People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Mohamed El Bachir El Ibrahimi University – Bordj Bou Arréridj
Faculty of Mathematics and Computer Science
Department of Computer Science

Course

Algorithmic and Data Structures 1

With Solved exercises in algorithmic language and C language

Dr. SAIFI Lynda

2022/2023

Foreword

This course is intended for first-year students in Mathematics and Computer Science, as well as Computer Science Engineering, with the aim of providing them with a reference document that is both comprehensive and concise.

This document provides simple definitions and examples, in both algorithmic formalism and the C language, for all the functionalities and concepts outlined in the course syllabus, in order to facilitate students' understanding and practice.

Prerequisites

It is recommended to have knowledge of:

- Basic mathematical and logical concepts
- Basic computer functions

Targeted Skills

Upon completion of this course, the student will be able to:

- Analyze a real problem and potentially break it down into sub-problems.
- Understand the difference between the role of a programmer and that of a user.
- Construct an arithmetic or logical expression using algorithmic formalism.
- Write the solution to an analyzed problem in the form of an algorithm.
- Execute the algorithm and identify its shortcomings and bugs.
    - Master the required data structures.

Summary

Chapter 5: Arrays and Strings

1. Introduction
2. The type “array”
    2.1. Syntax
    2.2. Syntax in C language
    2.3. Example
    2.4. Array operations
3. Multidimensional arrays (Matrix)
    3.1. Syntax
    3.2. Syntax in C language
    3.3. Example
4. Strings
    4.1. Syntax
    4.2. Syntax in C language
    4.3. Example
    4.4. String operations in C language

Chapter 6: Custom Types

1. Introduction
2. Enumerations
    2.1. Definition
    2.2. Syntax
    2.3. Enumeration Types
        2.3.1. Enumerated Type
        2.3.2. Interval Type
        2.3.3. Examples
        2.3.4. Examples in C language
        2.3.5. Remarks
3. Record (Structure)
    3.1. Definition
    3.2. Syntax
    3.3. Example
    3.4. Accessing a field of a record
    3.5. Example
    3.6. Example in C language

Part 2: Solved exercises

General information about the C language
Tutorial 1: Input / Output Statement
Solution
Tutorial 2: Conditional Statement

Solution
Tutorial 3: Loops
Solution
Tutorial 4: Arrays and strings
Solution

Bibliographic references

1 Introduction

1. A Short History of Computer Science

Computer science is a field that has evolved rapidly over the past century, transforming the world in the process. Here is a brief overview of its history:

- **Pre-20th Century:** The foundations of computer science can be traced back to ancient civilizations that developed simple calculating devices. Notable examples include the abacus in ancient China and the Antikythera mechanism in ancient Greece.
- **19th Century:** The 1800s saw significant developments in mathematics and logic, laying the groundwork for computer science. George Boole's Boolean algebra provided a mathematical framework for binary logic, which is fundamental in modern computing. Charles Babbage conceived and designed the Analytical Engine, an early mechanical computer, although it was never built during his lifetime.
- **Early 20th Century:** The early 20th century witnessed the birth of electronic computing. Alan Turing, a British mathematician, and logician, developed the concept of the "Turing machine," a theoretical device that can simulate the logic of any computer algorithm. His work was pivotal in defining the limits of computation.
- **1940s:** The first operational electronic computers emerged during World War II. The ENIAC (Electronic Numerical Integrator and Computer) and the Colossus were among the earliest electronic digital computers. These machines were massive and primarily used for military and scientific calculations.
- **1950s:** This decade saw the development of high-level programming languages like Fortran and LISP, making it easier to write software for computers. The term "artificial intelligence" was also coined during this period.
- **1960s:** The invention of the microchip or integrated circuit revolutionized computing. It allowed for the miniaturization of electronic components, making computers smaller, faster, and more affordable. The concept of time-sharing systems and the development of the ARPANET (a precursor to the internet) also began during this era.
- **1970s:** The birth of personal computing occurred in the 1970s with the introduction of the Altair 8800 and the development of the microprocessor by Intel. This decade also marked the emergence of the Unix operating system and the creation of C programming language by Dennis Ritchie.

- **1980s:** Graphical user interfaces (GUIs) became popular with the release of the Apple Macintosh and Microsoft Windows. The advent of the World Wide Web in 1989 by Tim Berners-Lee marked a significant milestone, bringing the internet to the public.
- **1990s:** The 1990s saw the explosive growth of the internet, e-commerce, and the development of web technologies such as JavaScript, HTML, and CSS. The rise of search engines like Google and the spread of email and online communication transformed society.
- **2000s and Beyond:** The 21st century has witnessed the proliferation of smartphones and the growth of social media. Cloud computing, artificial intelligence, machine learning, and quantum computing have become prominent areas of study and innovation. Advances in cybersecurity and data science have also played a crucial role in shaping the modern world.

Today, computer science continues to evolve at a rapid pace, influencing nearly every aspect of our lives, from communication and entertainment to healthcare and transportation. It's a field that continues to push the boundaries of what's possible and holds great promise for the future.

2. Introduction to Computer Science

2.1. Computer Science

Computer science is the science of processing information in scientific, technical, economic, and social domains. Data is the representation of information in a conventional form (encoded), designed to facilitate its processing. It is possible to break down the transformation of this data into results, into a sequence of elementary operations, each of which can be executed by a machine (a computer).

2.2. Computer

A machine that captures (using input devices), stores (in memory), processes (using programs), and outputs (through output devices) information.

The computer is divided into two main components: the central unit and peripherals.

The central unit: is the part of the computer that contains the basic circuits. It selects and executes the instructions of the current program. It is composed of the following elements:

1. Main memory
    a. Random Access Memory (RAM)
    b. Read-Only Memory (ROM)
    c. Secondary memory (hard drive, CD-ROM)
2. Processor

a. Arithmetic and Logic Unit (ALU)
b. Control and Command Unit (CCU)

3. System clock
4. Input-output unit
5. Buses

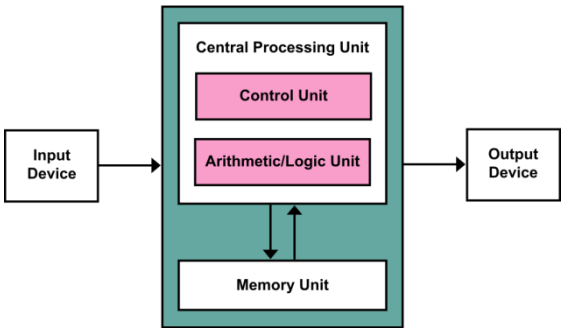


Figure 1.1 Von Neumann

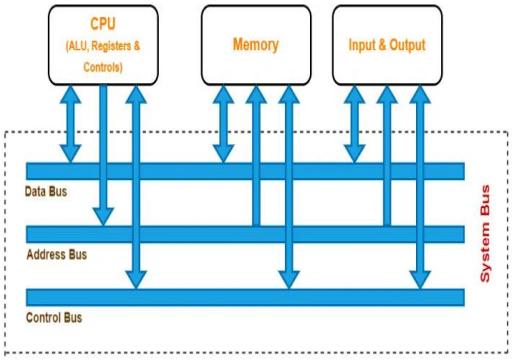


Figure 1.2 Bus System in computer architecture

1. Main memory: is an electronic device capable of memorizing voltages. It can record and restore information. The elementary storage unit is BIT (BinaryDigiT). The information is coded in binaries composed of 0s and 1s. Depending on the computer, a memory word is composed of 2bytes (16 bits) or 4 bytes (32 bits).

- **Units of measure**

  1 byte = 8 bits

  1 KB (kilobyte) ≈1000 bytes (exactly 2^{10} bytes)

  1 MB (mega byte) ≈1,000,000 bytes (2^{20} bytes)

  1 GB (giga byte) ≈ 1,000,000,000 bytes (2^{30} bytes)

  1 TB (tera byte) ≈ 1,000,000,000,000 bytes (2^{40} bytes)

- **Structure**

  Memory is organized into cells (bytes or words)

  Each cell is identified by its address which allows the computer to find the information it needs

- **Features**
  - Capacity: number of bytes
  - Access:
    - Direct: thanks to the address, immediate access to information (we speak of addressable media)

  - Sequential: to access information, you must have read all the previous information (e.g. audio cassette)
- Access time: time elapsed between the moment the information is requested and the moment it is available (in ms)

- The contents of the memory are composed of data (data(s) or data address) and instructions (elementary operation code)

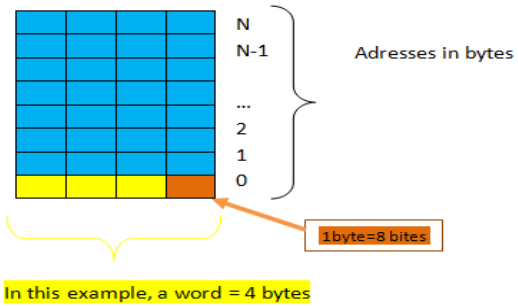


Figure 1.3 Principle memory

1. a. RAM (Random Access Memory)

- Limited size, direct access memory
- Its content is volatile
- Place where the computer temporarily stores the data and instructions (programs) that it is currently using and executing.

1. b. ROM (Read Only Memory)

- Permanent and unalterable memory (unchangeable)
- Contains small programs written by the manufacturer for starting the computer BIOS (Basic Input/Output System)

2. **The processor:** is the brain of the computer, it is which organizes the exchange of data between the different components and which does the calculations. Its power is expressed in Hz. Some computers are equipped with several processors.
It is divided into two parts:

- The command or control unit (UCC): responsible for reading from memory and decoding instructions.
- The processing unit, also called Arithmetic and Logic Unit (UAL), executes the instructions which manipulate the data.

3. **The system clock:**
   - It controls and synchronizes the processor and associated components
   - Its speed (frequency) is generally expressed in megahertz (MHz), that is to say in million cycles per second
   - The efficiency of the processor is directly proportional to the clock frequency: a high frequency is therefore desirable. Examples: Intel Pentium 4, approximately 3 GHz.

4. The input-output unit: it controls and manages the transfer of information between the processor and the peripherals. Examples: graphics card (screen), controller card (hard drive), sound card (microphone, speaker).

5. Buses: A bus is simply a set of n-conductive wires, used to carry n binary signals. Their characteristics are:

- Bus width: number of bits transmitted simultaneously
- Frequency (expressed in Hertz): the number of data packets sent or received for example (122 MB/s).

The types of buses are:

- The address bus (memory bus) carries the memory addresses that the processor wants to access to read or write data. This is a one-way bus.
- The data bus carries instructions to or from the processor. This is a two-way bus.
- The control bus (command bus) carries orders and synchronization signals from the control unit to all hardware components.

3. Running a program

3.1. Example: To calculate 12+5, we need a series of instructions

- Transfer:
  - the number 12 entered on the keyboard into the memory
  - the number 5 entered on the keyboard into the memory
  - the number 12 from memory to a microprocessor register
  - the number 5 from memory to a microprocessor register
- ask the calculation unit to do the addition
- Transfer:
  - the content of the result (17) in memory
  - the result in memory to the console screen (for display)

3.2 Execution of the previous program:

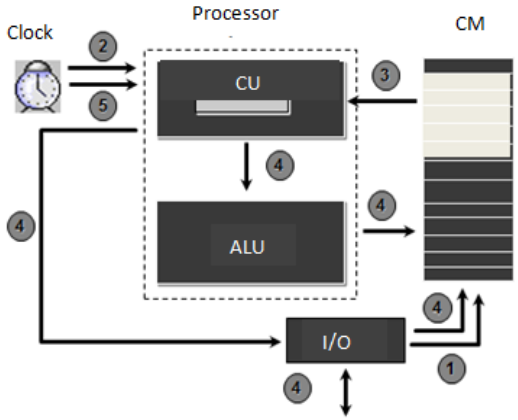


Figure 1.4 Execution of a program

Chapter 2 Simple sequential algorithm

1. Concept of language and algorithmic language

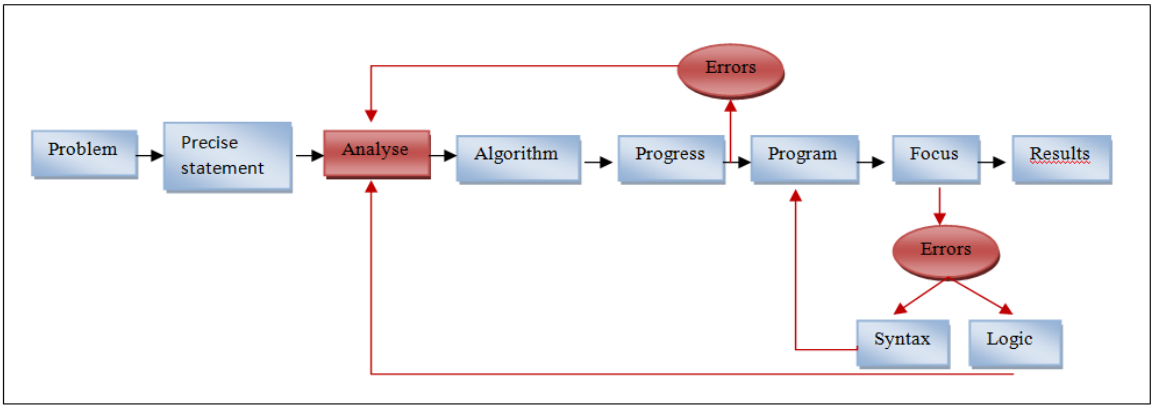


Figure 2.1 From problem to result

As soon as **a problem** is analyzed the designer must express his solution in a **universal formalism**. The problem is therefore to use a common language, in order to understand the solutions constructed by others and vice versa. Hence, the interest in a common, precise and unambiguous formalism. An **algorithmic formalism** is a set of conventions (or rules) in which any solution to a given problem is expressed.

1.1. Algorithm

A sequence of primitive (elementary) actions, executed by a machine (a computer), in a given order and in a finite time, with the aim of performing a very precise work.

The set of objects (elements) necessary for carrying out a work described by an algorithm is called an **environment.**

An action is a step in the algorithm. It's a finite event that changes the environment.

Properties of an algorithm

- It must take into account all possible cases. It treats the **general** case and the **particular** cases
- It always contains a **finite** number of actions.
- It is usually **repetitive** (it contains a repetitive treatment)
- It is **independent** of programming languages and Computer Hardware

1.2. Program

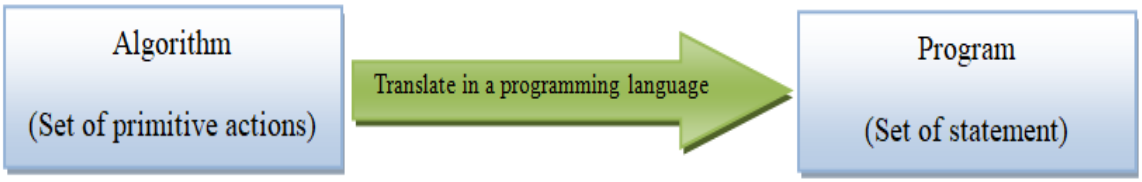


Figure 2.2 form algorithm to program

A program is sequence of instructions written in a programming language translating an algorithm. Each of these statements specifies the operation that the computer must execute.

The program is a simple text file (written with a text editor), called a **source file**. The source file contains lines of programs called source code. This source file when completed must be compiled. The compilation takes place in two steps:

- **The compiler** transforms the source code into object code, and saves it in an object file that has the extension .o (it writes in machine language).
- The compiler then uses a **linker** or binder which makes it possible to integrate into the final file all the auxiliary elements (functions or libraries) to which the program refers but which are not stored in the source file [3].

Then it creates an executable file that contains everything it needs to function autonomously, the file thus created has the extension .exe

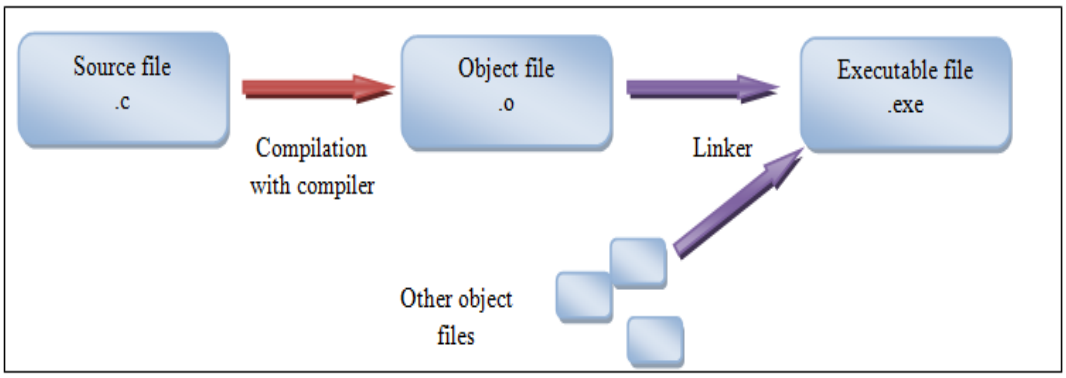


Figure 2.3 form Source file to executable file

2. Parts of Algorithm

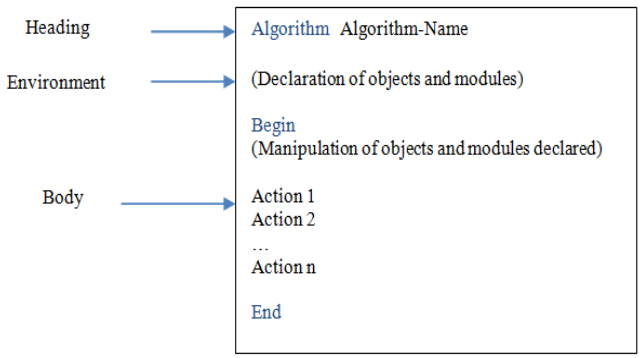


Figure 2.4 Parts of Algorithm

Actions are executed in the order in which they appear in the algorithm (sequence is important).

3. Data: variables and constants

Any algorithm uses objects that will be declared in its environment. An object can be: a label, a variable, a constant, a type, a subprogram.

Each object has:

- A NAME that will identify it and distinguish it from other elements,
- A TYPE which indicates the nature of the set in which the object takes its values
- A VALUE assigned to this object at a given time.

In this course, we'll cover only the elements: variable and constant. These two object s are basic (inseparable) and can contain data of an algorithm [1].

3.1. Variable declaration

A variable is an object that has a name and a type, it can contain a value of the same type. This value can be changed at any time in the algorithm.

Syntax Variable variable_name: type

Example Variable 1, m, n, o, p: real
x, y, z: integer
Rep: Boolean

In C language: this language allows you to define variables before they are used at any time (before the main function or after).

int x; / declaration of an integer variable

float y; / declaration of a real type variable

char Z; / declaration a character variable

3.2. Constants declaration

A constant is an object that its value is invariable during the algorithm execution.

Syntax Constant Constant_Name = Value

Example constant Pi = 3.14
one-hundred = 100
comma = ' , '

In C language: the "# define" directive allows to define constants, it is part of the preprocessor instructions, it must be declared in the heading part.

define PI 3.14 /declaration of a constant

4. Data types

- **The integer type:** This is the set of relative integers Z, however it should be pointed out that if in mathematics this set is infinite, in a computer the values are limited by the length of the machine words.
- **The real type:** This is the set of numbers having a fractional part. This set is also limited, There are two representations of the real:
  - The usual notation with the dot as a decimal symbol.

    Examples: 0.2467 345,876 -12.1 ....
  - The scientific notation, in the form of: aE+b, where:
    - a is the mantissa, which is written in a usual form,
    - b is the exponent representing a positive integer.

    Example: 247 = 2.47E+2 = 0.247E+3 = 2470E-1 = ...

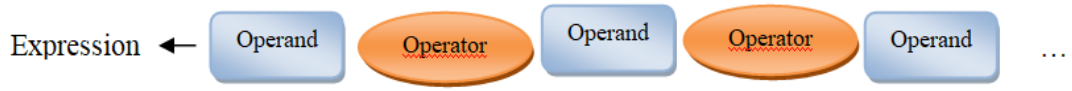
    582.34 = 58234 E-2
- **The Boolean (or logical) type:** This is the set of values {True, False}. They are involved in the assessment of a condition.
- **Character type:** includes all alphanumeric characters (alphabetic and numeric, special signs, and white character (or space)) such as: numbers from '0' to '9', letters from 'A' to 'Z' (uppercase or lowercase), and special characters ('+' '-' ',' ';' '.' '(' '{' '[' ']' '}' ')' '$' '%'...). A character will always be noted between single quotes. The white character (space) is written ' ' the apostrophe ''.

  Lowercase and uppercase characters are treated as different characters.

5. Basic Operations

Expressions

The expressions have the form:



5.1. Arithmetic expressions:

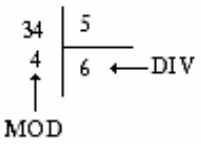
- Operands: integers, real
- Operators: +, -, /, *, DIV, MOD, ^ .
  /: Slash,
  ***:** asterisk

DIV: Integer Division. Division with truncation of the decimal part.

Example:

X ← 34 DIV 5

Consists in placing 6 in X, in fact the division of 34 by 5 gives 6.8 and only the integer part of this result will be taken.



MOD: modulo. Give the rest of a division.

Example:

Y ← 34 MOD 5

Y will contain 4 because the remainder of the division of 34 by 5 gives 6 and a remainder equal to 4.

Operator priority:

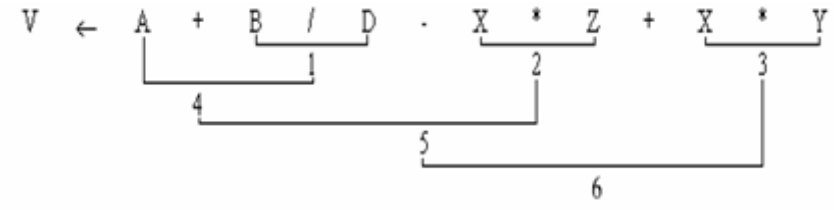
1. unary operator (-)

2. ( ) parentheses, starting with the innermost parentheses.
3. ^ power
4. / multiplication or division
5. + - addition or subtraction

The operator hierarchy lets us know how an expression will be evaluated.

When the hierarchy (the priority) is the same, the expression is evaluated from left to right.

Example



Let the following algebraic expression:

$$\frac{(3-xy)^2-4ac}{2x-z}$$

Its algorithmic form is as follows:

$$((3-y*x)\hat{}2-4*a*c)/(2*x-z)$$

The following table shows the type of arithmetic expression that should be used for the result of an arithmetic expression based on the operators and types of the operands. It must be respected and there is a risk of errors in the execution of programs [4].

Operators	Types of operandes	Type of result
+ - *	E op E R op R R op E E op R	E (integer) R (real) R
/	E op E R op R R op E E op R	R R R
DIV MOD	E op E	E

Figure 2.5 Type of arithmetic expression based on operand and operators

5.2. Logical expressions

The logical expression are combinations between variables and constants using relational operators (=, <, =<, >, >=,< >) and/or combinations between variables and logical constants using logical operators (not, and, or).

In the same way, parentheses and priorities between the different operators are used t o solve the problems of conflicts.

- Logical Operands: True, False
- Logical Operators: AND (intersection), OR (union), NOT (unary operator).

Logical Operator Priority:

1. NO
2. AND
3. OR

A	Not A
True	False
False	True

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

- Relational operands: Digital , Alphanumeric
- Relational operators: =, >, >=, <, <=, < >

These operators have the same hierarchy.

Relational operators represent an order between characters. This order is determined by ASCII coding, as flow:

‘ ‘ < ‘0’ < ‘1’ < ... < ‘9’ < ‘A’ < ‘B’ < ... < ‘Z’ < ‘a’ < ‘b’ < ... < ‘z’

Example: a <> b

B <= x

5.3. Mixed expressions

Mixed expressions can be found between arithmetic and logic.

Priority of all operators:

1. NOT
2. * , / , DIV , MOD , AND
3. + , - , OR
4. = , > , >= , < , <= , <>

Exemple :

(a * b > c) OU (a DIV c = 0) AND (x OR NOT y) AND (b MOD c >= r)

6. Basic statements

6.1. Assignment

Assignment is symbolized by ←. The assignment role is to assign (give, assign) a value to an object. The value can be a constant, the value of another object, or the result of evaluating an expression.

Syntax: Variable <- expression

Examples

X← 0 *Set zero in X.*

X← Y *Set the value of the Y object in X.*

X← X + 1 *Added 1 to X*

X← X - 5 + Z *Place the result of the evaluation of the expression X - Y + Z in X*

Functioning: In the assignment, it will therefore be necessary to evaluate, possibly, the entity located to the right of the assignment symbol and then to put this result in the entity located to the right of the assignment symbol.

In C language: X = 0 ;

S = X +45 ;

6.2. Input/output statement

a. Reading

It makes it possible to supply values coming from the outside to variables of the algorithm, because it often happens that an object does not vary during the execution of the algorithm, however the user changes its value between 2 executions of this same algorithm.

This object is called a parameter, and using parameters generalizes algorithms.

Syntax: READ (x) , READ (x, y,...)

Functioning: The values read from the keyboard are respectively assigned to the variables with type compatibility taken into account.

Examples:

READ (N)

READ (a, b, c)

In C language: scanf ("%d", & X); / such that X is an integer

b. Writing

It displays something to the user as the results of an algorithm or a sentence.

Syntax: WRITE (x), WRITE (x, y...)

When X can be:

- A variable
- A label: string between apostrophes
- An expression

Operating: Expressions are evaluated and results are written or displayed

In C language: printf ("negative");

printf (" %f ", X); / such as X is a real

7. Building a Simple Algorithm

Write an algorithm that asks for two integers from the user and then displays their sum.

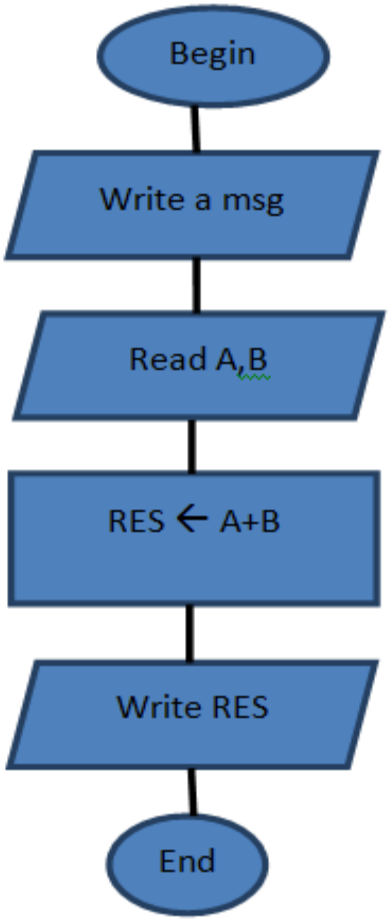
Algorithm Sum
Variable A, B, RES: integer
Begin
Write (“Please enter two numbers please:”)
Read (A,B)
RES← A+B
Write (” the sum of the two numbers” , A ,” and” , B,” is ‘’ , RES)
End

In C language :

```
#include <stdio.h>
int main() {
   int A, B, RES;
   printf("Please enter two numbers: ");
   scanf("%d %d", &A, &B);
   RES = A + B;
   printf("The sum of the two numbers %d and %d is %d\n", A, B, RES);
   return 0;
}
```

8. Sequential algorithm Representation with a flowchart

Flowchart: is a schematic representation of an algorithm. The ellipse is used to mark the begin and the end of a flowchart. The parallelogram is used to represent input and output statements. The rectangle is used to represent assignment operations. The diamond is used for representing conditional statement in an alternative process or in a repetitive process [5].


Begin
Write a msg
Read A,B
RES ← A+B
Write RES
End

Chapter 3 Conditional Structures

5. Introduction

We have seen the sequencing of actions in a sequential algorithm; these primitive actions are executed in the order in which they appear in the algorithm. However, sometimes an action must be performed only when a condition is satisfied, or two actions must be performed alternately depending on whether a condition is satisfied. In order to meet this need, conditional structures come in several types that we will see in this section.

6. Simple conditional structure

Syntax

IF Condition THEN
Instruction
END IF

Such as:

“Condition”: is a logical expression that can be verified (value after evaluation = true) or unverified (value = false)

The instruction part is called the “if block”, because we can find several instructions in this part, not just one instruction.

Functioning

In the case where the condition is verified then the instruction will be executed, then the execution continues after FINSI. Otherwise the instruction does not execute, execution continues after FINSI.

Example

ALGORITHM Positive
Variable A: integer
Begin
Write ("Give a number ")
Read(A)

```
IF (A >0) THEN
Write(A, "is positive")
END IF
END.
```

Translation to C language

```
# include <stdio.h>
int main ( ) {
int a;
printf ("anter a integer number please \n" ) ;
scanf ("%d", &a) ;
if (a>0) printf ("positif");
return 0;
}
```

Simple condition representation with a flowchart

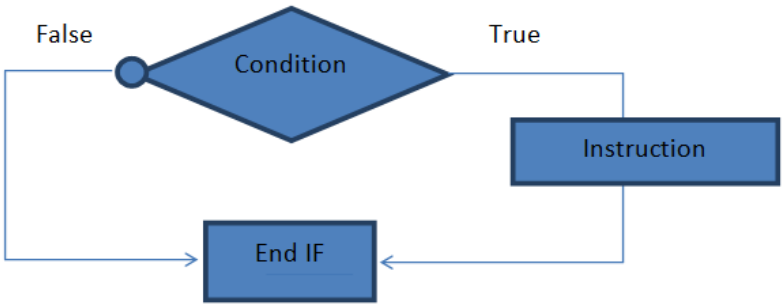


Figure 3.1 simple condition flowchart

7. Compound conditional structure (alternative structure)

Syntax

```
IF  Condition   THEN
Instruction 1
ELSE
Instruction2
End IF
```

Such as:
“Condition”: is a logical expression that can be verified (value after evaluation = true) or unverified (value = false)

The part “instruction 1” is called the” if block”, the part “instruction 2” is called the “else block ", because it can be several instructions in this part and not just one instruction.

Functioning

In the case where the condition is satisfied then instruction 1 will be executed, and then the execution continues after End If. Otherwise, instruction 2 will be executed, and then the execution will continue after End If.

Example

```
ALGORITHM positive-negative
Variable A: integer
begin
Write ("Give a number ")
Read(A)
IF (A > = 0) THEN
Write(A, "is positive")
Else
Write(A, "is negative")
End IF
END
```

Translation to C language

```
# include <stdio.h>
int main ( ) {
int a;
printf ("Give an integer number please \n" ) ;
scanf ("%d", &a) ;
if (a>=0) printf ("positif");
else printf ("négatif");
return 0;}
```

Note

The “if” block or the “Else” block may contain, among the instructions, simple or compound conditional structures, this is called “nested if”. As in the following example:

```
ALGORITHM  Pos-Neg-Null
Variable A: integer
Begin
Write("Give a number ")
Read(A)
IF (A >0) THEN
Write (A, " is positive")
Else {A <= 0}
  IF (A < 0)THEN
  Write(A, " is negative")
  Else {A = 0}
  Write (A, "null")
  End IF
End IF
End
```

Translation to C language

```
# include <stdio.h>
int main ( ) {
int a;
printf ("Give a number please \n" ) ;
scanf ("%d", &a) ;
if (a>0) then
 printf ("positif");
else
    if (a<0) then
     printf ("négatif");
     else
    printf ("nulle");
return 0; }
```

Compound condition representation with a flowchart

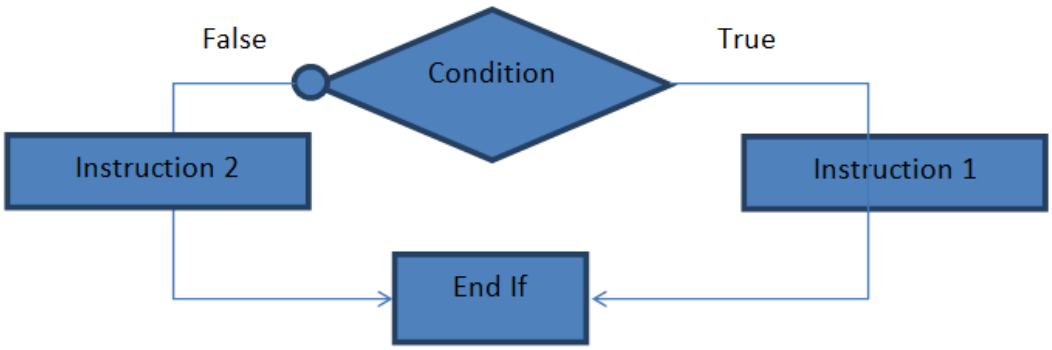


Figure 3.2 Compound condition flowchart

8. Multiple Choice Conditional Structure (Selective structure)

Syntax

Case (Variable) of

Value 1: Instruction 1

Value 2: Instruction 2

.....

Value n: Instruction n

Else: instrcution n+1

END

Such as:

- The variable can also be an arithmetic or logical expression.
- Value: Can be a single value, sequence of values separated with commas, or an interval.
- Values must be the same type as the variable.
- Instruction: may be a single instruction or a sequence of instructions (a block).

Functioning

The “Case” structure compares the value of the variable with values from 1 to n. If a value is matched, the corresponding instruction will be executed, and then the execution continues after End Case. If no value is matched, the n+1 instruction will be executed, and then the execution continues after End Case.

Example

```
ALGORITHM Observation
Variable mark: integer
Begin
Write ("Give your mark ")
Read (mark)
Case mark of
20: Write ("Excellent")
19..16: Write (" Very good")
15..12: Write ("Good")
11.10: Write ("Tolerable")
9..0: Write ("Insufficient")
Else
Write (" This is not a mark")
End Else
END.
```

Traduction de l'exemple en C++ :

```
# include <stdio.h>
int main ( ) {
int note;
printf ("entrer une note de type entier svp \n" ) ;
scanf ("%d", &note) ;
switch (note)
{
case 20: printf (" excellent"); printf (" meilleure note"); break;
case 16 .. 19: printf ("Very good \n");  break;
case 12: printf (" good"); break;
case 10:  case 11: printf ("tolerable \n"); break;
case 0 .. 9: printf ("Insufficient \n"); break;
default: printf ("This is not a mark \n");
}
return0;}
```

Chapter 4 Loops (repetitive structures)

1. Introduction

When one or more actions (instructions) are repeated, these are written in the same compound action and the execution is repeated several times. The number of iterations may or may not be known a priori. In the latter case, the execution of the iterative action (the loop) will determine its stopping. There are several ways (three) to express a loop.

2. The loop “While”:

Syntax

While Condition Do

    Instruction

End while

Such as:

“Condition”: is a logical expression that can be verified (value after evaluation = true) or unverified (value = false)

The “instruction” part can be one or more simple, compound, selective or iterative actions.

Functioning

As long as the condition is verified, the body of the loop will be executed, and we stop as soon as the condition is no longer verified.

Syntax in C language

While (condition) {.... }

Example 1: Perfect Number

```
# include <stdio.h>
int main (){
int x;
printf("veuillez enter un nombre SVP!\n");
```

```
scanf("%d",&x);
int i=1;
int s=0;
while (i<x) {
if (x % i == 0) s+= i;
i++;
}
if (s==x)
printf("le nombre est parfait\n");
else
printf("le nombre est nom parfait\n");
return 0;
}
```

Example 2

Write an algorithm that allows the user to enter a sequence of characters ending with '*', and displays at the end the number of times the letter 'B' appears.

```
Algorithm Appearance
Variable letter: character
        NbB: integer;
Begin
NbB ←0 ;
letter ←'X'; /* Initialize letter to a character other than '*'
While letter < > '*' Do
   Read(letter);  /* read before processing
   If  letter = = 'B' Then
       NbB ←NbB+1
   End if
End while
Write ('Number of B occurrences is:', NbB);
End.
```

3. The loop "repeat"

Syntax

Repeat

Instruction

Until Condition

Such as:

"Condition": is a logical expression that can be verified (value after evaluation = true) or unverified (value = false)

The "instruction" part can be one or more simple, compound, selective or iterative actions.

Functioning

The Repeat loop allows entering the loop regardless of the condition and repeats the execution until the condition is verified.

Syntax in C language:

Do {.... }

While (condition)

Example

Algorithm Appearance

Variable letter: character

NbB: integer

Begin

NbB ←0 ;

Repeat

Read (letter)

If letter = 'B' Then

NbB ←NbB+1

End if

Until letter =='*'

Write ('Number of B occurrences is:' NbB);

End.

Notes

- Whatever the state of the condition, in the "repeat" loop, the body of the loop will be executed at least once. Whereas in the loop "While", if the condition is not initially satisfied, the loop will not be executed.
- In these two loops, the number of iterations is not known a priori. It depends on the condition.
- In the body of the loop, there must exist a variable - called a control variable - or counter which will be modified to change the state of the condition. In general, this control variable (counter) must be initialized before entering the loop.
- The choice of the loop and how to use it must respond to the problem to be solved.

4. The loop "for"

Syntax

For i ← initial_val to final_val, step_Val DO

Instruction

End For

Such as:

The "instruction" part can be one or more simple, compound, selective or iterative actions.

Functioning

In this loop, the number of iterations is known, a counter (here i) is used which will be automatically initialized at the initial value and will be incremented by the step value, until to the final value.

Syntax in C language:

```
Int i;

For (i=0 ; i< 10; i++)   {…}
```

Note

In the body of the loop, it is forbidden to modify the value of the counter (the I variable), but it can be used (display it or use it in an expression,...).

Example

Algorithm that displays the multiplication table by 7:

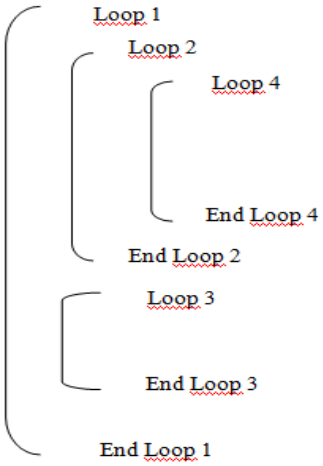
```
Algorithm multiplication
Variable i: integer
Begin
```

```
For i ← 1 to 10 do
  Write ( i, ' * 7 = ', i* 7)
End for
End.
```

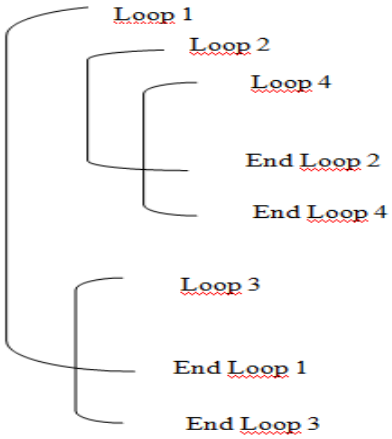
5. Nested loops [3]

In the body of a loop, there may be one or more other loops. Imbricate loop should follow these principles:

Authorized imbrications



unauthorized imbrications



Example

Algorithm that displays all multiplication tables by 1 up to 10

```
Algorithm multiplication
Variable i,j: integer
Begin
  For i  ←  1 to 10 do
    For j  ←  1 to 10 do
       Write ( i, '*', j, ' =', i* j)
    End for
  End for
End
```

Chapter 5 Arrays and Strings

1. Introduction

So far, we have only manipulated simple variables. In order to perform more complex treatments with maximum efficacy, more advanced data types such as tables are required.

2. The type “array”

An array is a set of data that are all of the same type, have a unique identifier (the name of the array) and are differentiated from each other, in this array, by their index number.

Syntax

<Array-Name> [<minimum-index> ..< maximum-index>]: Array of <datatype>

Such as: The index type must be an integer. The data type can be any simple or compound type.

Example

Marks [1..10]: array of real

Here we declare an array called Marks which can contain 10 real variables with index 1,2,.. up to 10.

Syntax in C language

float Marks [10];

An array called Marks has been declared here containing 10 real variables of index 0, 1, 2, ... up to 9.

Array operations

- **Filling**

Algorithm Filling
Variable i: integer

```
Tab [1..10] : array  of real
Begin
For i ← 1 to 10 do
  Write (“Give Item N° ”, i,“:”)
   Read (Tab[i])
End for
End
```

- **Display**

```
Algorithm Display
Variable i: integer
Tab [1..10] array of real
Begin
For i ← 1 to 10 do
  Write (“Give Item N° ”, i,“:”)
   Read (Tab[i])
End for
For i ← 1 to 10 do
 Write (“Item N° ”, i, “:”, Tab[i])
End for
End
```

- **Finding the smallest item in an array**

```
#include <stdio.h>
#include <stdlib.h>
#define N 10

int i, vmin;
                    /* Filling the array */
   t[N] = { 3, 2, 5, 10, 8, 6, 1, 7, 4, 9 };

main() {
   vmin = INT_MAX;

   for (i = 0; i < N; i++) {
      if (t[i] < vmin)
         vmin = t[i];
   printf("the minimun value is : %d\n", vmin);
}
```

- The sum of the numbers in an array
- Average of the numbers in an array
- Sum of two tables of the same size in a third
- Sorting (the principle of ascending or descending sorting and applying sorting by sel ection)
- …

3. Multidimensional arrays (Matrix)

A two dimensional array or "a matrix" is an array that has a unique identifier (a name) and is composed of multiple rows and columns. The data in the table is all of the same data type. Each element of the table is identified by the name of the table followed by two indices: index of the row and that of the column.

Syntax:

<Array-Name> [<minimum-row-index> ..< maximum-row-index>, <minimum-column-index> ..< maximum-column-index>]: Array of <data-type>

Such as: The index must be of the integer type. The data type can be any simple or c ompound type.

Example

Marks [1..5, 1..10]: array of Real

A two dimensional table (a matrix) called Marke is declared here, composed of 5 rows and 10 columns, each box can contain a real type variable.

Syntax in C language

```
float  Marks [5] [10] ;
```

4. Strings

A string is an array of characters. Each element is of character type and accessible in the string by its index (rank).

Syntax

<Array-Name> [<index-minimum> ..< index-maximum>]: String

Example

Sentence [1..20]: String

Syntax in C langage

Char Sentence [20];

- ## String operations in C language

1. Declaration

Char Word [10];

Word is a string of maximum length = 10

The string manipulation functions are contained in the heading file < string.h>, which must be included at the beginning of the program.

2. Initialization

Char word [ ] = " hello";

The compiler determines the required size

3. Reading a string

scanf ("% s", word); or:

gets (word);

Such as word is a string.

4. Writing a string

printf (" % s \n", word); or:

puts (word);

Such a word is a string.

5. The length of a string

x = strlen (word),

x contains the number of bytes occupied by the string: word

6. Copy a character string

strcpy (word1, word 2);

Word1 receives word2 content

Such as word1 and word 2 are strings

7. Lowercase to uppercase conversion

word1 = strupr (word 2);

word 1 receives the contents of word 2 and converts it to capital letters.

Such as word1 and word 2 are strings

8. String comparison

x = strcmp (word1, word 2);

x receives: 0 if the two strings are equal, a negative number if word1 precedes in alphabetical order, and a positive value if word1 follows word2.

9. String concatenation

word = strcat (word 1,word2);

This function concatenates the word2 at the end of the word1 in the string "word".

Chapter 6 Custom Types

1. Introduction

Up to this point, we have only used simple or standard types. However, there are times when we need to define our own type that precisely meets specific characteristics or adapts to a given situation. In this chapter, we will learn about custom types, using classic simple types.

2. Enumerations

2.1. Definition

Enumeration (Enum) is a custom data type that we allows to define a set of named values (constants) that represent possible states or categories of a variable.

2.2. Syntax

We can declare new types as follow:

Type Type_Name = description of objects handled in this type

2.3. Numeration types

There are two types of constructed types:

2.3.1. Enumerated Type

We define all the values that objects of this type can take. The values must be ordered and finite. Relational operators (<, <=, >, >=, =, ≠) can applyed to the different values of this type.

Example 1

Type Day = (Friday, Saturday, Sunday, Monday, Tuesday, Wednesday, Thursday)

Variable j1, j2: Day

j1 ← Tuesday

j2 ← Sunday

Example 2

Friday < Saturday < Sunday < Monday < Tuesday < Wednesday < Thursday

Example in C language

```
enum Day { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday };
```

```
enum Day Today;
 Today = Monday;
printf("Today is: %d\n", Today);
```

Remakes:

- In C, enumeration values are actually integers. By default, Monday = 0, Tuesday = 1, ...
- You can explicitly set values if you want, e.g. Monday = 1.
- If you want to print names (like "Monday"), you'll need to use an array of strings.

2.3.2. Interval Type

We define all the values that objects of this type can take. The values must be ordered, and only the minimum and maximum will be specified.

Example

Type Number = 0 .. 9

Type Letter = A .. Z

Type Working-Day = Sunday.. Thursday

Remarque

Values belonging to an interval must be of an already defined type. Although enumerated and interval types are rarely used, they nevertheless enhance the readability of algorithms and facilitate checks.

Example in C language

```
typedef enum {
   A = 'A', B, C, D, E, F, G, H, I, J, K, L, M,
   N, O, P, Q, R, S, T, U, V, W, X, Y, Z
} Letter;
int main() {
   Letter l = C;
   printf("Letter: %c\n", l);       // prints C
return 0; }
```

3. Record (Structure) [3]

3.1. Definition

A **structure** is custom **data type** that we allows to group together variables of different types under a single name.
Each variable inside the structure is called a **member** (or field). Structures are useful when we want to represent a real-world entity (like a person, student, or product) that has multiple attributes of different types. A variable of record type is declared as follows:

3.2. Syntax

```
Type Type_Name = Record
  Field_1: TypeElem1
  Field_2: TypeElem2
  ...
  Field_n: TypeElemn
End_record

Variable identifier: Type_Name
```

3.3. Example

```
Type Person = Record
  Name: String
  Surname: String
  Age: Integer
  Address: String
End record

Variable student, P1, P2: Person
```

3.4. Accessing a field of a record

The fields of a variable of record type are accessible using the variable's identifier and the field name.

3.5.Example

```
student.Name
```

student.Address

...

student.Name ← "Ahmed"

Read (student.Name)

Write (student.Name)

3.6. Example in C language

```
#include <stdio.h>
#include <string.h>
typedef struct {
   char Name[50];
   char Surname[50];
   int Age;
   char Address[100];
} Person;

int main() {
      Person student, P1, P2;

   strcpy (student.Name, "Ali");
   strcpy(student.Surname, "Mehdi");
   student.Age = 21;
   strcpy(student.Address, "Algiers, Algeria");

   printf("Name: %s\n", student.Name);
   printf("Surname: %s\n", student.Surname);
   printf("Age: %d\n", student.Age);
   printf("Address: %s\n", student.Address);
   return 0;
}
```

Part 2: Solved exercises

General information about the C language

1. The data types in the C language:

Data type	Meaning	Size (in bytes)	Accepted value range
Char	Character	1	-128 to 127
Unsigned char	Unsigned character	1	0 to 255
Short int	Short integer	2	-32768 to 32767
Unsigned short Int	Unsigned short integer	2	0 to 65535
Int	integer	2	-32768 to 32767
Unsigned int	Unsigned integer	2	0 to 65535
Long int	Long integer	4	-2147483648 to 2147483648
Unsigned long int	Unsigned long integer	2	0 to 4294967295
float	Floating (real)	4	$3.4 *10^{-38}$ to $3.4 *10^{38}$
Double	Double floating	8	$1.7 *10^{-308}$ to $1.7 *10^{308}$
Long double	long Double floating	10	$3.4 *10^{-4932}$ to $3.4 *10^{4932}$

2. Increment and decrement in C language:

x = x +1; or x++;

x = x - 1; or x -- ;

3. Operators in C language:

a. Arithmetic operators

+ Addition operator

- Subtraction operator

***** Multiplication operator

/ Division operator

% Modulo operator (remainder)

b. Arithmetic and assignment operators

+= Adds two values and stores the result in the variable (left)

- = Subtract two values and store the result in the variable

*** =** Multiplies two values and stores the result in the variable

/ = Divides two values and stores the result in the variable

c. Logical operators

|| Represents the logical OR

&& Represents logical AND

! Represents the logical NO

d. Comparison

= = Equality operator, do not confuse it with the assignment sign (=)

< Strict inferiority operator

<= Inferiority operator

> Strict superiority operator

>= Superiority operator

!= Difference operator

4. Format indicators:

Indicator	**description**
% c	Character format
% d	Integer format
% i	Integer format (equivalent to %d)
% f	Decimal format
% e ou %E	Scientific notation format
%g ou %G	Use %f or %e depending on the length of the decimal value
% o	Unsigned octal format

% s	String format
% u	Unsigned integer format
% x	Unsigned hexadecimal format (lowercase letters)
% X	Unsigned hexadecimal format (capital letters)
% p	Displaying the argument pointer
% n	Records the number of characters displayed
% %	Allows you to display the % symbol

5. The main escape characters:

Character	**Description**
\n	Line break
\t	Tab character
\b	Moves the cursor one character to the left (backspace)
\r	Place the cursor at the start of the current line
\f	Place the cursor at the start of the next line (page break)

6. functions (<math.h>)

Fonction	**Signification**
fabs	absolute value
fmod	remainder
ceil	ceiling
floor	floor
modf	whole and fractional parts
cos	cosine
sin	sinus
tan	tangent
acos	arc cosine
asin	arc sine
atan	arc tangent
atan2	tangent arc of two variables with quadrant adjustment
exp	exponential

cosh	hyperbolic cosine
sinh	hyperbolic sine
tanh	hyperbolic tangent
log	natural logarithm
log10	logarithm to base 10
pow	power
sqrt	square root
frexp	decomposition into normalized fraction in base 2
ldexp	produced by a power of 2
HUGE_VAL	value representing capacity overrun

Tutorial 1: Input / Output Statement

Educational objectives: I/O instructions and the difference between Read and Write. Concept of variable. Concept of constant. Arithmetic expression.

Exercise 1
Write an algorithm that displays: hello.

Exercise 2
Write an algorithm that calculates the sum of the two numbers: 18 and 51, then displays the result.

Exercise 3
Write an algorithm that asks the user for a number, then calculates and displays the double of this number (variable notion).

Exercise 4
Write an algorithm that asks the user for a character, and then displays it as a character and as an integer (its corresponding ASCII code).

Exercise 5

Write an algorithm that asks the user for two integer numbers then calculates and displays their division and their integer division.

Exercise 6
Write an algorithm that asks the user for two numbers, and then displays their permutation, using two methods (with and without an intermediate variable).

Exercise 7
Write an algorithm that asks the user for the radius of a circle R, then calculates and displays the surface area SURF of the circle. Such as: SURF = Pi * R^2

Exercise 8
Write an algorithm that reads the price excluding tax of an item, the number of items and the VAT rate, and which provides the corresponding total price including tax.

Exercise 9
Write an algorithm that reads the coordinates of two vectors u and v, and calculate their norm and their scalar product.

Exercise 10
Write an algorithm that reads the time in seconds, and then calculate and display the time in hours, minutes and seconds.

Tutorial 1 Solution

Exercise 1

Algorithm hello
Variable
Begin
Write (“Hello”)
End.

Exercise 2

Algorithm sum
Variable
Begin
Write (“The sum of 18 and 51 est ” , 18+51)
End.

Exercise 3

Algorithm double
Variable x, y : integer
Begin
Write (“Please, give me un number!”)
Read (x)
y ← x* 2
Write (“the double of ”, x, “is”, y)
End.

Exercise 4

Algorithm code
Variable x: character
Begin
Write (“Please, give me un character!”)
Read (x)
Write (“Here is your character ”, x)
Write (“Here is the ASCII code of this character”, (ASCII code) x)
End.

```
#include <stdio.h>
int main() {
   char x;
   // Ask the user for a character
   printf("Enter a character: ");
   scanf("%c", &x);
   // Display the character and its ASCII code
   printf("Character: %c\n", x);
   printf("ASCII Code: %d\n", x);
```

```
   return 0;}
```

Exercice 5

```
Algorithm division
Variable  Number1, Number2, y: integer
          x: real
Begin
Write(“Please, give me two integer numbers! )
Read (Number1, Number2)
  x← Number1 / Number2
  y ← Number1 div  Number2
Write(“Integer Division: ", y)
Write(“Division: ", x)
End
#include <stdio.h>
int main() {
   int num1, num2;
   float division;
   int integer_division;

   // Ask the user for two numbers
   printf("Enter the first number: ");
   scanf("%d", &num1);

   printf("Enter the second number: ");
   scanf("%d", &num2);

      division = (float) num1 / num2;
      integer_division = num1 / num2;

      // Display the results
      printf("Division %f\n", division);
      printf("Integer Division: %d\n", integer_division);
 return 0;}
```

Exercise 6

```
Algorithm permutation
Variable  Number1, Number2, Temp: integer
Begin
Write(“Please, give me two numbers! )
Read (Number1, Number2)
   Temp ← Number1
   Number1 ← Number2
   Number2 ← Temp
 Write(“After permutation: ", Number1, Number2”)
End
```

Method 2: Without an intermediate variable

```
Algorithm permutation
Variable  Number1, Number2: integer
Begin
Write(“Please, give me two numbers! )
Read (Number1, Number2)
   Number1 ← Number1 + Number2
   Number2 ← Number1 - Number2
   Number1 ← Number1 - Number2
Write(“After permutation: ", Number1, Number2”)
End
```

Exercise 7

```
Algorithm Circle
Variable  R, SURF:real
Contant Pi = 3.14
Begin
Write(“Please, enter the radius of the circle”)
Read (R)
SURF ← Pi * R ^ 2
Write(“The surface area of the circle is”, SURF)
End
```

Exercise 8

```
Algorithm price-tax
Variable  PriceExcludingTax, VAT, TotalPrice : real
          Quantity: integer
Begin
Write(“Please, give me the PriceExcludingTax and VAT )
Read (PriceExcludingTax, Quantity, VAT)
Write(“Please, give me the Quantity”)
Read (Quantity)
   TotalPrice ← PriceExcludingTax * Quantity * (1 + VAT / 100)
Write(“Total Price Including Tax: ", TotalPrice)
End
```

Exercise 9

```
Algorithm vectors
Variable   u1, u2, v1, v2, ScalarProduct: integer
           NormU, Normv
Begin
Write(“Please, give the coordinates of the first vector ”)
Read (u1,u2)
Write(“Please, give the coordinates of the second vector ”)
Read (v1,v2)
   NormU ← sqrt(u1^2 + u2^2)
```

```
   NormV ← sqrt(v1^2 + v2^2)
   ScalarProduct ← (u1 * v1) + (u2 * v2)
   Write(“Norm of u: ", NormU)
   Write(“ "Norm of v: ", NormV)
   Write(“"Scalar Product: ", ScalarProduct)
End
```

Exercise 10

```
Algorithm Time
Variable   total_seconds, remaining_seconds, hours, minutes, seconds: integer
Begin
Write(“Please, give the time in secondes ”)
Read (total_seconds)
   hours ← total_seconds div 3600
   remaining_seconds ← total_seconds % 3600
   minutes ← remaining_seconds div 60
   seconds ← remaining_seconds % 60
Write(“Time is: “, hours, “ hours, “, minutes, “ minutes, “, seconds, “seconds”)
End
```

```
#include <stdio.h>
int main() {
  unsigned total_seconds, hours, minutes, seconds;

   // Input time in seconds
   printf("Enter time in seconds: ");
   scanf("%d", &total_seconds);

   // Calculate hours
   hours = total_seconds / 3600;

   // Calculate remaining minutes
   minutes = (total_seconds % 3600) / 60;

   // Calculate remaining seconds
   seconds = total_seconds % 60;

   printf("Time is: %d hours, %d minutes, %d seconds\n", hours, minutes, seconds);

   return 0;}
```

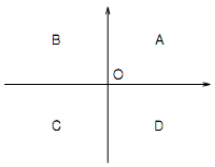
Tutorial 2: Conditional Statement

Educational objectives: Arithmetic expression. Difference between the programmer role and of the user role. Simple conditional structure. Alternative conditional structure. Multiple choice conditional structure.

Exercise 1
Write an algorithm that asks the user for two numbers and then informs him if their product is negative, positive or zero (without calculating the product of the two numbers).

Exercise 2
A bookstore charges 2 DA for the first ten photocopies, 1.50 DA for the next twenty and 1DA beyond that. Write an algorithm that asks the user for the number of photocopies made and then displays the corresponding amount.

Exercise 3 Coordinates
Write an algorithm that, given the x and y coordinates of a point, determines in which part (A, B, C, or D) of the plane the point is located:



Exercise 4
Write an algorithm that gives the flight duration in hours: minutes, knowing the departure time and arrival time. Departure and arrival are considered to take place on the same day.

Exercise 5
Write an algorithm that allows you to solve the second degree equation.

Exercise 6
Write an algorithm that allows displaying the season by entering the month number.

Exercise 7
Write an algorithm that asks the user for a month number and returns his name and the number of days in that month.

Tutorial 2 Solution

Exercise 1: Object: master component condition and nested blocks.

```
Algorithm Product
Variable a, b: real
Begin
Write('give two numbers please)
Read(a,b)
If (a>0 and b>0) or (a<0 and b<0)  then
Write('positive product')
Else
     If (a>0 and b<0) or (a<0 and b>0) then
     Write('negative product')
     Else
     Write('null product')
    End if
End iF
End.
```

Exercise 2:

```
Algorithm amount
Constante  P1=2 ; P2=1.5 ; P3=1
Variable  Mont :real
           Nbc :enteger
Begin
Write ('give the photocopies number please') ;
Read (Nbc) ;
        if Nbc≤10 then
        Mont←P1*Nbc
        else
           if Nbc≤30 then
              Mont←P1*10+P2*(Nbc-10)
           else
              Mont←P1*10+P2*20+P3*(Nbc-30)
           end if
        end if
Write ('the amount to pay is: ',Mont)
End.
```

Exercise 3: to make work easier we can use separate blocks, but that is not the best solution.

```
Algorithm Plan
Variable x, x: real
Begin
Write('give the coordinate x and y')
Read(x,y)
```

If (x>0 and y>0) **then**
Write(‘the point is in A’)
End if
If (x<0 and y<0) **then**
Write(‘the point is in C’)
End if
If (x>0 and y<0) **then**
Write(‘the point is in D’)
End if
If (x<0 and y>0) **then**
Write(‘the point is in B’)
End if
End.

Exercise 4: we can use in a block Begin and End

ALGORITHM flight-duration

VARIABLE h1, h2, m1, m2, hd, md: ENTIER

Begin
Write (" entrer horaire de départ: h m")
Read (h1, m1)
Write (" entrer horaire d’arrivée: h m")
Read (h2, m2)
if (m2 > m1) **then**
begin
hd ← h2-h1
md ← m2-m1
Write (" the flight duration is : ", hd , ' : ', md)
end
else
begin
hd ← h2-h1-1
md ← m2+60-m1
Write ("the flight duration is: ", hd , ' : ', md)
end
end

Exercise 5

ALGORITHM second-degree
VARIABLE a, b, c, delta, x,x1,x2 : ReaL
begin
Write ("enter the values a, b and c of the equation $ax^2+bx+c=0$: ")

```
Read (a, b, c)
        if (a=0 ) then
        Write ("first degree equation")
                if (b≠0 ) then
                    Write ("solution is ", -c/b)
                else if (c=0)Write (" infinite number of solutions")
                     else Write("no solution")
                     end if
                end if
         else delta ← b*b-4*a*c
                if (delta > 0) then
                begin
                x1 ← (-b-sqrt(delta))/2a
                x2 ← (-b+sqrt(delta))/2a
                Write ("The solutions are " , x1, x2 )
                end
                else
                        if delta =0 then
                        Begin
                         x ← -b/(2a)
                         Write ( "Solution is", x)
                        end
                         else Write ("no real solutions!!!!")
                        endif
                end if
        end if
end.
```

Exercise 7:

```
ALGORITHM Month
Variable m: integer
Begin
Write ("Give a number please ")
Read (m)
Case ( m) of
1: Write ("January, 31")
2: Write (" February, 28 or 29")
3: Write ("March, 31")
4: Write ("April, 30")
5: Write ("May, 31")
6: Write ("June, 30")
7: Write ("July, 31")
```

```
8: Write (" August, 31")
9: Write ("September, 30")
10: Write ("October, 31")
11: Write ("November, 30")
12: Write ("December, 31")
Else
Write (" This is not a month number")
End Else
END.
```

Tutorial 3: Loops

Educational objectives: Master the loops: while …do, repeat...Until and for...do. Nested loops, Conditions.

Exercise 1
Write an algorithm that calculates the sum of the first N positive numbers.
Give three distinct solutions using the loops: FOR, WHILE and REPEAT.

Exercise 2
Write an algorithm to calculate the factorial of a strictly positive integer n knowing that:

n ! = 1*2*3*…….*n
0 ! = 1

Exercise 3
Write a program that generates a random number between 1 and 20. Prompt the user to guess the number. Continue prompting until the user correctly guesses the number.

Exercise 4
A perfect number is a number that is equal to the sum of all its divisors except itself. Construct the algorithm for finding all perfect numbers between 1 and 1000.

Exercise 5
Build the algorithm for converting an integer to binary.

Example: 29 in base 10 gives 11101 in base 2.

Exercise 6
Consider a given integer NB. Write an algorithm that searches for the smallest divisor of NB different from 1 and the largest divisor of NB different from itself.

Example 1: NB = 7 Result: NB has no divisor
Example 2: NB = 4 Result: NB has only one divisor: 2
Example 3: NB = 8 Result: smallest divisor: 2 largest divisor: 4

Exercise 7
Construct the algorithm that calculates the N^{th} (with N>2) term of the Fibonacci sequence which is defined by:

$$\begin{cases} U_0 = U_1 = 1 \\ U_n = U_{n-1} + U_{n-2} \end{cases}$$

Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, …

Exercise 8
Write a program to print the first n rows of Pascal's Triangle.

Exercise 9

Build an algorithm that performs swapping, which exchanges the high and low order digits of any integer N.

Example: If N = 5961 result after swapping -----> 1965
If N = -18 result after swapping -----> -81
If N = 723859 result after swapping -----> 923857
If N = 9 result after swapping -----> 9

Exercise 10

In the theory of perfect numbers, EULER demonstrated that the expression: $2^{n-1}(2^n - 1)$ always returns a perfect number when the quantity in parentheses is a prime number.
Build the algorithm that allows us to obtain the first 5 perfect numbers.
Example: when n = 2, $2^n - 1$ is equal to 3 which is prime, and $2^{n-1}(2^n - 1)$ is equal to 6, and 6 is a perfect number.

Exercise 11

Construct the algorithm that gives us the value S for the N term of the sum, for any value of x read from the keyboard:

$$S = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots\ldots\ldots\ldots$$

Tutorial 3 Solution

Exercice 1

```
Algorithm  Sum
Var  i,N,S:integer
Begin
do
Write (‘Give an integer’)
Read(N)
Until N>0
S ←0
For i ←1 to N Do
    S ←S + i
End for
Write(‘The Sum of the first’,N,’ numbers is: ‘,S);
END.
```

Exercise 2:

```
factorial = 1
for i ←1 to N Do
   factorial = factorial * i
End for
Write(“The factorial “,factorial)
```

Exercise 3:

```
ALGORITHM gess
Variable target_number , guess :integer
Begin
target_number = random (1, 20)
guess = 0
Repeat
{
   Write ("Guess the number (between 1 and 20): ")
   Read (guess)
}
Until (guess == target_number)
Write ("Congratulations! You guessed the correct number:", target_number)

End.
```

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
   int target_number, guess;

   // Initialize random number generator
   srand(time(NULL));
```

```
    // Generate a random number between 1 and 20
    target_number = rand() % 20 + 1;

    // Initialize guess to 0
    guess = 0;

    // Repeat until the guess is correct
    do {
        // Prompt the user to guess the number
        printf("Guess the number (between 1 and 20): ");
        scanf("%d", &guess);
    } while (guess != target_number);

    // Congratulate the user for guessing the correct number
    printf("Congratulations! You guessed the correct number: %d\n", target_number);

    return 0;
}
```

Exercise 4

```
Algorithm  perfect
Variables: i,j,s: integer
begin
for i <--1 to 1000 do
    s<--0
    for j <--1 to i div 2 do
        if i mod j =0 then s<--s+j
    end for
   if s=i then write(i, 'is a perfect number')
end for
END
```

Exercise 5:

```
algorithm binary-convertion
variable n,i,binary:integer
begin
read(n)
i←1
binary←0
While n< >0 do
binary ←   binary+n mod 2*i
n←n div 2
i←i*10
end W
write(binary)
END
```

Exercise 6:

```
algorithm  divider
variables  sd,ld,nb,i:integer
           ok:boolean
begin
read (nb)
        i<--2
        ok<--true
        While  i<=nb div 2 do
                if nb mod i =0 then
                        if ok then
                            sd<--i
                           ok<--false
                    end if
                    ld<--i
                end if
                i<--i+1
        end while

        if ok then write('NB HAS NO DIVIDER')
        else
          if sd=ld then write('NB HAS ONE DIVIDER:', sd)
          else write('SMALLEST DIVIDER:', sd,' LARGEST DIVIDER:', ld)
          end if
        end if

END
```

Exercise 7:

```
ALGORITHM FIBO
variable n,a,b,fibo:integer
Begin
read(N)
a←1
b←1
fibo←1
For i ← 2 to N do
fibo←a+b
a←b
b←fibo
end for
write (fibo)
END
```

Exercise 8: Pascal triangle

```
for (row = 0; row < n; row = row + 1)
{
   for (space = 0; space < n - row; space = space + 1)
      print(" ")
   coefficient = 1
   for (col = 0; col <= row; col = col + 1)
   {
```

```
        Print (coefficient)
        coefficient = coefficient * (row - col) / (col + 1)
    }
    print("\n")
}
```

Exercise 9

```
Algorithm swapping
Variable  N,nb,r1,r2,cpt,n,power,x,result:integer
begin
Read (nb)
N←nb
R1←N mod 10 //the low weight If N = 5961  R1=1
Cpt←0  //the position numbers of N
While N< >0 do
          N←N div 10
          Cpt←cpt+1
End while // If N = 5961 cpt=4
Power←1
For i going from 1 to (cpt-1) do
   Power←power*10 // If N = 5961 power=1000
end for
N←nb
R2← N div power
X← (N mod power) div 10 // If N = 5961 x=96
Result←R1 * power +R2+x*10
Write (Result)
END
```

Exercise 10 Euler

```
Algorithm
Variables: nb, cpt, p, i: integer
         prem: boolean
Begin
   cpt ← 0
   p ← 2
   While cpt < 5 do
   Begin
      nb ← p - 1
      prem ← true
      i ← 2
      While i <= nb div 2 do
      Begin
         If nb mod i = 0 then
            prem ← false
         i ← i + 1
      End While
      If prem then
      Begin
         cpt ← cpt + 1
         write(p div 2 * nb)
      End
```

```
        p ← p * 2
    End While
End
```

Exercice 11 :

```
Algorithme seq
Variables X, Z, N, i integrer
            Terme,S:real
Begin
Read(x, n)
term ←  x
Z ← 1
S ←  0
For i going from 1 to N do
        S ←  S + term / Z
        term ←   - term * sqrt(x)
        Z ← Z + 2
end for

Write('for the', n, 'th term, S =', S)
End
```

Tutorial 4: Arrays and strings

Educational objectives: Learn and master the notion of compound data types, starting with structures: array, matrix and string.

Exercise 1: Write an algorithm to remove all zero values from a vector T.

Exercise 2: Write an algorithm that deletes (eliminates) a sequence of N characters from the string CH starting from position P.

Exercise 3: Write an algorithm to insert a value entered by the user into a sorted array. After the insertion, the array should remain sorted (ascending order).

Exercise 4: Write an algorithm that reads a string and then inserts a character entered by the user at a given position in the array.

Example: for N=6,
Array before insertion 'A' 'F' 'd' 'g' 'e' '/'
Character to insert: '*' position = 2
Array after insertion 'A' '*' 'F' 'd' 'g' 'e' '/' and N=7

Exercise 5: Write an algorithm that displays whether a given array is a palindrome or not. A palindrome is an array of character that reads the same forwards and backward.

Example: radar, laval, elle.

Exercise 6: Given array T of N integers, rearrange the even numbers at the beginning and the odd numbers at the end of the array. The elements and size of T are assumed to be entered by the user.

Exercise 7: Write an algorithm to determine if a given sentence contains all vowels.

Exercise 8: Write an algorithm that reads two arrays and fills a third array with the common elements between the two initial arrays, ensuring that the third array has no redundancies.

Exercise 9: Write an algorithm to fill a two-dimensional array (10x20) from the keyboard and calculate the totals per row and per column in the TotLig and TotCol arrays.

Exercise 10: Transposing a matrix involves swapping rows and columns. Write an algorithm to create and display the transpose of a square matrix.

Example: $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ Becomes after transposition $\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$

Exercise 11: Write an algorithm to determine if a square matrix is symmetric.

Exercise 12: Write an algorithm to perform and display the matrix product of two matrices, provided the number of columns in the first matrix equals the number of rows in the second matrix.

If A is a matrix of dimension (m,n) and B is a matrix of dimension (n,p), then their product, denoted C is a matrix of dimension (m,p) given by:

$$c(i,j) = \sum_{k=1}^{n} a(i,k) \times b(k,j)$$

where: c (i,j) is the element contained in the i th row and the j th column of the matrix C, a (i,k) is the element contained in the i th row and the k th column of the matrix A and b (k,j) is the element contained in the kth row and the jth column of matrix B.

Tutorial 4 Solution

Exercise 1:

```
Algorithm RemoveZeros;
Var T: Array[1..100] of integer;
   I, J, N: integer;

Begin
/* Read the exact size */
Write ('Enter the size of the array N≤100');
Repeat
   Read(N)
Until N > 0 and N ≤ 100;

/* Read elements of T */
For I ← 1 to N Do
   Read(T[I]);
End For

/* Removing zeros by shifting non-zero values */
I ← 1;
While I ≤ N Do
   If T[I] = 0 Then /* Shift loop */
      For J ← I to N-1 Do
         T[J] ← T[J+1];
      End For;
      N ← N - 1; /* Adjust array size */
   End If;
   I ← I + 1;
End While

/* Display the new array T */
For I ← 1 to N Do
   Write(T[I]);
End For;
End.
```

Exercise 2:

```
ALGORITHM RemoveCharacters;
VARIABLES
   Ch: String[100];
   p, n: integer;
BEGIN
   Write('Enter the string:');
```

```
   Read(Ch);
   Write('Enter the number of characters:');
   Read(n);
   Write('Enter the position:');
   Read(p);
   Delete(Ch, p, n);
   Write(Ch);
END.
```

Exercise 3:

```
Algorithm Insertion
Constants  N = 100
Variables
   T: [1..N] Array of integers;
   i, j, V: integer;
/* Ascending sort */
Begin
Write ('Enter the value of V:');
Read (V);

/* Find the position of V in array T */
i ← 1;
While V > T[i] and i <= N Do
   i ← i + 1;
End While

/* Overwrite the last element since the array is filled; */
If i = N Then
   T[i] ← V;
Else
   /* Shift elements greater than V to the end */
   For j ← N down to (i+1) Do
      T[j] ← T[j-1];
   End For;
   T[i] ← V;
End If
End.
```

Exercise 5:

```
ALGORITHM Palindrome;
VARIABLES
```

```
   Ch: String[100];
   i, N: integer;
   pal: boolean;
BEGIN
   Write('Enter the word to check:');
   Read(Ch);
   i ← 1; N ← Length(Ch); pal ← true;

   WHILE (i <= N div 2) AND (pal = true) DO
      IF Ch[i] <> Ch[N] THEN
         pal ← false;
      END IF;
      i ← i + 1;
      N ← N - 1;
   END WHILE;

   IF pal = true THEN
      Write('The word', Ch, 'is a palindrome');
   ELSE
      Write('The word', Ch, 'is not a palindrome');
   END IF;
END.
```

Exercise 6:

```
Algorithm EvenOdd;
Constant N = 100;
Variable i, j, x: integer;
         T: [1..N] Array of integer;
Begin
/* Move even values to the beginning */
J ← 1;
While J <= N and (T[J] mod 2 = 0) Do
   J ← J + 1;
End While
/* Move even values to the beginning (find the first odd value) */
I ← J;
While J <= N Do
   If T[J] mod 2 ≠ 0 Then /* Swap */
      X ← T[I]; T[I] ← T[J]; T[J] ← X;
      I ← I + 1;
   End If;
   J ← J + 1;
End While
```

```
/* Display the new array T */
For I ← 1 to N Do
   Write(T[I]);
End For
End.
```

Exercise 7:

```
ALGORITHM CountVowels;
VAR
   T, I: integer;
   PH, CV: string[200];
   res: boolean;
BEGIN
   T ← Length(PH); CV ← 'aeiouy';

   FOR I ← 1 TO T DO
      CASE PH[I] OF
         'a': CV[1] ← 'X';
         'e': CV[2] ← 'X';
         'i': CV[3] ← 'X';
         'o': CV[4] ← 'X';
         'u': CV[5] ← 'X';
         'y': CV[6] ← 'X';
      END CASE;
   END FOR;

   IF CV = 'XXXXXX' THEN
      res ← True;
   ELSE
      res ← False;
   END IF;

   Write(res);
END.
```

Exercise 8:

```
Algorithm Intersection;
Variable i, j, L, k, n, m: integer;
         exist: boolean;
         T1, T2, T3: [1..100] Array of integer;

Begin
```

```
/* Read the exact size */
Write ('Enter the size of the first array, then the size of the second array: ≤100');
Repeat
   Read(n, m)
Until n > 0 and n ≤ 100 and m > 0 and m <= 100;

Write ('Enter the elements of the first array');
For i from 1 to n Do
   Read (T1 [i])
End For

Write ('Enter the elements of the second array');
For i from 1 to m Do
   Read (T2 [i])
End For

k ← 1; /* k is the size of T3 */
For i from 1 to n Do
   j  ← 1;
   While j <= m Do
      If T1[i] = T2 [j] Then /* T[i] is a common element between the two arrays */
         /* Before adding this element to T3, make sure it doesn't already exist in T3 to
avoid redundancies */
         exist ← false;
         For L from 1 to k Do
            If T1[i] = T3 [L] Then
               exist ← true;
            End If
         End For
         If exist = false Then
            T3[k] ← T1[i]; /* Insert the element into T3 */
            k ← k + 1;
            For L from j to (m-1) Do
               T2 [L] ← T2 [L+1]; /* Overwrite the element in T2 */
            End For
            m ← m - 1;
         End If
      End If
      j ← j + 1;
   End While
End For

/* Display T3 */
For i from 1 to k-1 Do
```

```
   Write (T3[i])
End For
End.
```

Exercise 9:

```
ALGORITHM SumRowsColumns;
VARIABLES
   M: [1..10, 1..20] Array of real;
   RowTotal: [1..10] Array of real;
   ColTotal: [1..20] Array of real;
   i, j: integer;
BEGIN
   Write ('Enter the elements of matrix M:');
   FOR i ← 1 TO 10 DO
      FOR j ← 1 TO 20 DO
         Read (M[i, j]);
      END FOR;
   END FOR;

   Write ('Initializing RowTotal array:');
   FOR i ← 1 TO 10 DO
      RowTotal[i] ← 0;
   END FOR;

   Write ('Initializing ColTotal array:');
   FOR i ← 1 TO 20 DO
      ColTotal[i] ← 0;
   END FOR;

   FOR i ← 1 TO 10 DO
      FOR j ← 1 TO 20 DO
         RowTotal[i] ← RowTotal[i] + M[i, j];
         ColTotal[j] ← ColTotal[j] + M[i, j];
      END FOR;
   END FOR;

   Write ('Row Totals:');
   FOR i ← 1 TO 10 DO
      Write(RowTotal[i]);
   END FOR;

   Write ('Column Totals:');
   FOR i ← 1 TO 20 DO
      Write(ColTotal[i]);
```

```
   END FOR;
END.
```

Exercise 10:

```
ALGORITHM Transpose;
Variable S, i, j: integer;
Variable T: Array [1..3, 1..3] of integers;
BEGIN
   FOR i ← 1 TO 3 DO
      FOR j ← 1 TO 3 DO
         Read(T(i, j));
      END FOR;
   END FOR;

   FOR i ← 1 TO 3 DO
      FOR j ← i + 1 TO 3 DO
         S ← T(i, j);
         T(i, j) ← T(j, i);
         T(j, i) ← S;
      END FOR;
   END FOR;
END.
```

Exercise 11:

```
ALGORITHM Symmetric;
Variable S, i, j: integer;
Variable T: Array [1..3, 1..3] of integers;
BEGIN
   FOR i ← 1 TO 3 DO
      FOR j ← 1 TO 3 DO
         Read(T(i, j));
      END FOR;
   END FOR;

   S ← 1;
   FOR i ← 1 TO 3 DO
      FOR j ← i + 1 TO 3 DO
         IF T(i, j) ≠ T(j, i) THEN
            S ← 0;
         END IF;
      END FOR;
   END FOR;

   IF S = 1 THEN
```

```
      Write("Symmetric");
   ELSE
      Write("Not symmetric");
   END IF;
END.
```

Exercise 12:

```
ALGORITHM MatrixMultiplication;
CONSTANT m = 40, n = 20, p = 30;
VARIABLES
   A: [1..m, 1..n] Array of integer;
   B: [1..n, 1..p] Array of integer;
   C: [1..m, 1..p] Array of integer;
   i, j, k: integer;
BEGIN
   FOR i ← 1 TO m DO
      FOR j ← 1 TO p DO
         C(i, j) ← 0; /* Initialize to 0 */
         FOR k ← 1 TO n DO
            C(i, j) ← C(i, j) + A(i, k) * B(k, j);
         END FOR;
      END FOR;
   END FOR;

   FOR i ← 1 TO m DO
      FOR j ← 1 TO p DO
         Write(C(i, j));
      END FOR;
   END FOR;
END.
```

Bibliographic References

[1]University of Nantes, "Algorithmics." http://miage.univ-nantes.fr/miage/DVD-MIAGEv2/Algo.html (accessed on June 12, 2023).

[2] P. Geurts, "Data Structures and Algorithms."

[3] S. Akrouf, " Informatique générale " University of Bordj Bou Arréridj, 2012.

[4] M. F. Omar, "Algorithm and Programming Course," University of Sciences and Technology in Oran, Working Paper, May 2019. Accessed on: June 12, 2023. [Online]. Available at: http://dspace.univ-usto.dz/handle/123456789/380

[5] "Les algorithmes pour les Nuls grand format - Google Books." https://www.google.dz/books/edition/Les_algorithmes_pour_les_Nuls_grand_format/7uUzDwAAQBAJ?hl=fr&sa=X&ved=2ahUKEwj5z9HKt73_AhVNVKQEHQx7DQQQiqUDegQIERAH (accessed on June 12, 2023).