Democratic and Popular Republic of Algeria
Ministry of Higher Education And Scientific Research
University A. MIRA of Béjaïa

Faculty of Exact Sciences
Department of Computer Science

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

L1 Computer Science

L1 Economy – CS

L1 Law - CS

Algorithmic and Data Structures 1



Dr. Meriem BENADDA BENYAHIA
Senior Lecturer B

© January 2026

Preface

Algorithms and data structures form the foundations of any education in computer science. Before learning to program in a specific language, it is essential to understand how to design, analyze, and represent the logic of a problem in a rigorous and efficient manner. This course aims to introduce first-year students to algorithmic thinking through a progressive approach, moving from the resolution of simple problems to the design of more structured algorithms.

Students will discover the fundamental principles of algorithmics: the concepts of variables, data types, control structures (sequential, conditional, and iterative), as well as modeling methods using pseudocode and flowcharts. In a second part, the course addresses data structures that make it possible to represent and efficiently manipulate information in memory, such as arrays, lists, and their variants.

The adopted approach is both theoretical and practical: each concept is accompanied by examples, exercises, and reflective activities that foster understanding and implementation of key ideas. This material thus aims to develop students' rigor in reasoning, autonomy in problem-solving, and a solid foundation for subsequent learning in programming and advanced areas of computer science.

Dr. M. BENADDA

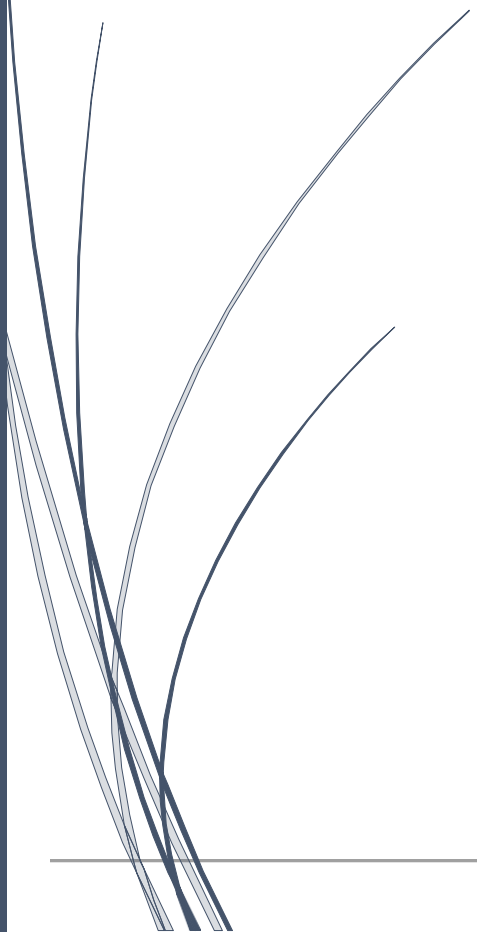


Table Of Content

Chapter 1: Basic Elements of Algorithmics 6
1.1 Introduction 6
1.2 Notion of algorithm 6
1.2.1 Origin 6
1.2.2 Definition 6
1.3 Programming Language 7
1.4 Steps to Solve a Problem 7
1.5 Structure of an Algorithm 7
1.6 Variables and Constants 8
1.6.1 Variable 8
1.6.2 Variable Declaration 8
1.6.3 Constant 8
1.6.4 Declaration of constants 9
1.7 Identifier Concept 9
1.8 Basic Data Types In Algorithmic 9
1.9 Expressions 10
1.9.1 Arithmetic expressions 10
1.9.2 Relational expressions 10
1.9.3 Logical expressions (Boolean) 11
1.9.4 Expression Evaluation 11
1.10 Basic Instructions in Algorithmics 12
1.10.1 Read Instruction 13
1.10.2 Write Instruction 13
1.10.3 Assignment Instruction 14
1.11 Execution Trace 15
Conclusion 16
Chapter 2 : Alternative Control Structures (Conditions) 19
2.1 Introduction 19
2.2 Purpose 19
2.3 Condition Concept 19
2.4 Simple Alternative Test (If.. then) 20
2.5 Double Alternative Test (If .. Else) 20
2.6 Multiple Choice (Nested Conditional Structure or Switch) 21
2.7 Flowchart Representation (Algorigram) 22
2.8 Conditional Structures in Flowcharts 23
Conclusion 24

Knowledge Assessment Test 24
Correction: 25
Chapter 3 : Iterative Control Structures (Loops) 26
3.1 Introduction 26
3.2 Definition 26
3.3 Types of loops 26
3.4 The“While” loop 27
3.5 The “Repeat...until” loop (do...while) 27
3.6 The “For” loop 28
3.7 Iterative Structures in Flowcharts 29
3.8 Loops for Input Control (Validation) 30
Knowledge Assessment Test 31
Chapter 4: Arrays 33
4.1 Introduction 33
4.2 Data Structures 33
4.3 Definition of an Array 34
4.4 One-Dimensional Array (Vector) 34
4.5 Manipulating a Vector 35
4.6 Two-Dimensional Array (Matrix) 36
4.7 Manipulating a Matrix 37
4.8 Common Examples 39
Conclusion 40
Chapter 5: Subroutines (Procedures and Functions) 41
5.1 Introduction **41**
5.2 Concept of a subroutine **41**
5.3 Declaring a subroutine **42**
5.3.1 Declaring a function 42
5.3.2 Declaration of a procedure 43
5.4 Passing Parameters and Scope of Variables **43**
5.5 Subroutine calls **44**
5.6 Scope of variables **45**
5.7 Recursion **46**
5.8 Steps of transformation of a function into a procedure and a procedure into a function **47**
Conclusion **47**
Exercises 48
1: Basic elements of algorithmics 48
Solution 50
2: Alternative Control Structures (Conditions) 54
Solution 55
3: Iterative Control Structures (Loops) 59
Solution 60

4 : Arrays....64
Solution....65
5 : Subroutines....71
Solution....72



Chapter 1: Basic Elements of Algorithmics

What is an algorithm? Why is the study of algorithms important? What is the role of algorithms in relation to other computer technologies?

This chapter answers these questions.

1.1 Introduction

Algorithmics is a fundamental discipline in computer science. It is the science that studies how to solve a problem by describing, step by step, the sequence of instructions required to reach a solution.

Before writing a program in a programming language, it is essential to go through the algorithmic phase in order to design a clear, logical, understandable, and executable method for a computer [3].

1.2 Notion of algorithm

1.2.1 Origin

The term algorithm originates from the name of the Persian mathematician Muḥammad ibn Mūsā al-Khwārizmī (9th century), considered one of the founders of algebra.

When his works were translated into Latin in the 12th century, his name was transcribed as Algoritmi, which later evolved into algorithmus and finally into the modern word algorithm.

1.2.2 Definition

An algorithm is a finite and ordered sequence of operations (instructions) that allows solving a problem. An algorithm must satisfy the following fundamental characteristics [3],[2]:

- **Finiteness:** The algorithm must have a finite number of steps.
- **Precision:** Each instruction must be clearly and unambiguously defined.
- **Efficiency:** The algorithm must produce a result in a reasonable time.
- **Inputs and outputs:** An algorithm must receive input data and produce output results.

1.3 Programming Language

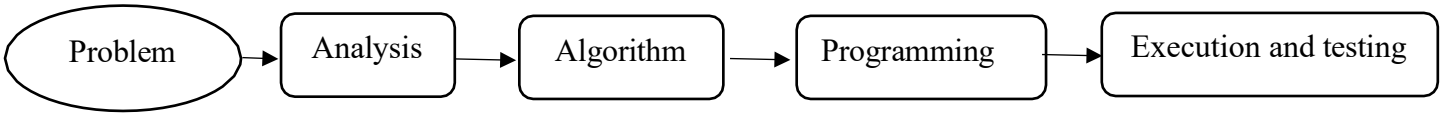
A programming language is a formal means used to write algorithms in a form understandable by a machine. There are several types of programming languages [4]:

- **Low-level languages:** close to machine language (e.g., Assembly).
- **High-level languages:** more readable for humans (e.g., C, Python, Java).

A computer program is the concrete implementation of an algorithm using a programming language.

1.4 Steps to Solve a Problem

Solving a computer problem generally follows these steps [5]:



1. **Problem analysis:** Understanding the problem, identifying inputs, outputs, constraints, and special cases.
2. **Algorithm design:** Developing a logical sequence of instructions.
3. **Programming:** Translating the algorithm into a programming language.
4. **Execution and testing:** Running the program and validating results with different test cases.

From the following section onward, each algorithmic concept will be accompanied by its equivalent in the C language, which is the programming language used in the practical sessions.

1.5 Structure of an Algorithm

An algorithm is presented in a structured and readable form consisting of three main parts:

```
Algorithm id_Algorithm ; // Header

Variables <variables declaration> ;
Constants <constants declaration> ;          // Declaration

Begin
   <instruction_1> ;
      .
      .                                  // Body of the algorithm
      .
   <instruction_N> ;

End.
```

a. **Header:**
Contains the identifier of the algorithm; it is a name that identifies the algorithm and must comply with certain rules that we will discuss later in the course.

Example :

Algorithm CalculateSum ;

b. **Declaration :**
Part where variables, their types, and possibly constants are defined.

c. **Body (or instruction block):**
Main part that describes the instructions to be executed step by step; it is delimited by the keywords *Begin* and *End*.

In C language:

```
#include<stdio.h>
int main() {
      //instructions
return 0;
}
```

1.6 Variables and Constants

1.6.1 Variable

A variable is a memory location identified by a name and containing a value that can change during program execution.

1.6.2 Variable Declaration

Variables are declared using the keyword `Variables` (Once a time for all variables), following by the identifier and its type. The general form is as following :

```
Variables id_variable : type ;
```

In C language :

```
type id_variable;
```

Example :

```
Variables x : integer;
          y:real   ;   →
```

In C

```
int x ;
float y ;
```

Note: Variables with the same type can be declared together :

Example :

```
Variables a,b,c : integer; →
```

In C language

```
int a,b,c;
```

1.6.3 Constant

A constant is a fixed value that cannot be modified after declaration. Its purpose is to avoid repeated direct use of literal values.

Let us imagine that an algorithm uses the value 3.14 about fifteen times and that we want to replace this value with a more precise one, namely 3.14159. In this case, we would have to modify all occurrences of 3.14. On the other hand, if we use a constant PI = 3.14, we would only need to modify it once.

1.6.4 Declaration of constants

Constants are declared using the keyword **Constants** (used only once for all constants in algorithmic declaration), followed by the identifier and the value of the constant. The general form is:

```
Constants id_constant = value ;
```

Example : **In C language :**

```
Constants PI = 3.1415 ;     →    const float PI = 3.1415 ;
```

Note: In C, each constant declaration must be preceded by the keyword **const**, unless the constants have the same type. **Example** `const int a=15 , b=10 , c = 20 ;`

1.7 Identifier Concept

An identifier is the name given to a variable, a constant, a function, or even to the algorithm itself. It must comply with certain rules:

- It can only Begin with an alphabetic character (letter) or an underscore (_).
- It can only contain alphanumeric characters (letters and digits) and underscores (_); therefore, it cannot contain special or punctuation characters such as #, space, $, &, =, <, etc., or Greek letters like π, β, Ω, etc.
- It must be unique.
- In a program, an identifier cannot be a reserved keyword of the programming language. Example: In C, one cannot use **for**, **while**, **if**, **until**, etc.

1.8 Basic Data Types In Algorithmic

Basic types are the fundamental categories of data used in algorithmics (programming). Here are the main types with their memory representation:

- **Integer (int, long):** Represents the set $\mathbb{Z}$ of integers. Example: -3, 0, 20075, etc.
- **Real (float, double):** Represents the set $\mathbb{R}$ of real numbers. Example: 3.14, -0.5, 123.789, etc.
- **Character (char):** Can be any symbol found on a computer keyboard: space, letters, digits, punctuation marks, and special characters (#, @, &, etc.).
- **String (char variable_name[n], where n is the number of characters):** A sequence of characters, stored in memory as an array of characters.
- **Boolean (bool):** A boolean data type can have two values: **true** or **false**. It is theoretically stored in 1 bit (a bit—short for *binary digit*—is the smallest unit of information in computing and can only take the values 0 or 1. A byte consists of 8 bits).

Examples :

Algorithmic	**In C language :**
Variables name : String [20] ; →	`char name[20];`
Constants name = "BENADDA" ; →	`const char name[10] = "BENADDA" ;` or `const char name[] = "BENADDA" ;`

Once the variables and constants have been introduced, we can address the notion of expression, which represents the way data are combined and processed to produce new values within an algorithm.

1.9 Expressions

Definition

An expression is a combination of constants, variables, and operators (arithmetic, relational, logical, etc.) that can be evaluated to produce a value.

Expressions are essential elements of an algorithm's calculation, comparison, or decision instructions. In algorithmics, expressions can be arithmetic or logical (Boolean).

1.9.1 Arithmetic expressions

An arithmetic expression consists of numeric operands (values, variables, constants) connected by arithmetic operators.

Operation	Algorithmic	In C
Addition	+	+
Subtraction	-	-
Multiplication	*	*
Real division	/	/
Integer division	div	/
Modulo	mod	%

The modulo is the remainder of the entire division.

1.9.2 Relational expressions

A relational expression uses relational (or comparison) operators. The result is Boolean (true or false).

Operation	Algorithmic	In C
Equality	=	==
Inequality	≠	!=
Strict superiority	>	>
Strict inferiority	<	<
Greater than or equal	>=	>=
Less than or equal	<=	<=

1.9.3 Logical expressions (Boolean)

A logical expression is a combination of Boolean variables and Boolean operators. It can combine relational expressions. The result is Boolean.

Operation	Algorithmic	In C
Negation	Not	!
Conjunction	and	&&
Disjunction	or	||

1.9.4 Expression Evaluation

The evaluation of an expression consists of calculating its value by applying the order of priority of the operators.

In algorithmics as in C, some operators take precedence over others, which influences the final result of a calculation.

1.9.4.1 Operator precedence (from highest to lowest priority)

When evaluating an expression, operators are not executed in the order in which they are written, but according to their priority level. Here is the order, from highest to lowest:

1. **Parentheses** : `()`
   Used to enforce the order of evaluation. Example : `(a + b) * c`

2. **Functions calls**

   Functions are evaluated before they are used in a calculation.

   Example : `x + y * 3 - sqrt(a + b)` → The `sqrt()` function is executed before any other operation.

3. Unary operators **:** - (negative sign), Not (logical negation)
4. Multiplication, division and modulo
5. Addition and subtraction
6. Relational comparisons
7. Equality or inequality tests
8. Logical And
9. Logical Or

Examples of common functions in algorithmics and C language

Function (algorithmic)	Role/ Description	C language equivalent	Required C Library	Example of use in C
abs(x)	Gives the absolute value of x	abs(x)/ fabs(x)	<stdlib.h> / <math.h>	val = abs(-5); // 5
root(x)	Calculate the square root of x	sqrt(x)	<math.h>	r = sqrt(9); // 3
power(x, y)	Calculate x to the power of y	pow(x, y)	<math.h>	p = pow(2, 3); // 8

int(x)	Give the integer part of x	floor(x) / ceil(x)	<math.h>	e = floor(3.7); // 3
				
				e = ceil(3.7); // 4

round(x)	Round x to nearest integer	round(x)	<math.h>	a = round(3.6); // 4
random(n)	Generates a random number less than n	rand() % n	<stdlib.h>	x = rand() % 10;
length(String)	Gives the length of a string	strlen("String")	<string.h>	n = strlen("test"); // 4
uppercase(String)	Converts a string to uppercase	strupr("String")	<string.h>	strupr(nom);
lowercase(String)	Converts a string to lowercase	strlwr("String")	<string.h>	strlwr(prenom);

Notes:

- Operators of the same priority level are evaluated from left to right.
- The use of parentheses is recommended to avoid ambiguity and to clarify the reading of the code.

After establishing the order of precedence of the operators, it is necessary to introduce the truth table, which describes how logical expressions are evaluated according to the different possible values of Boolean variables.

1.9.4.2 Truth table

A truth table shows the result (True or False) of a logical expression for all possible combinations of the variables' values.

A	B	Not A	A and B	A or B
True	False	False	False	True
True	True	False	True	True
False	True	True	False	True
False	False	True	False	False

After studying variables, constants, and expressions, it is now necessary to understand how they are used in a program.

1.10 Basic Instructions in Algorithmics

Basic instructions are the building blocks that make it possible to write any algorithm: they make it possible to act on the data, to perform calculations and to communicate with the user. They essentially include: the read instruction, the write instruction, and the assignment instruction.

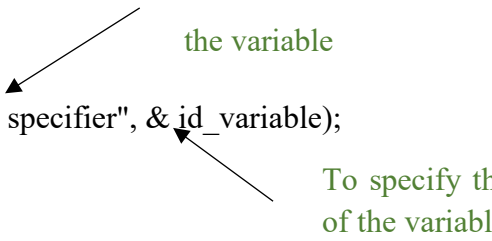
1.10.1 Read Instruction

Read is an instruction that allows data to be entered from the keyboard and stored in variables for use in the algorithm. Only variables can be read.

Syntax:

Read(id_variable) ;

In C language :



Examples :

scanf("%d", & id_variable); // for an integer or a boolean, we can also use %i

scanf("%f", & id_variable); // for a real

scanf("%c", & id_variable); // for a character

scanf("%s", id_variable); // for a string

Note: A string is read without specifying its address (without &).

1.10.2 Write Instruction

Write is an instruction that allows messages or results to be displayed on the screen.

Syntax :

Write ("Message to display") ; // Displays a message
Write (id_variable) ; // Displays the value of the variable
Write (expression) ; // Displays the value of the expression
Write (value) ; // Displays the value

In C language :

printf("Message");
printf("format specifier", id_variable);

printf("format specifier", expression);

printf("format specifier", value);

Note : A message and a value can be grouped together in the same script (in algorithmics and in C) by separating them by a comma called a concatenation comma.

Examples:

Write ("The result is:");
Write (R); → Write ("The result is:", R) ;

In C :

printf("The result is: %d", id_variable);

Algorithm for reading and writing an integer:

Algorithm ReadDisplay;
Variables x: integer;
Begin
Write("Enter an integer:");
Read(x);
Write("The value of x is :", x);
End.

In C language:

```
#include <stdio.h>
int main() {
    int x;
    printf("Enter an integer:");
    scanf("%d", &x);
    printf("The value of x is : %d\n",x);
    return 0;}
```

1.10.3 Assignment Instruction

Assignment is an instruction that allows a value to be assigned to a variable. This value can be a literal value, the value of another variable, or the result of an expression..

Syntax :

id_variable ← valeue ; id_variable ← id_variable; id_variable ← expression ;

In C :

id_variable = value ; id_variable = id_variable; id_variable = expression ;

Notes :

- When a value is assigned to a variable for the first time using an assignment, this operation is called **initialization**.
- When the value of a numeric variable is increased by a certain amount, most often by 1, this operation is called **incrementing**. It is widely used in loops (Chapter 3) and counting operations.

Example: Calculating the average of a module

Algorithm Average;
Variables TS, Ex, avg: real;
Begin
  Write("Enter the TS grade:");
  Read(TS);
  Write("Enter the exam grade:");
  Read(Ex);
  avg ← (TS * 2 + Ex * 3) / 5;
  Write("Average = ", avg);
End.

In C language:

```
#include <stdio.h>
int main() {
    float TS, Ex, avg;
    printf("Enter the TS grade:");
    scanf("%f", &TS);
    printf("Enter the exam grade:");
    scanf("%f", &Ex);
    avg = (TS * 2 + Ex *3) / 5;
    printf(" Average = %.2f\n ", avg);
    return 0;
}
```

Before Begining a new chapter, it is important to learn how to follow step by step the execution of an algorithm in order to understand how it works. This is called an execution trace.

1.11 Execution Trace

An execution trace is a table that allows the step-by-step visualization of the execution of an algorithm. It shows:

- The evolution of variable values
- The content of the outputs produced during execution

It is a very useful analysis and verification tool to ensure that the algorithm works as expected.

The execution trace is generally presented in the form of a table with three main columns, as shown in the following table:

Instructions	**Variables**			**Display**
	V1	**…**	**Vn**	
Instruction1 ;				
…..				
InstructionN ;				

Example :

Algorithme Trace ;
Variables A, B, C : integer ;
Begin
  A ← 3 ;
  B ← 5 ;
  C ← A * 2 + B ;
  Write("The sum is:", C) ;
End.

Execution Trace

Instructions	Variables			Display
	A	B	C	
A ← 3 ;	3	/	/	/
B ← 5 ;	3	5	/	/
C ← A * 2 + B ; C ← 3 * 2 + 5 = 11	3	5	11	/
Write("The sum is:", C) ;	3	5	11	The sum is: 11

Conclusion

In this first chapter, we learned about the fundamental basics of algorithmics: variables, constants, expressions, and basic instructions.

However, in order to make an algorithm more dynamic and capable of adapting its behavior, it is essential to use control structures that allow decision-making and the repetition of actions.

Knowledge Assessment Tests

Q1. What is an algorithm?

Q2. Name the main steps in solving a problem in algorithmics.

Q3. Give an example of a programming language.

Q3. What is the difference between a variable and a constant?

Q4. Which basic data type is used to store a number with decimals?

Q5. Multiple Choice Questions (MCQs)

An algorithm must be:

a) Infinite

b) Ambiguous

c) Precise and finished

d) Executed only on paper

Q6. What is the role of the programming language ?

a) Translate an algorithm into instructions understandable by the machine

b) Replace the algorithm

c) Test only the program

d) Store data

Q7. What is the memory representation of a Boolean ?

a) 1 bit

b) 1 byte

c) 2 bytes

d) 4 bytes

Q8. An identifier can Begin with:

a) A digit

b) A space

c) A letter

d) A special character (@, #, etc.)

Q9. Which basic data type is used to store a single letter ?

a) Integer

b) Boolean

c) Character

d) Real

Q10. Arrange the following operators according to their precedence: +, *, &&, <, ()

Q11. In scanf("%d", &x);, what does the symbol & represent ?

Q12. What is an execution trace and what is it used for ?

Q13. What is the memory representation of a Boolean ?

a) 1 bit

b) 1 octet

c) 2 octets

d) 4 octets