In this chapter, we present basic notions on algorithmics, i.e. how to write an algorithm that solves a given problem by explaining:

1. Basic actions (assignment, reading, writing),
2. Notions of variable and constant,
3. Types of a variable
4. The declaration of a variable,
5. …

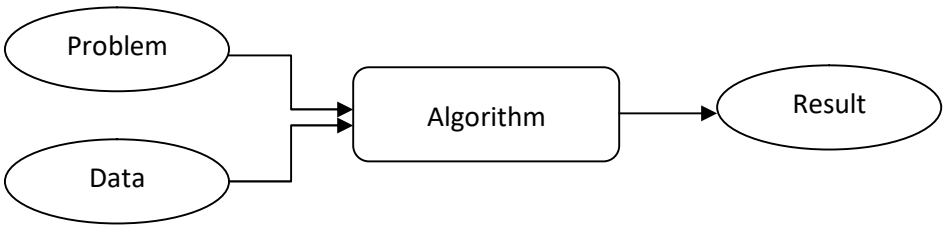
---

1. Notion of algorithm

A computer programme enables the computer to solve a problem, but before we can tell the computer how to solve the problem, we must first be able to solve it ourselves in the form of an algorithm.

1.1. Algorithmics refers to the discipline that studies algorithms and their applications in computer science.

1.2. An algorithm is a method for solving a given problem in a finite time; the word algorithm comes from the name of the famous Arab mathematician Al Khawarizmi (Abu Ja'far Mohammed Ben Mussa Al-Khwarismi)



1.3. The programme will simply be the translation of the algorithm into a programming language, i.e. a language that is simpler than French in its syntax, without ambiguities, that the machine can use and transform to execute the actions it can describe. Pascal, C, Java and Visual Basic are all names of programming languages.

1.4. Algorithm and programme

- The development of an algorithm precedes the programming stage
- A programme is an algorithm
- A programming language is a language understood by the computer
- Developing an algorithm is a demanding problem-solving process
- Writing an algorithm is an exercise in thinking on paper

- The algorithm is independent of the programming language. For example, the same algorithm will be used for an implementation in Java, C++ or Visual Basic.
- The algorithm is the crude resolution of a computer problem.

1.5. Algorithm Representation

Historically, there have been two ways of representing an algorithm:

- Flowchart: graphical representation using symbols (squares, diamonds, etc.)
  - Provides an overview of the algorithm
  - Virtually abandoned today
- Pseudo-code: textual representation with a series of conventions resembling a programming language (this is an informal language close to natural language and independent of any programming language).
  - It is more practical for writing algorithms
  - Its representation is widely used

1.6. Problem solving steps (Algorithm)

1. Understand the problem statement
2. Break the problem down into simpler sub-problems
3. Associate a specification with each sub-problem:
   - The data required
   - The resulting data
4. The process to be followed to arrive at the result, starting from a set of data.
5. Designing an algorithm

1.7. Creating a programme

Solving a given problem involves a succession of stages, as follows:
Problem → Explicit statement → Algorithm → Program
When writing a programme, two types of error can occur:

- **Syntactic errors:** these are noticed at compile time and are the result of poor writing in the programming language.
- **Semantic errors:** these are noticed at runtime and are the result of poor analysis. These errors are much more serious because they can be triggered while the programme is running.

1.8. Programming languages (computer languages)

A programming language is a communication code, enabling a human being to dialogue with a machine by submitting instructions and analyzing the material data supplied by the system. The programming language is the intermediary between the programmer and the machine, Computers “think” in binary — strings of 1s and 0s. Programming languages allow us to translate the 1s and 0s into something that humans can understand and write. A programming language is made up of a series of symbols that serves as a bridge that allow humans to translate our thoughts into instructions computers can understand.

Programming languages fall into two different classifications — low-level and high-level.

Low-level programming languages are closer to machine code, or binary. Therefore, they're harder for humans to read.

High-level programming languages are closer to how humans communicate. High-level languages use words (like object, order, run, class, request, etc.) that are closer to the words we use in our everyday lives. This means they're easier to program in than low-level programming languages

Two types of High-level programming languages are distinguished:

• Procedural languages: Fortran, Cobol, Pascal, C, …

• Object-oriented languages: C++, Java, C#,…

Choosing a programming language isn't easy: each one has its own specificities and is better suited to certain types of use. The most popular programming languages include the following: Python, Java, JavaScript, C#, C++, PHP, R, Swift, Kotlin.

1.9. Basics of an algorithmic language

1.9.1. Basic structure

In pseudo-code, the general structure of an algorithm is as follows:

Algorithm name_of_the_algorithm

CONST { Defining constants }

TYPE { Defining types } } they are optional (if there is no var or const or type, they are not written)

VAR { Declaring variables}

BEGIN

{ Sequence of instructions }

END.

- **Header part:** this is the first line of the algorithm, starting with the word algorithm followed by a space followed by the name of the algorithm.

- **Declarations section:** in this section, all the objects used to solve the problem must be declared.
- **Processing section:** this section begins with the word Begin and ends with the word End. It contains the actions used to solve the problem.

1.9.2. Basic instructions

An algorithm is made up of four types of instruction considered as small basic blocks:

1. Variable assignment
2. Reading and/or writing
3. Testing
4. Loops

Before describing these instructions, we will first introduce the notion of variable and constant.

1.9.3. Constants and variables

Objects are divided into two classes: constants and variables.

1.9.3.1. Notion of variable

The data manipulated in an algorithm are called variables. In a programming language, a variable is used to store the value of a datum. It designates a memory location whose contents can change during the course of a programme (hence the name variable).
Each memory location has a number that allows it to be referred to uniquely: this is the memory address of that cell.
Rule: The variable must be declared before being used, and must be characterised by :

- A name (Identifier)
- A type that indicates the set of values that the variable can take (integer, real, Boolean, character, string, etc.)
- A value

Identifiers : rules
The choice of name for a variable is subject to a few rules which vary according to the language, but in general :
- A name must begin with an alphabetical letter. Example: E1 (1E is not valid)
- Must consist solely of letters, numbers and underscores ("_") (avoid punctuation characters and spaces). Examples: SMI2008, SMI_2008. (SMP 2008, SMP-2008, SMP;2008: are invalid)
- Must be different from the language's reserved words (for example in C: int, float, double, switch, case, for, main, return, ...)
- The length of the name must be less than the maximum size specified by the language used (in C, the name must be no longer than 32 characters).
- In c, upper case is case sensitive: a and A are two different names.

Tip: to make the code easier to read, choose meaningful names that describe the data being manipulated. Examples: Student _Mark, Price_TTC, Price_HT
Note: in algorithmic pseudo-code, we will respect the rules cited above, even if we are free in the syntax.

Variable types

- The type of a variable determines the set of values it can take. The types offered by most languages are :
- Numeric type (integer or real)
- Byte (encoded in 1 byte): from [-27,27[or [0, 28[.
- Short integer (coded on 2 bytes): [-215,215[
- Long integer (coded on 4 bytes): [-231,231[
- Single-precision real (coded on 4 bytes): precision of order 10-7
- Double precision real (coded on 8 bytes): precision of order 10-14
- Logical or Boolean type: two values TRUE or FALSE
- Character type: upper case letters, lower case letters, digits, symbols, ... Examples: 'A', 'b', '1', '?', ...
- String type: any sequence of characters. Examples: " " Last name, First name", "postal code: 1000", ...

A) Integers: To represent integers, the operations that can be used on integers are :

- All basic operations are permitted: + , - , * , /
- The classic comparison operators: <, >, =, ...
- Division / is Euclidean (or integer) division. Ex: 11 / 4 = 2 and not 2.75!)
- The modulo % operator gives the remainder of the Euclidean division. Ex: 11%4 = 3

B) Real: To represent real numbers, the operations that can be used on real numbers are :

- The classic arithmetic operations: + (addition), - (subtraction), * (product), / (division)
- Standard comparison operators: <, >, =, ...
- Division / gives a decimal result
- The modulo % operator does not exist

C) Boolean: A logical variable (Boolean) can take two values: TRUE or FALSE. The main operations used are :

- Logical operators: NOT, AND, OR

Truth table :

A	B	A ∧ B	A ∨ B
FALSE	FALSE	FALSE	FALSE
FALSE	VRAI	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

A	NON A
FALSE	TRUE
TRUE	FALSE

• Comparison operators: = , ≤ , ≥ , ≠

D) Character

This is the domain made up of alphabetic and numeric characters. A variable of this type can only contain a single character. The basic operations that can be performed are comparisons: <, >, =, ...

E) String

A string is an object that can contain several characters in an ordered way.

Variable declaration

Remember: any variable used in a program must have been declared first. In pseudo-code, variables are declared using the following form :

Variable list of identifiers **:** type

Example:

Variable
i, j, k : integer
x , y : real
OK : boolean
Ch1, ch2 : string

In C: basic types :

- **char (characters) For example: 'a'...'z', 'A'...'Z',...**
- **int (integers) For example: 129, -45, 0, ...**
- **float (real numbers) For example: 3.14, -0.005, 67.0, ...**

In general, their declaration is as follows:
<type><name> ;
<type><name1>,<name2>,<name3> ;
Examples :

```
int a;
float value, res;
char a,b,c ;
```

Declaring a variable means reserving a memory location for it. We don't know its initial value!

1.9.3.2. Notion of constant

A constant is a variable whose value does not change during program execution. It can be a number, a character or a string of characters. In pseudo-code,
Constant identifier = value (by convention, constant names are in uppercase)
Example: to calculate the area of circles, the value of pi is a constant but the radius is a variable.

- Constant PI=3.14, MAXI=32
- A constant must always be given a value as soon as it is declared.

In C :

1. **Literal constants: Exp. Int a=10; float tax_rate = 0.28;**
2. **Symbolic constants**
   - 2.1. **#define PI 3.14159 (#define can be found anywhere in the source code, but its effect is limited to the part of code that follows it. Generally, programmers group all #define together at the beginning of the file, before the main() function).**
   - 2.2. **const float pi = 3.14159;**

1.9.4. Basic operations

In what follows, we describe the list of basic operations that can make up an algorithm. They are described in pseudocode (pseudo language).

1.9.4.1. Assignment

Assignment consists of assigning a value to a variable (i.e. filling or modifying the contents of a memory area). In pseudo-code, assignment is denoted by the sign ←

- Var← e: assigns the value of e to the variable Var
- e can be a value, another variable or an expression
- Var and e must be of the same type or of compatible types

Assignment only modifies what is to the left of the arrow

Example 1 : i ←1, j ←i , k ←i+j , x ←10.3 , OK ←FAUX , ch1 ←"SMI" , ch2 ←ch1 , x ←4 , x ←j

(avec i, j, k : entier; x :real; ok :boolean; ch1,ch2 : string)

Invalid examples: i ←10.3 , OK ←"SMI" , j ←x

Example 2

a ←10 a receives the constant 10

a ← (a*b)+c a receives the result of (a*b)+c

d ←'m' d receives the letter m

Remarks

- The C programming language uses the equal sign = for the assignment ←.
- When an assignment is made, the expression on the right is evaluated and the value found is assigned to the variable on the left. Thus, A←B is different from B←A
- Assignment is different from a mathematical equation:
- The operations x ← x+1 and x ← x-1 have meaning in programming and are called incrementing and decrementing respectively.
- A+1 ← 3 is not possible in programming languages and is not equivalent to A ← 2.
- Some languages give default values to declared variables. To avoid any problems it is preferable to initialise declared variables.

Example in C

```
int n;
int p;
 n = 10;
 p = 2*n-3;
```

Some useful operators

- i + + : operator used to add a unit to the variable i (of type int or char)

- i - - : same as above, but to remove a unit

- **- x* = y, x/ = y, x- = y, x+ = y: operators for multiplying (dividing, subtracting or adding) x by y (no restriction on type**)

1.9.4.2. Reading

Read (variable)

This operation assigns to a variable a value entered using an input device (usually the keyboard).

Examples of reading

Read(a) The user is asked to enter a value for a

Read(a,b,c) The user is asked to enter 3 values for a, b and c respectively

In C :

Input with scanf.

- Reads from the standard input (the keyboard)

- scanf("<code format>", &<variable>); with <code format> = %d, %f or %c to read an integer, float or character.

Examples

- int n ; scanf(‘‘ %d ‘‘, &n);
- float x ; scanf(‘‘ %f ‘‘, &x);
- char a ; scanf((‘‘ %c ‘‘, &a);

Note: The program stops when it encounters a Read instruction and does not continue until the input expected by the keyboard has been entered and the Enter key pressed (this key signals the end of the input).

Tip: Before reading a variable, it is strongly recommended that you write messages on the screen to warn the user what to type.

1.9.4.3. Writing

Write (expression)

It communicates a given value or the result of an expression to the output device.

Writing examples

Write('hello') Displays the message hello (constant)

Write(12) Displays the value 12

Write(a,b,c) Displays the values of a, b and c

Write(a+b) Displays the value of a+b

Write(a, b+2, "Message") Displays the value of a, then the value of the expression b+2 and finally the word "message".

Examples in C

- printf(‘‘ hello /n ‘‘);
- int x=10; printf(‘‘value of x = %d/n ‘‘ , x);
- int x=10; float y = 3.4; printf(‘ ‘value of x = %d et de y+x = %f/n ‘‘ , x , y+x);

Main format codes

- %d for the type int
- %c for the type char
- %f for the type float

Formatting characters

- /n: line feed
- /r: return to start of current line
- /t: tab

1.9.5. General syntax of the algorithm

Example algorithm
Constant var1 = 20 , var2 = "hello! "
Variable
var3, var4 : real
var5: character
Begin // body of the algorithm

```
    Instruction 1
    Instruction 2
    …….
    Instruction n
```

End

Notes:

- The instructions in an algorithm are usually executed one after the other, in sequence (top to bottom and left to right).
- The order of execution is important
- This sequence cannot be changed arbitrarily

1.9.6. Expressions and operators

An expression can be a value, a variable or an operation made up of variables linked by operators.
Examples: 1, b, a*2, a+ 3*b-c, ...
Evaluating the expression provides a single value, which is the result of the operation.
The operators depend on the type of operation:

- Arithmetic operators: +, -, *, /, % (modulo),^(power)
- Logical operators: NOT(!), OR(| |), AND (&&)
- Relational operators: =, <, >, <=, >=
- String operators: & (concatenation)
- An expression is evaluated from left to right, taking into account the priorities of the operators.

Note:

- An integer and a character cannot be added together.
- However, in some languages, an operator can be used with two operands of different types, as is the case with arithmetic types (4 + 5.5).
- The meaning of an operator can change depending on the type of operands, for example
    - The + operator with integers performs addition, 3+6 is 9
    - With character strings, it performs concatenation: "hello" + "everyone" is "hello everyone".

1.9.7. Operator priority

For the arithmetic operators given above, the order of priority is as follows (from highest to lowest priority) :

- () : parentheses
- ^ : (raising to a power)
- *, / : (multiplication, division)
- % : (modulo)
- +, - : (addition, subtraction)

Example : 9 + 3 * 4 is 21

- Where necessary, brackets are used to indicate the operations to be performed first. Example: (9 + 3) * 4 is 48
- With equal priority, the expression is evaluated from left to right

1.9.8. Boolean operators

- Associativity of the operators and and or: a and (b and c) = (a and b) and c
- Commutativity of the operators and and or: a and b = b and a; a or b = b or a
- Distributivity of the operators and and or: a or (b and c) = (a or b) and (a or c); a and (b or c) = (a and b) or (a and c).
- Involution (reciprocal homography): non non a = a; Morgan's Law: non (a or b) = non a and non b; non (a and b) = non a or non b.

Example : let a, b, c and d be any four integers:
(a<b)||((a>=b)&&(c==d)) ⇔ (a<b)||(c==d)car (a<b)||(!(a<b)) always true

---

Home Exercises

Exercise 1 :

What is the type of each variable :

A=1, B=true, test= 12.23, speciality='m',

Exercise 2 :

Let A, B be two variables of integer type, C, D two variables of real type, E, F two variables of Boolean type.

What is the type of the following variables : A1, B1, C1, A2, B2, C2, D2, A3, B3, C3, D3

A1 ← A+B ; B1←A*B; C1←A/B;

A2←C+D; B2←C*D; C2←C/D; D2←A*C;

A3←E or F; B3←E and F; C3←(A>B) ; D3← false ;

Exercise 3 :

Which identifiers are valid and which are invalid:

A, cA, 12, 1A, A1, A12m, Batna, test?, if, ex1+ex2 , exo 1, éxo1, int .

Exercise 4 :

What are the values of variables A, B and C after executing the following instructions?

Variable A, B, C : Integer

Begin

A ← 7

B ← 17

A ← B

B ← A+5

C ← A + B

C ← B – A

End

Exercise 5 :

What are the values of variables A and B after executing the following instructions?

Variable A, B : Integer

Begin

A ← 6

B ← 2

A ← B

B ← A

End

Can the last two instructions be used to exchange the values of A and B?

Exercise 6 :

Write an algorithm to exchange the values of two variables A and B?

Exercise 7 :

Write an algorithm to enter a surname and first name and then display the full name.

Exercise 8 :

Write an algorithm that asks the user for an integer number, then calculates and displays the square of that number.

Exercise 9

We have three variables A, B and C. Write an algorithm that transfers the value of A to B, the value of B to C and the value of C to A (whatever the previous contents of these variables).

Exercise 10 :

Write a C program to enter a number and display its double and half.

Exercise 11 :

Write a an algorithm to calculate and display the result of the expression : $(A+B)*x^2/(C+5)$

Exercise 12 :

Write a an algorithm to display the message " university of Batna 2 ".

Exercise 13 :

write an algorithm to calculate and display the surface area of a cylinder.