
    ch: string;           // Declaration of variable ch of type string.
```

Remarks :
- Declaring a variable is equivalent to reserving a memory location for it. Its initial value is unknown !
- In algorithmics and programming languages, the name of the algorithm (algo_name), the name of the constant (const_name), and the name of the variable (var_name) must be an **identifier**.

Definition :

Identifier : An identifier is a sequence of alphabetical characters (a, b, ..., z, A, B, ..., Z) and/or numerical characters (0, 1, ..., 9) and/or underscore (_) characters. This sequence must not start with a digit and must not be a reserved word in the algorithmic language (for example, these words are reserved in the algorithmic language : **Algorithm, const, var, integer, real, begin, end, read, if, then, else, ...**)

1.3. **The instructions part :** It contains the instructions used in problem-solving. This part starts with the reserved word "**Begin**" and ends with the reserved word "**End**." Among the basic instructions in this part, we can mention the following:

- *Reading instruction.*
- *Writing instruction.*
- *Assignment instruction.*
- *Conditional instruction (to be discussed in Chapter 3).*
- *Repetitive instruction (loops) (to be discussed in Chapter 4).*

1.3.1. Reading instruction:

The reading instruction is used to retrieve (request) a value from the user and assign it to the memory space specified by the variable in parentheses. The value is typically entered through an input device (usually the keyboard).

Its syntax is:

```
Read (var_name); // var_name: is the name of a variable.
```

Note : several statements **read**() can be replaced by a single statement **read**() by listing all the variables as parameters, separating them with commas. For example, "**Read** (x); **Read** (y); **Read** (z);" can be replaced by "**Read** (x, y, z);"

Examples :

In Algorithmics	In the C Language
Read (x) ; // x : integer ;	**scanf**("%d",&x) ;
Read (y,z) ; // y : real ; z : char ;	**scanf**("%f%c",&y,&z) ;

1.3.2. **Writing instruction:** This is an instruction that allows displaying (communicating) to the user, using an output device (usually the screen), a text (message) or a value.

Syntax :

```
Write ("a text"); // To display text that should be enclosed in double quotes.
Write (expression); // To display the value of an expression. This expression can be: a literal constant (2, 'a', "abcd", ...), symbolic constant (Pi), a variable, an arithmetic or logical expression.
```

Exemples :

In Algorithmics	In the C langage
Write("Hello") ;	**printf**("Hello") ;
Write(x) ; // x : Integer ;	**printf**("%d",x) ;
Write("The value of x is : ",x) ; // x : integer ;	**printf**("The value of x is : %d",x) ;

Note: several **write**() statements can be replaced by a single write() statement by listing all the expressions and texts of the different write() statements, in the same order, separated by commas.

Example:

write ("the sum of "); **write** (x); **write** (" and "); **write** (y); **write** (" is = "); **write** (s); are replaced by **write** ("the sum of ", x, " and " ,y , is = ", s);

1.3.3. Assignment instruction : This is one of the most important and widely used instructions in algorithmics. The assignment is denoted by (←) and consists of assigning the value of an expression (after its evaluation) to a variable.

Syntax :

```
var_name ← expression; // After the evaluation of the expression, its value is assigned to the variable var_name.
```

such as:

var_name : is the name of a variable.

expression : can be a literal constant (2, 'a', "abcd", ...), a symbolic constant (Pi), a variable, an arithmetic or logical expression.

Remarks :

- **var_name** and **expression** must be of the same type or compatible types.
- The instructions x←x+1 and x←x-1 have meaning in programming and are respectively called the increment and decrement instructions.
- Some languages provide default values for declared variables. To avoid any problem it is better to initialize declared variables.
- Each variable is characterized by:
  - A name (identifier).
  - A type: indicates the values that can be taken by the variable.
  - A value: indicates the magnitude taken by the variable at a given time.

Examples :

Algorithm basic_instructions_example ;	
Const Pi=3.14 ;	Declaration of a constant Pi with a value of 3.14.
var x,y : **integer** ;	Declaration of two integer variables x and y.
z,k,p : **real** ;	Declaration of three real variables z, k, and p.
c1,c2 : **char** ;	Declaration of two character variables c1 and c2.
ch1, ch2 : **string** ;	Declaration of two string type variables ch1 and ch2.
Begin	
Read(x,z) ;	Reading the value of x and the value of z.
y ← 10 ;	Assigning the value 10 to the variable y.
k ← x ;	Assigning the value of x to the variable k.
p ← k+2*y+Pi ;	Assigning the value of the expression k+2*y+Pi to the variable p.
c1← '!' ;	Assigning the character '!' to the variable c1.
c2 ← c1 ;	Assigning the value of the variable c1 to the variable c2.
ch1 ← "abcd" ;	Assigning the string "abcd" to the variable ch1.
ch2 ← ch1 ;	Assigning the value of the variable ch1 to the variable ch2.
Write ("the value of z = ", z);	Displaying the text "the value of z = " followed by the value of z.
End.	

Expressions and Operators:

Definition of an Expression :

An expression is a combination of literal constants, variables, constants, operators, and functions that is evaluated (or computed) following the rules of operator precedence and associativity.

Operators :

In algorithmics, operators can be classified into :

- ***Arithmetic operators :*** these operators are applied to numeric type operands (integer and real). The basic operators are : +,- (unary), - (binary), *, /, % (modulo or rest of division), ^ (exponentiation).
- ***Relational operators :*** these operators apply to numeric type operands (integers and real) and to the character type. Relational operators are : <, =, ≥, ≤, ≠, >.
- ***Logical operators :*** these operators apply to operands of logical types. The basic logical operators are: AND, OR, NOT.

Operator Priority :

The following table provides the list of algorithmic operators, arranged in order of increasing priority, along with their nature and mode of associativity.

Operator	**Nature**	**Priority (lowest to highest)**	**Associativity**
OR	Logical	1 (**lowest**)	Left to right
AND	Logical	2	Left to right
=,≠	Relational	3	Left to right
<, <=, >, >=	Relational	4	Left to right
+,-	Arithmetic	5	Left to right
*,/,%	Arithmetic	6	Left to right
^	Arithmetic	7	Left to right
- (unary) NOT	Arithmetic Logical	8	Right to left
( ) it's not an operator		9 (**highest**)	Left to right

Forms of an Expression:

An expression can be:

- A literal constant. Example: 27, 'A', false, -2.19, "abcd".
- A symbolic constant. Example: Pi.
- A variable. Example: X, Y.
- A function call. Example: sin(X), sqrt(Y).
- A simple expression (a single operator). Example: x*y.
- A complex expression that contains multiple operators. Example: -X + Y/Z^3 - sqrt(Y)*2.

Algorithm construction :

To construct an algorithm, one must first go through the problem analysis step, during which the different problem data, desired results, and various problem-solving steps are determined, allowing us to derive results from the data. The body of the algorithm (the instructions part) consists of a sequence of instructions placed between **Begin** and **End**. Each step in problem resolution (problem analysis phase) is transformed into an algorithmic instruction.

In the declaration part of the algorithm, all objects (variables or constants) used in the resolution are declared.

Note: The algorithm can contain comments starting with // or enclosed between /* and */.

To build an algorithm, we must first go through the step of analysis of the problem which we determine the different data of the problem, the desired results and the different steps of solving the problem that allow us to achieve the results from the data.

Examples:

Problem 1:

Consider a disk with a radius R. Write an algorithm to calculate the surface area of the disk.

Solution:

- **Problem Analysis :** In this phase, we determine the data, results, and the various steps for resolution.
    - **Data :** The radius of the disc, R, and the value of the constant PI=3.14.
    - **Results :** Only one result, which is the surface area of the disc.
    - **The resolution steps** are as follows **:**
        1. Read the value of the disc's radius, R.
        2. Calculate the surface area of the disc using the formula : S = PI * R^2.
        3. Display the value of the disc's surface area (result).

- **Algorithm Writing :**
  The algorithm's instructions represent the resolution steps, and in the declaration section, we declare all objects used in the resolution (R, PI, and S).

Analysis phase		Algorithm writing
		Algorithm disk_surface;
		Const Pi=3.14 ;
		Var R, S : **real** ;
		Begin
• Reading the value of the disc's radius	----->	**Read** (R) ;
• Calculating the disc's surface area	----->	S ← Pi*R*R ;
• Displaying the value of the surface area	----->	**Write**("The surface area of the disc=",S);
		End.

Problem 2 :

Let be a rectangle with length (L1) and width (L2). Write an algorithm to calculate the perimeter and the area of the rectangle.

Solution :

- **Problem Analysis :** In this phase, we determine the data, results, and the various steps for resolution.
    - **Data :** The length (L1) and width (L2) of the rectangle.
    - **Results :** Two results, which are the perimeter (P) and the area (S) of the rectangle.
    - **The resolution steps** are as follows:
        1. Read the value of the length (L1) of the rectangle.
        2. Read the value of the width (L2) of the rectangle.
        3. Calculate the perimeter (P) of the rectangle using the formula: P = (L1 + L2) * 2.
        4. Calculate the area (S) of the rectangle using the formula: S = L1 * L2.
        5. Display the value of the rectangle's perimeter (result 1).
        6. Display the value of the rectangle's area (result 2).

- **Algorithm writing :**
  The algorithm's instructions represent the resolution steps, and in the declaration section, we declare all objects used in the resolution (L1, L2, P, and S).

Analysis phase		Algorithm writing
		Algorithm perimeter_area_rectangle;
		Var L1, L2, P, S: real;
		Begin
• Reading the value of the length (L1)	----->	**Read**(L1);
• Reading the value of the width (L2)	----->	**Read**(L2);
• Calculating the perimeter of the rectangle	----->	P ← (L1 + L2) * 2;
• Calculating the area of the rectangle	----->	S ← L1 * L2;
• Displaying the perimeter of the rectangle	----->	**Write**("The perimeter of the rectangle = ", P);
• Displaying the area of the rectangle	----->	**Write**("The area of the rectangle = ", S);
		End.

2. Translation into Programming Language

To be executed on a computer, an algorithm must be translated into a language understandable by the machine, i.e., a programming language such as Pascal, C, Java, etc. Our examples are translated into the C programming language.

- Translation of the algorithm from Problem 1 into C language results in the following program:

Algorithm		C Program
Algorithm disk_surface;		**# include** <stdio.h>
		main()
		{
Const Pi=3.14 ;	----->	**const float** Pi=3.14 ;
Var R, S : **real** ;	----->	**float** R, S ;
Begin		
Read (R) ;	----->	**scanf**("%f", &R) ;
S ← Pi*R*R ;	----->	S = Pi*R*R ;
Write("The surface area of the disc=",S);	----->	**printf**("The surface area of the disc=%f ",S) ;
End.	----->	}

- Translation of the algorithm from Problem 2 into C language results in the following program:

Algorithm		C Program
Algorithm perimeter_area_rectangle;		**# include** <stdio.h>
		main()
		{
Var L1, L2, P, S: real;	----->	**float** L1, L2, P, S ;
Begin		
Read(L1);	----->	**scanf**("%f", &L1) ;
Read(L2);	----->	**scanf**("%f", &L2) ;
P ← (L1 + L2) * 2;	----->	P = (L1+L2)*2 ;
S ← L1 * L2;	----->	S = L1*L2 ;
Write("The perimeter of the rectangle = ", P);	----->	**printf**("The perimeter of the rectangle =%f",P) ;
Write("The area of the rectangle = ", S);	----->	**printf**("The area of the rectangle = %f",S) ;
End.		}

Note :

During the phase of translating the algorithm into a program written in a programming language, you may encounter two types of errors :

- **Syntax errors :** These are errors caused by incorrect writing (failure to adhere to the syntax) in the programming language. They are noticeable during compilation.
- **Semantic errors :** These are errors caused by a faulty analysis of the problem. They are detected during execution.

3. Program Execution (searching of the solution)

Program 1 (Solving of the Problem 1):

Upon executing Program 1:

- 1st instruction: The machine requests the value of the radius R using the "scanf" instruction. We assume that the value entered by the user is 10.
- 2nd instruction: The machine calculates the value of the surface area and stores it in the memory allocated for the variable S (S = 314).
- 3rd instruction: The machine displays the value of the result (solution) on the screen using the "printf" instruction.

Program 2 (Solving of the Problem 2):

Upon executing Program 2 :

- 1st instruction: The machine requests the value of length L1. We assume that L1 = 10.
- 2nd instruction: The machine requests the value of width L2. We assume that L2 = 5.
- 3rd instruction: The machine calculates the perimeter value and stores it in the space allocated to the variable P (P=30).
- 4th instruction: The machine calculates the surface value and stores it in the space allocated to the variable S (S=50).
- 5th and 6th instructions: The machine displays the value of the perimeter and the value of the area on the screen, respectively.