Ministry of Higher Education and Scientific Research
Hassiba Benbouali University of Chlef
Faculty of Exact Sciences and Computer Science

Algorithmic and data structure

Algorithmic and data structure, part 1 1st year MI

Auteur : Mme BELALIA AICHA

Academic Year :2025-2026

Introduction

This course is designed for first-year MI (Mathematics and Computer Science) students. Students will learn the fundamental concepts of algorithms and the data structures necessary for algorithm development. They will also cover basic information about the C programming language and work on several exercises with solutions in both algorithms and C.

Module plan:

The document below contains the following chapters:

- **Chapter 01**: Introduction to Computer Science and Algorithms
- **Chapter 02**: Simple Algorithms (variables and constants, data types, assignment, input/output)
- **Chapter 03**: Conditional Structures (if-then, switch)
- **Chapter 04:** Repetitive Structures (loops: for, repeat-until, while-do)
- **Chapter 05:** Arrays (vectors and matrices)
- **Chapter 06**: Records

Chapter I: Introduction to Algorithms

Introduction

In this first chapter, the student will be introduced to the fundamental concepts related to computers. The chapter aims to provide a solid foundation by exploring the basic components of a computer, including the central processing unit, memory, storage, and input/output devices. Students will also be given a brief overview of key concepts such as programs and programming, highlighting their roles in instructing computers to perform specific tasks.

Furthermore, this chapter will guide students through the essential steps of problem-solving using a computer. By understanding how to analyze a problem and represent its solution in a structured way, students will learn how to define an algorithm, a step-by-step procedure that forms the basis of all computer programs. By the end of the chapter, learners will be equipped not only with theoretical knowledge but also with the practical ability to approach simple problems methodically and prepare them for programming.

1. Computer Science:

The term **"computer science"** was coined in 1962 by French computer scientist Philippe Dreyfus and was accepted by the French Academy in April 1966. Computer science is the discipline focused on the automatic processing of information. It encompasses the study of computers and algorithmic

processes, including their principles, hardware and software design, applications, and societal impact. It focuses on two complementary areas:

- **Programs or software**, which describe a process to be executed.
- **Machines or equipment (hardware)**, which carry out such processing.

2. The Computer:

- A computer is a highly powerful device that allows the processing of information at very high speed, with a high degree of accuracy, and the ability to store large amounts of data.
- It can receive input data, perform operations on it according to a program, and finally provide output results.
- A computer consists of two main components: **hardware** and **software**. The combination of these two parts forms what is known as a **computer system**.

2.1 Hardware

Hardware refers to the physical, visible components of a computer system, such as the monitor, CPU, keyboard, and mouse. It is divided into:
The central unit, which is responsible for processing and coordinating all operations.
Peripherals (input/output/storage devices).

2.2 Software

Software refers to a set of instructions or programs that enable hardware to perform specific tasks. It defines how information is processed by computer equipment.
Software must be installed on hardware to function properly, and hardware must be present for tasks to be executed.
Two main types of software:

- **Operating systems** (e.g., Windows, Linux, Unix, MacOS, iOS).
- **Application software** (e.g., Word, Paint, chat applications).

A computer is made up of the central unit and input/output peripherals.

I. **Input/output peripherals:** Devices that allow communication with the outside world (e.g., keyboard, mouse, printer, etc.).

II. **The central unit contains:**

a. **Motherboard:** A circuit board that connects all components and coordinates communication between them.

b. **Hard disk:** Used to store information permanently.

c. **RAM (Random Access Memory):** Used to store information currently in use. It is volatile.

d. **Processor:** The brain of the computer; it organizes data exchanges between different components (hard disk, RAM, graphics card) and performs calculations.

3. **Von Neumann Architecture:**

A computer's operation is based on the Von Neumann architecture, in which a single memory stores both data and instructions. The CPU (Central Processing Unit) controls operations, interacting with memory, input/output devices, and the ALU (Arithmetic and Logic Unit) to execute programs. The following diagram illustrates this structure:

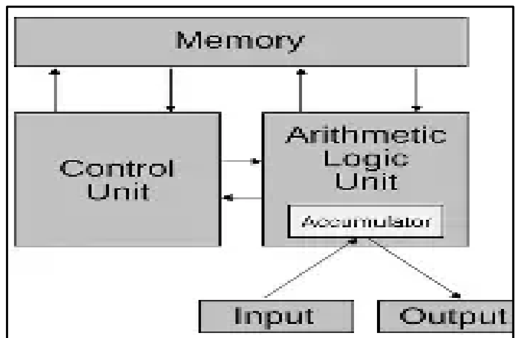


Fig 1: Von Neumann Architecture

4. **Roles of each part of a computer:**

a. **Arithmetic and Logical Unit (ALU) or Processing Unit:** Performs basic operations such as arithmetic operations (addition, subtraction, multiplication, etc.) and logical operations (AND, OR, etc.).

b. **Control Unit:** Responsible for sequencing operations.

c. **Memory:** Stores both data and programs that instruct the control unit on what operations to perform. Memory is divided into volatile memory (programs and data in execution) and permanent memory (programs and basic system data).

d. **Input/output devices:** Interfaces for interacting with the computer. Typically, a keyboard serves as an input device and a screen as an output device.

5. **Other architectures:** There are several alternatives and evolutions:

- **Harvard Architecture:** separates instruction memory from data memory, commonly used in some microcontrollers and embedded systems.
- **Parallel / Multi-Processor Architecture:** multiple processors work simultaneously to speed up computation.

- **RISC / CISC Architecture:** different processor designs, either simplifying or complicating the instruction set.
- **Modern Architectures (GPU / Cloud / Quantum Computers):** designed for massive or specialized computation.

6. Programming

Programming includes all activities related to defining, writing, developing, and executing computer programs—a sequence of instructions that a device must follow.

7. Program

A program is an orderly sequence of instructions or operations written in a programming language to solve a given problem. Programs help humans save time and avoid mistakes.

8. Algorithm

An algorithm is the basis of all computer programming. It is a procedure used to solve a problem or perform a computation. An algorithm consists of a series of instructions arranged in a precise order, allowing systematic problem solving.

Algorithms are written in a human-readable form (not directly understood by a computer).

A program is an algorithm translated into a programming language.

The word "algorithm" comes from the Latinized name of the Persian mathematician Al-Khwarizmi.

9. Instruction

An instruction is the basic element of a program or algorithm. It is a set of characters that defines, in a given programming language, an elementary operation that a computer must execute.

Problem-solving schemes:

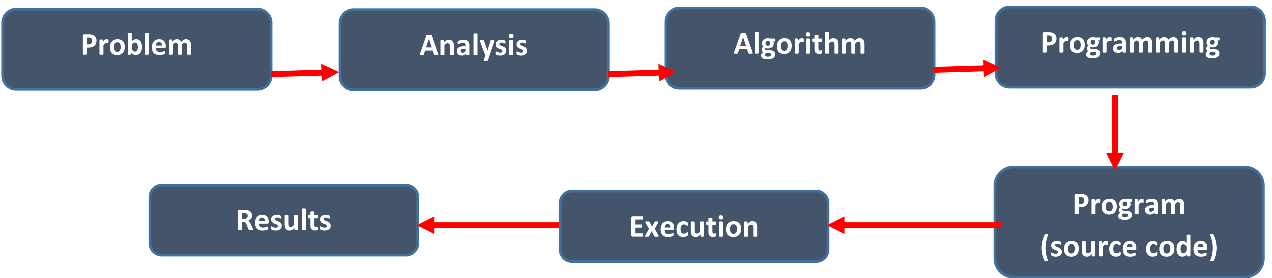


Fig 2: Diagram Illustrating the Steps to Solve a Problem

Chapter 02: The Simple Algorithm

Introduction

After studying the basic concepts of computer science and the functioning of a computer, we begin this chapter, which is based on the basic notions needed to design and write simple algorithms. It covers:

- The **general syntax of an algorithm** (declaration, begin, end).
- **Input and output instructions** (read and display data).
- **Assignment instruction** (store and update values).
- **Data types** (integer, real, character, boolean, …).
- **Operations on each type** (arithmetic, relational, logical, concatenation, …).
- The correspondence between **algorithm syntax and C language syntax**.
- Examples of **simple algorithms** (addition, average, etc.).

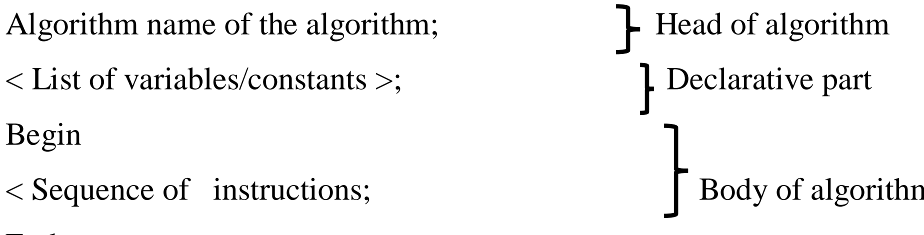
1. Characteristics of algorithms

1.1 General structure:

An algorithm is composed of three main parts:

1. **The head:** this part is used to give a name to the algorithm. It is preceded by the word "Algorithm".
2. **The declarative part**: in this part, we declare the different objects that the algorithm uses (constants, variables, etc.);
3. **The body of the algorithm**: This part contains the instructions of the algorithm. It is started by the word "**begin"** and finished wit "**End"**.

1.2 Syntax:



Algorithm name of the algorithm; } Head of algorithm
< List of variables/constants >; } Declarative part
Begin
< Sequence of instructions; } Body of algorithm
End.

1.3 Variables: Variables are symbolic names given to data where the value of the **data stored may change during the execution** of the program. In effect, a variable is **a named area of memory used to store data.**

1.4 Constants: **Constants** are **symbolic names** given to data where the value of **the data cannot change during the execution of the program**. Many constants, such as PI, are built into programming languages.

So: A **variable** is something that can be changed. However, the **constant is** something that cannot be changed during the execution.

1.5 Declaring variables and constants:

1. The declaration part consists of **listing all the variables and constants** that will be needed during the algorithm.
2. Each declaration of a variable must include **the variable name (identifier**) and its **type**.
3. Each declaration of a constant must include **the constant name (identifier**) and its **value**.

Examples:

1.6 Variable declaration: synatx

Variable surface: real;

Variable a: integer;

Variable a, b, c, d: integers;

Variable absent : :booléen;

Constant declaration syntax:

Const Pi=3.14 ;

2. Identifiers:

It is a unique name, which is given to an entity to distinctly identify it meanwhile the execution of the source-code.

The rules for naming identifiers are as follows −

- Identifier names are unique.
- Cannot use a (LDA Latent Dirichlet Allocation) keyword as identifiers.
- Identifier has to begin with a letter or underscore (_).Shouldn't begin with number .
- It should not contain white space.
- Special characters are not allowed.(% (percent) &and , @(at) , # (hash) , * /?!(exclamation mark) ….)
- Identifiers can consist of only letters, digits, or underscore.(capital letters and small letters A… Z, a..z, numbers 0…9,_).
- In algorithm, there is no difference between lowercase and uppercase letters, **unlike in C language.**
-

Examples:

Correct	Not correct	Explanation.
The_whidt FnameLname Whidt15	5whidt Fname&lname The whidt _whidt.	Begin with number Contain a special character & Cotain white space Begin with (_).

3. Data types

The type of a variable is the set of values it can take.

For example, a **logical type variable** (**boolean**) can take the values **True** or **False**.

3.1 The different types used in algorithms:

□ **Integer** Type is used to handle positive or negative integers.

For example: 5, -15, etc.

□ **Type Real** is used to manipulate decimal numbers.

For example: 3.14, -15.5, etc.

Character Type : allows you to manipulate alphabetic and numeric characters.

For example: 'a', 'A', 'z', ' ?', '1', '2', etc.

□ **Character String Type**: used to manipulate character strings used to represent words or sentences.

For example: “hello”, “Sir”, etc.

□ **Logical Type (Boolean)**: uses logical expressions. There are only two Boolean values: **True** and **False**

3.2 There are two families of types:

□ **simple types:** these are types that are supported and recognized by the majority of programming languages. When writing the algorithm or program, there is no need to declare them in the declarative part reserved for types.

□ **Compound or complex types**: these are types that are constructed from simple types but must be declared in the part reserved for types: arrays, strings, records, etc.

Example:

Variables n: integer

r: real

a, b: Boolean; student_name: string:

4. Possible operations for each type:

INTEGER	Addition substraction multiplication division integer division. modulo(the remainder of the integer division). x exponent y. Comparison.	+ (plus) - (minus sign) * (asterisk) / (slash) Div in VB (/ in C) MOD in VB % in C ^ in VB , and pow(x,y) in C. = (equal),< less than . > greater than .<=.>=. ≠
In algorithmic, we symbolize the integer division by DIV and the remainder of the integer division by MOD.		
REAL	Addition substraction multiplication division Exponent Comparison.	+ - * / ^ =,< . > .<=.>=. ≠ , <>(in VB)
CHARACTER	Comparison	
string (chain) of characters	Concatenation between characters Comparison	(+ , & in VB). = , ≠
BOOLEAN	And Or NOT XOR	And in VB , && in C. OR in VB , || in C . NOT in VB, ! in C

Examples:

Var a, b , c : real;
I,n,s : integer
L, m : Boolean;

A= 5 , b= 1,5 , c=2	a/c , a/b , a+b ,	2.5, 3.33 , 6.5
I= 5 , n = 2, s=4	s/n , Idiv n , I mod n , 5^2	2, 2 , 1 , 25

L= true, m= false	L and m , L or m , l xor m	False , true , True
L= true , m= true	L and m , l xor m , not m	True , false , false.

Boolean type:

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

- **AND** → testing / filtering (networks, bit masks, multiple conditions).
- **OR** → combination / activation (options, settings, flags).
- **XOR** → difference / inversion (cryptography, error detection, value swapping).

The expression 5 > 2 is true . however this expression 7< 4 is false .

For the character and character string type: we find the concatenation operation

'A'+'B' = = 'AB'.

"hellow "&" "&" every one " is "hello every one ".

The operation calculates the length of the chain

Length (C) / len: len('hello')= = 5.

5. Order of descending priority of arithmetic and concatenation operators:

- Parentheses
- "^ : caret _circumflex" (raise to power)
- "- : dash " (change of sign)
- "* : asterisk ", "÷ divide "
- div
- "mod"
- "+" and "-"
- & (concatenation)

6. Order of decreasing priority of logical operators:

- "not", "and", "or". "xor"

7. Order of decreasing priority of comparison operators:

"=" , " < >" , "< " , " >", "<= ", " >= ".

8. Basic instructions

An instruction is an elementary action commanding the machine to perform a calculation, or communicate with one of its input or output devices. The basic instructions are:

8.1 The assignment instruction

Assignment allows you to assign a value to a variable. It is symbolized in algorithmic terms by " ←".

The sign "←" specifies the direction of the assignment.

* An assignment statement is executed as follows:

- evaluation of the expression located to the right of the instruction, and

- assignment of the result to the variable located to the left of the instruction.

The expression can be:

• a constant (c ← 10)

• a variable (v ← x)

• an arithmetic expression (e ← x + y)

• a logical expression (d ← a or b).

Example :

Algorithm Calculate01

Variables A, B, C, D : integer ;

Begin

A ← 10

B ← 30

C ← A+B

D ← C*A

End .

Memory state:

This algorithm contains 4 instructions:

	0	1	2	3	4
A	#	**10**	**10**	**10**	**10**
B	#	#	**30**	**30**	**30**
C	#	#	#	**40**	**40**
D	#	#	#	#	**400**

Example 02:

Algorithm logic

Variable A, B, C: Boolean;

Begin

A ← True;
B ← False;
C ← A and B;
End

Memory state:

This algorithm contains 3 instructions:

	0	**1**	**2**	**3**
A	#	**1**	**1**	**1**
B	#	#	**0**	**0**
C	#	#	#	**0**

Example 03

Algorithm ex03

Variable x,y: integer;
z: real;
a,b: char ;
Classify the following instructions into two types (correct/incorrect):

x ← y ; a ← z ; y ← x ; a ← b ; x ← z ; z ← x ; x ← a ; b ← y ; b ← a ;

Corrects instructions	Incorrects instructions
x ← y ; y ← x ; z ← x ; a ← b ; b ← a ;	a ← z ; x ← z ; x ← a ; b ← y ;

8.2 The input instruction (read)

This instruction is very important in an algorithm. It allows you to read input values and assign them to variables stored in memory. The assigned values are often data entered from an input device such as the keyboard.(**so when we use this instruction to read a value , on the screen the cursor blinks until the user types a value , so this value will assigned to this variable in memory**)

Syntax:

read (identifier)

Examples:

read(A);

read(A, B, C):

The **read(A)** instruction allows the user to enter a value using the keyboard. This value will be assigned to variable A.

All instructions must finish with semicolon (;)

8.3 The output instruction (write)

This instruction is also of great importance in algorithms. It allows the data resulting from processing carried out by the algorithm (value, text, etc.) to be written as output by displaying them, for example, on an output device such as the screen.

Syntax:

write (expression)

Expression can be a value, a result, a message, the content of a variable, etc.

Example: calculate and display the sum of two integers.

Algorithm sum

Var x, y,z : integer ;

Begin

read (x) ;

read(y) ;

z ← x+ y ;

write (z) ;

Fin.

Messages and comments:

Display a message on screen

Syntax

write (''message……''); message will be shown on sceen must be between two parentheses ( ) and between double quotes " "

Display a comment in algorithm / C :

Syntax:

/* comment ……*/ or // comment ……

Begin with slash followed by an asterisk end finished with asterisk followed by slash

Or

Begin with double slash . (but we repeat it in each line)

Example: show student information's

Algorithme student

Begin

/* This algorithm displays student information */

write (‘’ my first name and last name are :’’) ;

// show age

write (‘I am 18 years old ‘’) ;

End

Syntax and declaration of variables in C language

Data type:

Data type	**Meaning**	**Accepted value rang**
Char	character	-128 to 128
Int	Integer	-32 768 to 32 767 -2 147 483 648 to 2 147 483 647
Short int	Short integer	-32 768 to 32 767
Long int	Long Integer	-2 147 483 648 to 2 147 483 647
Float	Real	$3.4*10^{-38}$ to $3.4*10^{38}$
Double	Double float	$1.7*10^{-308}$ à $1.7*10^{308}$
Unsigned short int	Unsigned short integer	

Booleen type in C:

In fact, there is no real “booleen” type in C: However, the C standard says that any integer value other than 0 will be considered true (in case of logical testing) and that the value 0 will be considered false.

Example:

```
int x = 3, y = 4 ,  result ;

result = (x < 3) && (y == 4);  // result is  0

result = (x < 3) || (y == 4);  // result is  1 ;
```

9. How to write a first C program?

9.1 General syntax

[Preprocessor directives]

[external variable declarations]

[secondary functions]

int main ()

{ internal variable declarations

Instructions

return 0;

}

9.2 [Preprocessor directives]:

C's directives

#include:

Whose role is to copy the content of a file into the current file. Typically used to include library headers, such as math functions (#include <math. h>), or standard input/output functions (#include <stdio.h>).

Syntax :

#include <systemFilename.h> // for standard library header files

#include "userFilename.h" /* for user-created files */

<stdio.h>. /*Standard Input/Output Header , is the header of the C standard library declaring macros, constants and function definitions used in input/output operations.

<math.h>. The math.h header file contains declarations of mathematical functions.(sin , cos , pow , ….).

<stdbool.h>. Allows you to define a Boolean pseudo type (in fact an unsigned integer).

The #define directive

The **#define** directive allows you to define:

• symbolic constants,

• macros with parameters.

Examples :

#include <stdbool.h>

#define **bool** unsigned int

#define **true** 1

#define **false** 0

Example 01 :

#include <stdio. h>

#include <math. h>

#include <stdbool. h>

int main ()

{…

Declaration
Instructions

return 0;
}

main () : A main is a predefined keyword or function in C. It is the first function of every C program that is responsible for starting the execution and termination of the program. The void main() indicates that the main() function will not return any value, but the int main() indicates that the main() can return integer type data

9.3 Variable declaration in C :

Syntax :

Data type : identifier ;

(Variable) type variable name

Example

int a ;

int a=5 ; declaration and initialization of the variable a;

int a,b ;

int a=5 , b=2 ;

int a ; b=2 ;

9.4 Declaration of constants:

We can declare a constant in C by using **const** keyword

Synatx: const int pi=3.14; we write this instruction after main( ) {

Or using **the #define "directive"**

Syntax: **#define** Pi 10 without writing the ";" : semicolon.

Examples :

```
#include <stdio. h>
#define PI 3.14
main ( )
{     const int a=10 ;
      const float b=7.77 ;
 return 0;
}
```

10. Basic instructions in C language:

10.1The assignment instruction in C:

Syntax

Variable = expression;

/* expression can be a value, a variable, an operation*/

Example:

int a, b,sum;

Sum = a+b;

A= 5 ;

Exercise 01: mathematic

```
#include <stdio.h>
int main()
{
int x , y , result ;  //  declaration of integer variables
x = 17;    // assigning values to variables x, and y
y = 3;
result  = x + y;           // result  is 20
result  = x - y;           // result  is 14
result  = x * y;           // result  is 51
result  = x / y;           // result  is 5    Warning integer division!
result  = x % y;           // result  is 2
result  = - result  ;      // result  = -2
return 0; }
```

Exercise 02: comparison

3 != 4 will evaluate to true, (different)

3 == 4 will evaluate to false, (equal)

7 == 7 will evaluate to true,

3 <= 8 will evaluate to true, etc. (less than or equal).

10.2 Input instruction in C

Algo : read (identifier) ;   in C :   scanf(identifier);

10.3 Output instruction

Algo : write (expression) ;   in C: printf(expression) ;

Example 01 :

Algo :	**C**
write ("how old are you ");	printf("how old are you ? "); /* message*/

read (age);   scanf("%d", &age);

As you notice, I preceded the reading in C program with “%” symbol and the “d “letter, followed with coma and “&” symbol, so %d , is the integer data type format , it is used to tell the compiler about the type of data to be printed or scanned in input and output operations. And “&” means the address of the variable age.

The following table shows some format data types:

10.4 The format specifier in C:

Type	Conversion indicator(s)
Bool	d (or i)
Char	C
signed char	d (or i)
Short	d (or i)
Int	d (or i)
Float	f, e (or E) or g (or G)
Double	f, e (or E) or g (or G)
long double	Lf, Le (or LE) or Lg (or LG)

11. Exercices and examples

Example 02 :

```
#include <stdio.h>
int main(void)
{
   char z = 'z';
   int c = 30;
   long int  d = 40;
   float e = 50.;
/* display values */
   printf("%c\n", z);
   printf("%d\n", c);
   printf("%li\n", d);
   printf("%f\n", e);
```

```
    return 0;
}
```

Example 03

```
int x, y;
scanf("%d %d", &x, &y);
printf("x = %d | y = %d\n", x, y);
```

Example 04 : write the algorithm and a program calculates the sum of 3 numbers

Algorithm

```
Algorithm sum
Variable a, b : integer;
         c: real;
Begin
   write ("give me 2 integer numbers ");
   read (a, b);
   write (" give me one  reel number");
   read ( c);
   write (" the result is :");
   write ( a+b+c);
End
```

C program

```
#include <stdio.h>
int main(void)
{   int a,b;
    float c;
    printf ("give me 2 integer numbers and one reel number);
    scanf("%d%d%f",&a,&b,&c);
    printf("the result is %f", a+b+c);
    return 0;
}
```

Or with the using of assignment instruction:

Algorithm

```
Algorithm sum
Variable a, b : integer;
         c , res: real;
Begin
   write ("give 2 integer numbers ");
   read (a, b);
   write (" give me one  reel number");
   read ( c);
   res ← a+b+c;
   Write ("the result is :", res);
End .
```

C program

```
#include <stdio.h>
int main(void)
{   int a,b;
    float c,res ;
    printf ("give 2 integer numbers and one reel number);
    scanf("%d%d%f",&a,&b,&c);
    res = a+b+c;
    printf("the result is %f", res);
    return 0;
}
```

Example 05 :

```
#include <stdio.h>
main()                                                      // Main function
{
 int i;    float x;   char ch; char chaine[20];             // Variable declaration
printf("Give the value of  ch :"); scanf("%c",&ch);         // Writing and reading a character
printf("Give the value of  i :");  scanf("%d",&i);          // Writing and reading an integer
printf("Give the value of  x :"); scanf("%f",&x);           // Writing and reading a real
printf("Give the value of   chaine :"); scanf("%s",chaine); // Writing and reading a string
 }
```

Exercise 01 : Write an algorithm that calculates the average of two reel notes entered by the user.

Solution: We need two real numbers, so two variables of type float. A third variable can be used to store the average, but we can also display the result directly without using this third variable.

```
Algorithm average
Var   note1, note2,sum : integer ;
 moy : reel ;
Begin
/* calculate of average*/
write  ('' give the first mark \n'') ;
read (note1) ;
write ('' give the second  mark \n '') ;
read (note2) ;
sum ←    note1+note2 ;
moy    ←    sum/2 ;
write ('' the su mis \n '', sum ) ;
write  ('' the average is  :'') ;
write  (moy) ;
End
```

C program :

```
#include<stdio.h>
int main()
{
int a, b, sum;
```

```
float  moy ;
printf("type two integer numbers  \n");
scanf("%d%d", &a, &b);
sum = a + b;
moy = (float)sum/ (float)2 ;
printf("the sum of two numbers is  = %d\n", sum);
printf("the average is  = %f\n", moy) ;
return 0;
}
```

Exercise 02 :

Write an algorithm that calculates and displays the volume of a sphere. The algorithm will ask the user to enter the value of radius r. **Volume** $V = (4\pi/3) \times r^3$

Solution

```
Algorithm sphere
Const  Pi = 3.14 ;
Var
R : real ;
V : real ;
Begin
Write  ("This program calculates the volume of a sphere ") ;
Write  ("Enter the radius of a sphere ") ;
read (R) ;
V <– (4/3) * Pi * R * R * R ;
Write  ("the  volume of  this  sphere is :") ;
write (V) ;
END.
```

```
Algorithm sphere
Var
R : real ;
V : real ;
Begin
Write  ("This program calculates the volume of a sphere ") ;
Write  ("Enter the radius of a sphere ") ;
read (R) ;
V <– (4/3) * 3.14 * R * R * R ;
Write  ("the  volume of  this  sphere is :") ;
write (V) ;
END .
```

Solution en C

```
#include<stdio.h>
int main( )
{   const  float pi=3.14 ;
    float r,v ;
    printf('' Enter the radius of a sphere '') ;
    scanf (''%f'' , &r) ;
    v=(4/3)*pi*r*r*r ;
    printf('the  volume of  sphere with the radius %d is %f\n'',r,v) ;
    return 0 ;
}
```

12. The flowchart :

12.1 Definition An algorithm can be:

- Represented graphically by a flowchart or algorithm.

• Written in literal form with an algorithmic or pseudo-code language.

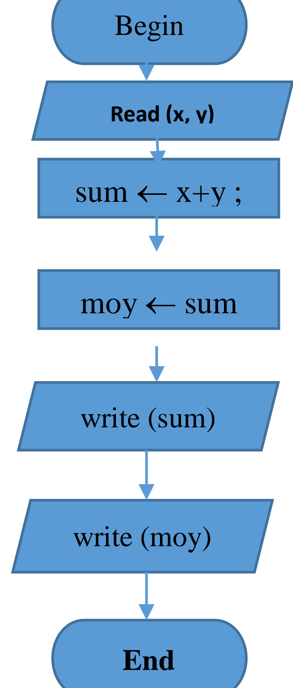
The organization chart : A flowchart is a diagram that shows an overview of a program. Flowcharts normally use standard symbols to represent the different types of instructions. These symbols are used to construct the flowchart and show the step-by-step solution to the problem. Flowcharts are sometimes known as flow diagrams.

12.2 Flowchart Symbols :

Here is a chart for some of the common symbols used in drawing flowcharts.

Symbol	Symbol Name	Purpose
	Start/Stop	Used at the beginning and end of the algorithm to show start and end of the program.
	Process	Indicates processes like mathematical operations.
	Input/ Output	Used for denoting program inputs and outputs.
	Decision	Stands for decision statements in a program, where answer is usually Yes or No.
↓	Arrow	Shows relationships between different shapes.

Example 01: write a computer flowchart for the exercise N°01:



Chapter 03: Control structures

Conditional instructions

Introduction

In our daily life, we often make choices based on conditions.

- *If it rains, we take an umbrella. Otherwise, we go without one.*
- *If I finish my homework, I can play. Otherwise, I must keep studying.*
- *If the traffic light is red, we stop. If it is green, we go.*

These everyday examples show how decisions depend on certain conditions.

In programming, we use a special tool to do the same thing.

Conditional instructions, also called test instructions, allow choices to be made in a program. They make it possible to change the flow of the program according to conditions. There are three main types of conditional statements:

The if statement,

The if–else statement, and

The switch…case instruction.

These instructions allow the program to choose according to conditions.

They are used to specify:

- The different possible cases of a problem (“conditions”),
- And what to execute in each case (“branches”).

A conditional instruction is composed of:

- **One condition to test** (a boolean expression).
- **One or two branches** (instruction blocks), of which only one is executed.

1. The if conditional structure

1.1 Algorithmic syntax:

```
If (condition) then
    Instructions
End if
```

Example 1 – Algorithm: Negative number

```
Var x : integer ;
Begin
    write (‘’ give an integer number ) ;
    read (x) ;
    if (x < 0) then
```

write (x,‘ is negative ‘’) ;

end if ;

End

1.2 The C Syntax

Syntax:

```
If (condition)
{
  Instructions
}
```

Example :

```
#include <stdio.h>
int main() {
   int x;
   printf("Give an integer: ");
   scanf("%d", &x);

   if (x < 0) {
      printf("The number %d is negative\n", x);
   }
   return 0; }
```

2. The if .. then,…. else structure

2.2 Algorithmic syntax

if (condition) then

instructions

else

instructions

End if

Example 02:

Algorithm: Positive or Negative number

Var x: int;

Begin

write (“give an integer number”);

read(x);

if (x < 0) then

```
        write (x," is negative");
        else
        write (x, “is positive”);
    End if ,
    End
```

2.2 C Syntax:

```
if (condition)
{
instructions
}
else
{
Instructions
}
```

Example 02:

```
#include<stdio.h>
int main( )
{  int x ;
   printf(‘’ give an integer’’ );
   scanf(‘’%d’’, &x);
   if (x < 0)
     { printf ("the number %d is negative \n",x);
     }
     else
     {printf ("the number %d is positive\n",x);
     }
return 0;  }
```

3. Nested test: several tests on the same values

3.1 Algorithmic syntax (nested form):

Syntax:

```
if (condition1) then
instructions
else
  if (condition2) then
```

```
instructions
else
  if (condition3) then
        instructions
       else
        Instructions
     End if ;
   End if ;
End if
```

3.2 C syntax:

Syntax :

```
if (condition1)
 { ... }
else if (condition2)
 { ... }
else if (condition3)
  { ... }
else
   { ... }
```

Example 03 – Algorithm: Sign of a number

```
Var x : integer
Begin
  write ("Give an integer")
  read(x)
  if (x < 0) then
     write (x, " is negative")
  elseif (x > 0) then
     write (x, " is positive")
  else
     write (x, " is zero")
  end if
End
```

C solution :

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

```
int main() {
    int x;
    printf("Give an integer: ");
    scanf("%d", &x);
    if (x < 0) {
        printf("The number %d is negative\n", x);
    }
    else if (x > 0) {
        printf("The number %d is positive\n", x);
    }
    else {
        printf("The number is zero\n");
    }
    return 0;       }
```

4. Simplified algorithm syntax (with elseif):

We can write only one **endif** in the program

```
if (condition1) then
   instruction
elseif (condition 2) then
      instructions
elseif (condition3) then
      instructions
 else
      instructions
End if ;
```

Example :04 : Write a program that **asks the user to read 3 integers** nbr1, nbr2, nbr3 and **displays** the **maximum** (max) of its numbers.

Solution

The algorithm:

Algorithm Maximum of 3 numbers

Var nbr1,nbr2,nbr3,max **: integer ;**

Begin

write (‘’ give 3 integer ‘’) ;

```
read (nbr1,nbr2,nbr3) ;
if (nbr1 > nbr2 ) then
   if ( nbr1 > nbr3) then
      max ← nbr1 ;
    else
       max  ← nbr3 ;
    endif ;
 else
    if (nbr2  > nbr3 ) then
      max  ←  nbr2 ;
    else
       max  ← nbr3 ;
   endif ;
  endif ;
  write (‘’ the  maximum is ‘’, max ) ;
end  .
```

Solution in C

```
#include <stdio.h>
int main() {
    int num1, num2, num3, max;
    // Input
    printf("Enter 3 numbers: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    // Processing
    if (num1 > num2) {
        if (num1 > num3) {
            max = num1; // num1 is greatest
        } else {
            max = num3; // num3 is greater than num1
        }
    } else {
        if (num2 > num3) {
            max = num2; // num2 is greatest
        } else {
```

```
            max = num3; // num3 is greater than num2
        }
    }
    // Output
    printf("The maximum is = %d\n", max);
   return 0;
}
```

5. Multiple Choice Structure: switch

The switch structure allows you to select **one choice among several possibilities**.
It is more suitable than if ... else if ... when there are many alternatives.

Syntax:

```
switch (selector) :
case <list of values-1>: instructions
case <list of values-2>: instructions
case <list of values-3>: instructions
…..
default : instructions
End switch
```

Remarks:

- The *selector* can be a scalar variable or an arithmetic/logical expression.
- Each case can contain:
  - A **single value**
  - A **list of values** (e.g., 2,3,4)
  - Or a **range of values** (e.g., 1..5)
- Execution continues after end switch.
- Real numbers and strings are usually **not allowed** as case values.
- The **switch** structure evaluates the "selector", proceeds to compare it respectively with the values in the lists. In case of equality with a value, the corresponding actions, which are in front of this value will be executed.
- In front of "Case", there can be a single value, a sequence of values separated by commas and/or a range of values.
- After processing the corresponding sequence of actions, execution continues after **endswitch.**

Noticed

- ✓ The selector must have the same type as the values in front of the cases.
- ✓ The type of these values must be neither real nor string.

6. Examples and Exercises

Example 05:

Write a program that asks the user to give an integer, the program displays the day name equivalent to that number (for example for 1 : Saturday .

Solution:

```
Algorithme day
Var   day : integer ;
begin
write (“give an number\n”) ;
read (day) ;
switch (day ) in
Case 1 : write (‘’ saturday ‘’) ;
Case 2 : write (‘’sunday ‘’) ;
Case 3 : write (‘’monday ‘’) ;
Case 4 : write (‘’tuesday ‘’) ;
Case 5 : write (‘’wednesday ‘’) ;
Case 6 : write (‘’thursday ‘’) ;
Case 7 : write (‘’friday ‘’) ;
default : write (‘’ mistake ‘’) ;
endswitch ;
end.
```

In C language:

Syntax:

```
switch (Variable)
 {
case value 01 :
        Liste d'instructions;
        break;
case value 02 :
        Liste d'instructions;
        break;
```

case value ... :

Liste d'instructions;

break;

default:

Liste d'instructions;

}

Remarks:

- The *selector* must be of type **integer** or **character**.
- Each case must contain a **single constant value** (no ranges, no lists).
- The break; statement is used to stop execution after each case.
- default is optional and used when no other case matches.

Example :05

```
#include <stdio.h>
int main()
{   int day;
      /* Enter input data */
   printf("enter the number of the day : ");
   scanf("%d", &day);
    switch(day)
   {  case 1:  printf("saturday ");
          break;
      case 2:  printf("sunday");
          break;
      case 3:  printf("Monday ");
          break;
      case 4: printf("tuesday ");
          break;
      case 5:  printf("wednesday ");
          break;
      case 6: printf("thursday ");
          break;
      case 7:  printf("friday ");
          break;
```

```
    default: printf("mistake.");
  }
  return 0;
}
```

Exercise 01:

```
Algorithm days:
Var day : integer;
Begin
Write ("give the number of day");
Read(day);
switch (day) :
  case 1,2,3,4,5 : write("Weekday")
  case 6,7       : write("Weekend")
  default        : write("Invalid day")
end switch
```

in C :

```
#include <stdio.h>
 int main()
{     int day;
      printf("enter a number ");
    scanf("%d", &day);
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:
     printf("Weekday\n");
     break;
  case 6:
  case 7:
     printf("Weekend\n");
     break;
  default:
```

```
    printf("Invalid day\n");
}
```

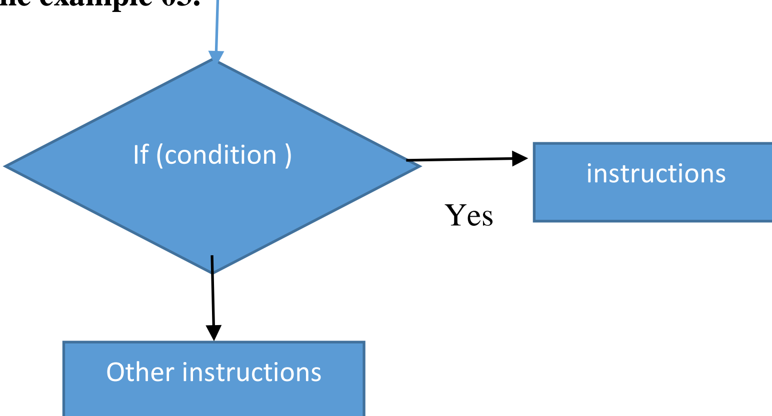
Exercise 02:

Write a program to input a number from the user and check if that number is even or odd using "switch case".

```
#include <stdio.h>
 int main()
{     int num;
     printf("enter a number ");
    scanf("%d", &num);
    switch(num % 2)
    {       case 0:  printf("even");    /* if  n%2 == 0 */
            break;
            case 1:   printf("odd");    /* else  n%2 == 1 */
            break;
    }
     return 0;
}
```

7. The flowchart:

Write the flowchart for the example 03.



Exercise 03:

Write an algorithm that asks you for the current time and displays the time after one second later. For example, if the user types 21, then 32, then 8, the algorithm should respond: "In one second it will be 21 hour(s), 32 minute(s), and 9 second(s)."Obviously, the user must enter a valid date.

Solution:

```
Algorithm time
Var h,m,s : integer;
Begin
Write (''enter the current time hour then minute and second'');
read(h,m,n);
if (h<0 or h>23) or (m<0 or m>59) or (s<0 or s>59) then
write ('' this values are note correct'');
else
  s←s+1;
   if (s>59) then
        s←0;
        m←m+1;
                if (m>59) then
                m←0;
                h←h+1;
                if(h>23) then
                        h←0;
                endif
            endif
   endif
write (''the time is '',h,''hours'',m,''minutes'',s,''secondes'')
endif
end.
```

Chapter 04: Iteration Control Structures (LOOPS)

Introduction: In our daily life, we often repeat the same action many times:

- When a teacher wants to calculate the average of all students in the class, they don't just stop after one student.
- Printing student cards:

The university prints one card, then the next, and continues until all cards are done.

Instead of describing every single step one by one, we usually say:

- "Do this action for each student."
- "Repeat this task until all cards are printed."
- "Continue the process until there are no more elements left."

This is exactly what we call a **loop (iteration)** in algorithms.

A loop allows us to repeat a set of instructions **without writing them many times**, which makes our program shorter, clearer, and more efficient.

Utility:

It is an algorithmic structure that allows processing to be repeated several times for the same data.

For example:

Write an algorithm that displays the word "**hello**" **50 times.**

50: number of repetitions.

This is the process of repeating the word **"Hello."**

Note: we say the number of repetitions or the number of iterations (which means the passage from one step to another in a loop).

• There are three types of loops:

- The structure **for (...) do,** **(loop1)**
- The structure **while** … do **(loop2)**
- The structure **repeat…until. (loop3)**

1. The" for" loop

1.1 Algorithmic Syntax:

for (initial value of the counter...to final value, step) do

Instructions

End for ;

Noticed : This loop is used whenever we know the **number of iterations to perform**. (**before** starting the process).

The counter: this is the **variable** that controls the **number of iterations** to perform.

1.2 Examples and exercises

Example 01: Display the word "hello" **10 times**

```
Algorithm loop1
    Var i: integer;
    Begin
    for (i = 1, to 10, step 1) do
        write ("hello");
    endfor;
    END
```

Explanation :

i=1: the **initial counter** value,

10 the **final counter** value.

1 is the **counter step** (**at each iteration** we **add 1**)

If we change the counter to 2, how many times will we see the word "hello"

i=1, i=3, i=5, i=7, i=9 (5 times)

Example 02: display all odd numbers between 1and 6

```
Algorithm odd
Var i : integer;
Begin
for (i = 1, to 6 , 2 ) do
write (i);
endfor
End
```

Execution

The **initial counter value** is **1 ,** so **1** is an **odd** number, it will be displayed.

The **counter step** is **2 ,** so the **next value** of **i** is **3 , 3** is an **odd** number , it will be displayed.

The **counter step** is **2 ,** so the **next value** of **i** is **5 , 5** is an **odd** number , it will be displayed.

The **next value** of **i** is **7, but** the **final counter value** is **6 ,** so the **loop** will **stopp** in this case.

So we will have on the screen : 1.3.5

Exercise 01: Calculation exercise

Calculate and display the averages of 5 students, knowing that they each have two exam scores

Solution

How many variables I have:

- Two exams: 2 variables
- Plus, the average: 1 variable
- 5 students: so I need a **loop (a counter)**

So, I have **4 variables.**

Algorithm average

Var i: integer;

score1, score2, moy: real;

Begin

for (i=1, to 5 , step 1)do

write("Give the two exam scores for student ", i);

read(score1,score2);

moy←(score1+score2)/2;

write("The average of student ", i, " is ", moy);

endfor

end.

The counter step can be incremented/decremented in the following cases:

1. For (i= initial value, to final value**, the step value**) do
2. For (i= initial value, to final value) do

   Instructions

   i= i+ number /*i = i - number */
3. For (i= initial value, to final value) do

   Instructions

   i++ ; /* (means plus 1); */ /*i - -;(means minus 1);

Exercise 02: calculate the sum from **one to six:**

Algorithm Sum

Var som,i: integer

Begin

Som ← 0 ; /* initialization of the sum variable */ ;

for (i= 1 to 6) do

som ← som+i ;

End for ;

write (“the sum is”, som);

End.

1.3 The “for “ loop in C language:

Syntax

```
for (initialize; Condition; increment)
{
   /* Instructions to repeat */
}
  Rest  of program
```

Execution sequence:

1. Execute the initialization (only once).
2. Test the condition.

- (If it is **true** then)

  2.1 Execute the loop body (instructions to repeat).

  2.2 Execute the increment, then go back to step 2.

- If it is **false** : stop the loop and continue with the rest of the program.

Example 01 : Display hello 10 times

```
# include  <stdio.h>
int main()
{
int i ;
 for (i=1 ; i <= 10 ; i++)
       {  /* if there is only one statement in the body of the loop the { } is not required*/
         printf("hello \n" ) ;
       }
 return 0 ;
}
```

Example 02: Sum of integers between two numbers

Calculate and display the sum of all numbers between two integer numbers entered by the user

Solution :

```
#include <stdio.h>
int main()
{    int i, a, b, temp, sum = 0;
     printf("Enter two integer numbers:\n");
     scanf("%d%d", &a, &b);
```

```
        // Swap if a > b
        if (a > b) {
            temp = a;
            a = b;
            b = temp;
        }
        // Calculate the sum
        for (i = a; i <= b; i++) {
            sum += i;
        }
        printf("The sum between %d and %d is %d\n", a, b, sum);
        return 0;
    }
```

Exercise 03 : display all odd numbers between two numbers entered by the user.

```
# include  <stdio.h>
int main()
{  int  i , a,b,c;
printf("give two integer numbers  \n");
scanf("%d%d",&a,&b);
// swap if a > b
if(a>b)
{c=a ;
a=b ;
b=c ;
}
 for (i=a ; i <= b ; i++)
       { if(i%2!=0)
         printf("%d est odd \n",i) ;
       }
  return 0 ;
}
```

Execution

give two integer numbers

3

6

For i=3, is 3<= 6; yes , so we execute the statement :

Is 3%2!=0, yes , print (3). And increment the counter by 1 , so i=3+1=4

I=4,Is 4 <= 6, yes . so we execute the statement :

Is 4%2!=0, no , so there is no display increment the counter by 1 , so i=4+1=5

I=5, Is 5 <= 6, yes . so we execute the statement :

Is 5%2!=0, yes , print (5). And increment the counter by 1 , so i=5+1=6.

I=6, Is 6 <= 6, yes . so we execute the statement :

Is 6%2!=0, no , so there is no display increment the counter by 1 , so i=6+1=7

I=7 ; is 7<=6 , no , so , the loop is stopped.

2. The repeat….until loop

2.1 Algorithmic Syntax:

Repeat

<Instruction block>

Change of stop condition value

Until < stop condition >

Explanation: This type of loop allows you to repeat the <Instruction block> one or more times and stop when a condition is met. When the condition becomes true, the loop ends; otherwise, the <Instruction block> is executed again.

Remarks

1. In this loop, processing is executed at least once before the stopping condition is cheked.
2. There must be at least one **action** between **Repeat** and **Until** that **changes the value of the stopping condition.**
3. **Steps for Executing the Repeat Loop**

1) Execution of the <Instruction block>.
2) Update the value of the condition
3) Test the <stop condition>
    - If it is true, the loop stops.
    - If it is false return to Step 1.

2.2 When to Use This Loop:

The **Repeat…Until** loop is often preferred when:

- User input must be filtered,
- Or when it is necessary to do the work at least one time before verifying the condition.

Exercise 04: write an algorithm that calculates the sum of integers entered by the user.

```
Algorithm  example1
Var  s, nbr : integer ;
       ans : booléen ;
begin
s  ←0 ;
repeat
    write (‘’ give an integer number  ’’) ;
read (n) ;
s ← s+ n ;
write (“do you want to add another number true or false?’’( true or false)”);
read (ans) ;
until (ans = false) ;
write (‘’ the sum is  :’’, s) ;
end.
```

Execution sequence

S=0;

Loop execution:

enter a number: 6↵ s= s+n = 0+6 =6

Message 'do you want to add another number true or false, true

ans = true

Test the stop condition. Is ans = false no, so execute the instruction after repeat

enter a number: 10↵ s= s+n = 6+10 =16

Message 'do you want to add another number true or false, true

ans = true

Test the stop condition. Is ans = false no, so repaet

enter a number: 15↵ s= s+n = 16+15 =31

Message 'do you want to add another number true or false, false

ans = false

Test the stop condition. Is ans = false yes, so stop

Message: The sum is: 31.

Exercise 05:

Write an algorithm that takes an even number and determines how many times it is divisible by 2.

Example 8 is divisible 3 times by 2 (2*2*2).

Solution

Algorithm even

Var nb, count : integer ;

Begin

repeat

Write("Enter an even number: ");

read (nb);

until (nb mod 2 =0);

count ← 0 ;

repaeat

nb ← nb div 2

count ← count +1 ;

until (nb = 1)

Write("The number is divisible ", count, " times by 2.");

End

```
#include <stdio.h>
 main()
{
int nb,count ;
do
   {
   printf("give an even number \n ");
   scanf ("%d",&nb);
    }
 while (nb%2!=0) ;
 count =0;
 do
 {     nb=nb/2;
     count=count+1;
 }while (nb!=1);
 printf("the number is divisible  %d  times
by 2\n",count);
}
```

Process sequences :

Give an even number 7 (not even number, ask again)

Give an even number 8

Count =0; nb=8/2=4

Count =0+1; is nb =1 , no so continue the loop

nb=4/2=2; count =count +1=2, is nb =1 , no, continue the loop

nb=nb/2=1, count =2+1=3, test condition , is nb=1 , yes , so stop the loop

final message :The number is divisible 3 times by 2 .

2.3 The repeat…. until loop in C : do …..while

Syntax :

do

{

<**instructions** >

Change the stop condition value

}while (stop condition)

Exercise 06: entering a number between 1 and 10

Solution

include <stdio.h>

}int main()

```
{
int nb ;
do
  {
  printf("Give an integer number between 1 and 10: ");
  scanf (‘’%d’’, &nb) ;
    }
 while (nb < = 1 || nb >= 10) ;
  printf("correct\n") ;
return 0 ;
}
```

3. The while loop

3.1 Algorithmic Syntax

WHILE (execution condition) DO

<Processing>

// update the condition value

END WHILE

While Loop Explanation

This iteration structure allows the <Processing> to be repeated **zero or more times** and stops when the execution condition is no longer true.

- If the condition is true, <Processing> is executed.
- If the condition is false, the loop stops immediately.

3.2 Steps for executing the while loop

1) Test the value of the <execution condition>
2) If it is true, Then
    - Execution of <Processing>
    - Return to step 1.
3) Otherwise Stop the loop.

Remarks

1. In this loop, **<Processing>** may not be executed at all, if the condition is false from the beginning.
2. The condition parameters must be initialized (by input or assignment) **before** entering the loop.
3. There must be at least one action inside **<Processing>** that modifies the condition value.

3.3 Exercises

Exercise 07: Write an algorithm that takes an even number and determines how many times it is divisible by 2. Example 8 is divisible 3 times by 2 (2*2*2).

Algorithme even

Var nb, count : integer ;

Begin

count ←0 ;

write (‘’ give an even number \n ‘’) ;

read (nb) ;

// First loop: make sure the number is even

while (nb mod 2 <> 0)do

write(“give an even number”);

endwhile

// Second loop: count how many times divisible by 2

while (nb <>1) **do**

count ← count+1 ;

nb ← nb div 2

endwhile

write (‘’ the number is divisible ‘’, count, ‘times) ;

end .

Execution sequences:

Give an even number: 3

Test first loop condition : 3 mod 2 <> 0 → true → ask again.

Give an even number :8

Test first loop condition (8 mod 2 <>0)→ false → exit the first loop

Test second loop condition 8<>1, true, -so execute second loop

count=0+1 =1:

nb = 8 div 2 =4

Test loop condition 4 <>1, → true → execute loop

count=1+1 =2:

nb = 4 div 2 =2

Test loop condition 2<>1 → true → execute loop

count=2+1 =3:

nb = 2 div 2 =1

Test loop condition 1 <>, → false → stop loop

Final message: “The number is divisible 3 times by 2”;

3.4 The while loop in C language:

Syntax:

```
While (condition)
{
  Instruction block
  // chang the condition value
}
```

Process

1. Test the condition (it must **have an initial value)**
2. If it is false stop the loop. Otherwise
3. Execute the instruction block
4. Change the condition value, and go to step 1.

Exercise 12:

Write an algorithm that asks the user for a score between 0 and 20 until the answer is valid.

```
# include  <stdio.h>
int main()
{   float note ;
printf ("Enter a score between 0 and 20:\n") ;
scanf ("%f" ,&note) ;
while (note < 0 || note >20 )
{
printf ("Invalid input. Enter a score between 0 and 20:\n");
scanf ("%f" ,&note) ;
}
printf ("Valid input. The score is %.2f\n", note);
return 0 ;
}
```

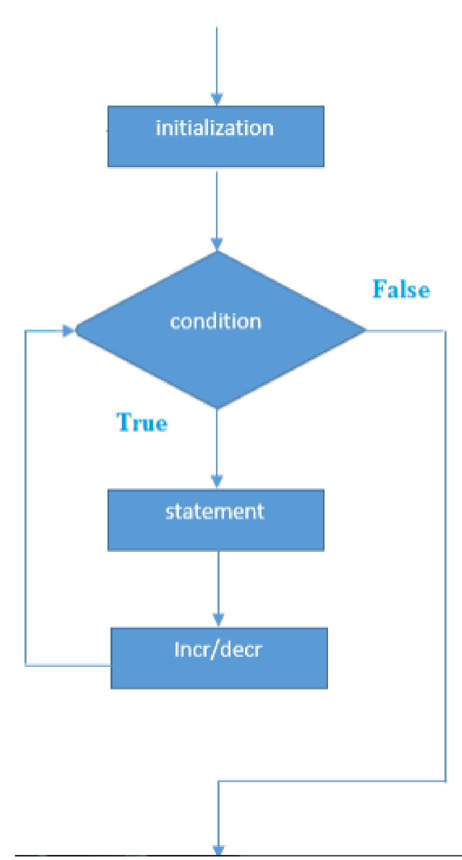
Exercise 13:

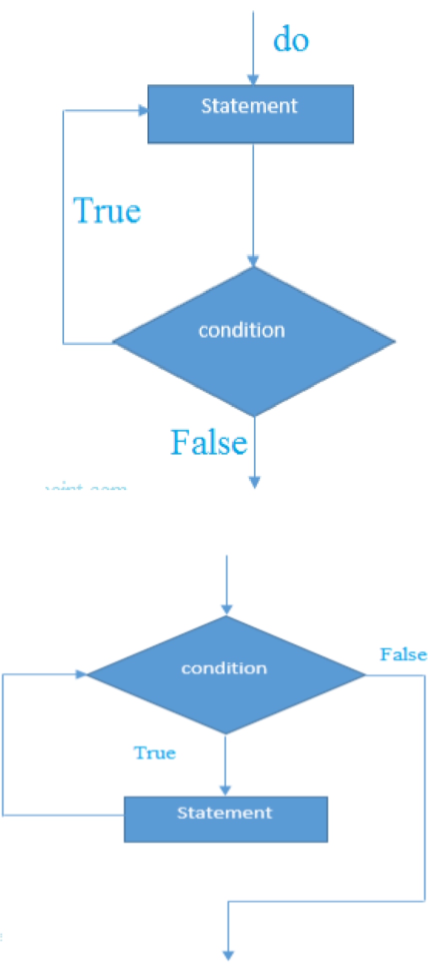
```
# include  <stdio.h>
int main()
{   int count=0,n;
printf("give an even number \n ");
```

```
scanf("%d",&n);
while(n%2!=0)
      { printf(" error give an even number \n ");
      scanf("%d",&n);


      }
printf("it is correct \n");
while(n!=1)
     {n=n/2;
      count=count+1;
        }
    printf(" this number is divisible  %d times by 2 \n",count);
                                        return 0};
```

3.5 The flowchart for each loop





3.6 Exercises

Exercise 14 : Write an algorithm that requires a starting umber, and calculates and displays the sum of its strict divisors . For example, if you enter 10, the program must calculate and display 8 (because 8 = 1 + 2 + 5 where 1, 2 and 5 are the strict divisors of 10).

Solution

Algorithme divisor

Var x,s ,i : integer ;

begin

write (‘give an integer number n ‘’) ;

read (x) ;

for (i = 1 to x div 2) do

if (x mod i = 0) then

s ← s+i

endif,

end for

write ("The sum of the divisors is ", s);

end.

Exercise 15: (Tutorial N°4) :

What is displayed on the screen after running the following C program when n=5.

```
#include <stdio.h>
main() {
        int n, i, j;
        printf("enter a number of lines !\n");
        scanf("%d",&n);
        for(i=1; i<=n; i++) {
                    for(j=i;j<=n;j++) printf("*");
        printf("\n");   }
}
```

Exercise 16: (Tutorial N° 4):

Write an algorithm that takes an integer as an input, and then displays the next ten numbers. For example, if the user enters the number 12, the program will display the numbers 13 to 22.

Algorithm exo02

Var n,i: integer ;

Begin

rrite ("Give an integer number: \n ");

read(n);

for(i from 1 to 10 step 1)do

write (n,’’+”,i,”=”,n+i);

end for;

end

Exercise 17: (Tutorial N° 4):

Write an algorithm that ensures that the number entered on the keyboard is positive. If it is negative, you must re-enter another number until the answer is valid.

Algorithm exo03

Var n : integer;

Begin

repeat

write (“give a positive number \n);

read(n);

until (n>0);

end

Exercise 18: (Tutorial N° 4):

Write an algorithm that requires a number, and calculates its factorial. The factorial of 8 (noted 8!) is given by 1 x 2 x 3 x 4 x 5 x 6 x 7 x 8

There is no factorial for a negative number

Algorithm factorial

Var n ,i,f: integer;

Begin

repeat

write(“give a positive number \n”);

read (n);

until(n>=0)

f ← 1;

for (i from 1 to n step 1) do

f= f*i;

end for

write (“ the factorial of the number “, n, “is”,f);

end.

Exercise 19: (Tutorial N° 4) :

When Fatima was born, her grandfather Mohamed opened a bank account for her. Then, on each of her birthdays, her grandfather deposited 1000 AD plus double her age. into her account. For example,

At two (0) years old→1000 AD.

At two (1) years old→1000 AD+1002 AD

At two (2) years old→1000 AD+1002 AD +1004 AD.

Write an algorithm that determines how much money Fatima will have on her n-th birthday. **Algorithm Exo19-1**

Var i,age,count: integer;

Begin

repeat

write (“ give the age of Fatima \n”);

read (age);

until (age >0);

count←1000;

for(i from 1 to age step 1) do

count ←count +1000+2*i;

end for

write ("On her ", age, "th birthday, Fatima's account has ", count, " AD");

End.

3.7 Transform the for loop into a while….do loop:

Algorithm Exo19-2

Var i,age,count: integer;

Begin

Repeat

write (“ give the age of Fatima \n”);

read (age);

until (age>0);

i←1;

count←1000;

while (i<=age) do

count ←count +1000+2*i;

i←i+1;

end while

write ("On her ", age, "th birthday, Fatima's account has ", count, " AD"); end.

3.8 Transform the for loop into a repeat ….until loop:

Algorithm Exo19-3

Var i,age,count: integer;

Begin

Repeat

write (“ give the age of Fatima \n”);

read (age);

until (age>0);

count←1000;

i←1;

repeat

count ←count +1000+2*i;

i←i+1;

until (i>age) do

write ("On her ", age, "th birthday, Fatima's account has ", count, " AD");

End.

3.9 How to Convert the for loop into while loop and do while loop in C :

Exercise 20 : exercise Tutorial N° 4

```
#include <stdio.h>
main() {
int n, i, j;
// Input validation
do
{ printf("enter a number of lines !\n");
scanf("%d",&n);
}while (n<0);
for(i=1; i<=n; i++) {
for(j=i;j<=n;j++) printf("*");
printf("\n"); } }
```

Converted to while loop

```
#include <stdio.h>
main()
```

```
{  int n, i, j;
printf("enter a number of lines !\n");
scanf("%d",&n);
while (n<0)
{     printf(" error enter a number of lines !\n");
   scanf("%d",&n);  }
i=1;
while (i<=n)
 { j=i;
 while (j<=n)
     {   printf("*");
       j++;
      }
printf("\n");
i++ ; } }
```

converted to do while loop:

```
do
{ printf("enter a number of lines !\n");
scanf("%d",&n);
} while (n<0);
i=1;
do
 { j=i;
 do
     {   printf("*");
      j++;
      } while (j<=n);
     printf("\n");
      i++ ;
      }while (i<=n); }
```

Chapter 05: Arrays and character strings

Introduction:

Let us return to exercise 03 which involved calculating the average of 5 students using repetitive structures:

Algorithm average

Var i:integer;

score1, score2,moy:real;

Begin

for (i=1, to 5 , step 1)do

write (“give the two exam scores for the student”,i);

read(score1,score2);

moy←(score1+score2)/2;

write (“the average of student “,i,”is “,moy);

endfor

End.

In this example, we declared a single **variable moy** which stores the **new average value** at **each iteration**. If we want **to save all the averages** for later use, we would need to declare **5 variables**, but if we have **300 students**, this solution would not be practical.

So we use **another solution,** which allows us to **group multiple values into one variable: Arrays**

1. Definition:

Is a **complex type** unlike the other types seen previously

• The array is a **storage space** of **limited and** a fixed **size reserved in advance , this** size cannot be changed during program execution. It contains **several boxes (elements)**

• Each **box** has a **value** and a **position** called **index**, the **first index** is always equal to **0**

• The **type of the array** is the **type of its elements**

• An array **has a predefined size (capacity**) which is the **number of boxes** it can hold.

• An array cannot be completely filled but it can never contain more elements than the number provided during the declaration.

Arrays Types: There are 3 types

• One-dimensional arrays are called vectors

• A two-dimensional arrays are called a matrices

• Multidimensional array (up to 7 dimensions in Fortran).

For example:

if we want to calculate the average of 5 students, we only need one array variable of size 5:

Tab[5] : array of real

Tab : array **name**, **5** array **capacity, real** type of array values, all **5** values are **real.**

2. One-dimensional arrays (vectors)

Let T be an array of 10 elements:

T[10] : array of integer

T(1)	T(2)								T(10)

T(1) , T(2),…..T(10) : are elements of the same type

1,2,…,10 are the indices (positions) of these elements.

Let us take this example: consider T is array of 5 boxes that contains the names of students:

T[5] : array of string;

Fatima	Mohamed	Ali	Khadija	Ahmed

- T → array name 5 → array capacity (it can store 5 names)
- string (or char* in C) → type of elements (each element is a student's name).

Thus:

- T[1] → name of the first student
- T[2] → name of the second student
- …
- T[5] → name of the fifth student

Example with indices and names

- The box with index 0 contains the name Fatima → so Fatima is in the 1st position.
- The name Ahmed is in the 5th position, so the index of this box is 4.
- If we write T[3] = ???, then T[3] = Khadija.

2.1 Declaration syntax in algorithm

```
type
  array-identifier = array[size] of element-type;
var
  identifier: array-identifier
```

Example:

type

tab1 =array [0..10] of real; // array of 11 real numbers (index 0 to 10)

tab2 = array [0..25] of chain of characters; // array of 26 strings (chain of characters)

var T1,T2 : tab1;

T3:tab2;

X: real;

2.2 Assigning values to array elements:

Syntax:

Array-identifier [position]← expression ; // (where expression can be (value , variable, operation)

For exemple :

T1[0]←12.5;

T1[5]← x;

T1[9]← x+10 ;

T3[5]←Fatima ;

Or, initialization in one line:

T1={12.5,5 ;12,55,………} ; **//we must write all values.**

2.3 Input /Output syntax:

- **Read (T1[i]);**
- **Write(T1[i]);**

2.4 Exercises

Exercise 01: write an algorithm that fills and displays 10 elements of an array with 0.

Algorithme zero

Type tab = array [1..10] of integer ;

Var T: tab;

i: integer ;

Begin

/* Fill the array with zeros */

For (i =1 to 10 step 1) do

T[i] ← 0 ;

End for

// display the array

For (i =1 to 10 step 1) do

write (T[i]);

End for

End

Exercise N° 2 : Consider an array of 10 integers. (let us fill and display its elements)

Algorithm read-and-display ;

Type tab = array [1..10] of integer;

Var T: tab ;

i: integer;

Begin

write (“give 10 integer number \n”) ;

for (i = 1 to 10 step 1) do

read (T [i]) ;

end for ;

/* display the array */

For (i = 1 to 10 step 1) do

write (T [i]) ;

End for

End.

Exercise 03: display an array that contains the first 5 integers

Algorithm display

Type tab = array [1..5] of integer;

Var T : tab ;

i : integer ;

Begin

for (i =0 to 4 step 1) do

T[i] ← i ; // assignment instruction

write (T[i]) ;

end for;

Fin

Execution Sequences: T[0] = 0

- T[1] = 1
- T[2] = 2
- T[3] = 3
- T[4] = 4

Output on screen:

0 1 2 3 4

Exercise 04: calculate and display the average of 5 students.

Algorithm average ;

Type average = array [1..5] of real ;

Var note1, note2 : reel ;

i : integer ;

Moy : average

Begin

for (i= 0 to 4 step 1) do

write (" give the two markes of the student number",i," \n'') ;

read (note1, note2) ;

Moy [i] ← (note1+note2)/2 ;

end for

// Display the average

for (i= 0 to 4 step 1) do

write ('' the average of the student '', i, is '', **moy[i])** ;

end for ;

end

2.5 Declaring an Array in C

Syntax: **Data type identifier [size];**

- data_type: type of data (int, float, char, double, …)
- identifier: name of the array
- size: number of elements in the array (fixed size)

Examples:

int tab[10] : tab is an array contains 10 integer elements

char T[26] : T is an array contains 26 characters elements

float T3[15] : T3 is an array contains 15 real elements .

Array initialization syntax:

Data type identifier [length] = { , , , …..,}

Example

int tab[3] = {5.89,40}

Or

int tab [ ] = { 5,89, 40} /* in this case the compiler will detect the size of the array from the number of elements in the list.

int tab[3] = { [2] = 3 }; /* initialization of the 3rd element .

2.6 declaration and initialization :

int T1 [5] = { 5,7,18,365,95} ; Full initialization

5	7	18	365	95

int T2 [5] = { } ; Empty initialization (0 by default)

0	0	0	0	0

int T3 [5] = { 5,7} ;

Partial initialization

5	7	0	0	0

int T4 [] = { 5,7,18,365,95}

5	7	18	365	95

int T1 [5] = { 5,7,18,365,95 ,254} Error

5	7	18	365	95	254

The number of elements must not exceed the declared size.

2.7 Assigning a Value to an Array Element in C:

Array-name[index]= value ;

T1[2] = 50; // assigns 50 to the 3rd element (index 2)

2.8 Input and Output of Array Elements in C

Input (reading values into an array)

scanf (‘’%d’’, & Array-name[index]) ;

Output (displaying values of array elements)

printf (‘’%d’’, Array-name [index]) ;

2.9 Exercises

Exercise05: Array basis

```
#include <stdio.h>
int main(void)
 { int tab[3] = { 1, 2, 3 };
   printf(" %p\n", tab); // display the address of the array tab
   printf(" %d\n", tab[2]); // display the content of the 3rd element
   printf(" %d\n", tab[0]); // display the content of the 1st element
   printf(" %d\n", *tab);  // display the value of the 1st element
   return 0;
}
```

Exercise 06: write a C program that allows you to enter the number of the students (less than 300), then enter and save the average of each student.

```
#include <stdio.h>
int main()
{ float avg[300];
 int n,i;
   printf("this program saves the averages of students !\n");
   do
   {   printf("give the number of student less than 300\n ");
       scanf("%d",&n);
   }while ((n<=0)||(n>=300));
    // input averages
       for(i=0;i<n;i++)
       {
         do
         { printf("give the average  of the student number %d\n",i+1);
          scanf("%f",&avg[i]);
         } while ((avg[i]<0)||(avg[i]>20));
       }
       // display  average
       for(i=0;i<n;i++)
          printf("the average of the student number %d is %.2f \n",i+1,avg[i]);
   return 0;
}
```

Exercise 07: write a c program that fills an array with all even number between 0 and 20

```
#include <stdio.h>
int main()
{ int i,j,tab[11];
      j=0;
    for(i=0;i<=20;i++)
    {   if(i%2==0)
          {   tab[j]=i;
              printf(" the content of the case %d is %d \n",j+1,tab[j]);
              j=j+1;
```

```
        }
    }
    return 0;}
```

3. Arrays operations:

3.1 The sum of array elements, traverse the array and add the selected element in each iteration

Exercise 08: write an algorithm that calculates the sum of array elements. and theirs average

```
Algorithme sum
Type sum = array  [0..9] of integer ;
Var  i, S  : integer ;
        M : réel ;
      T1:sum;
Begin
/* Array filling */
  write  (‘give   10 integers ‘’) ;
   for  (i =0 to  9 step  1 ) do
   read (T1[i]) ;
   end for;
/* calculate the  sum */
   S ← 0 ;
   For  (i =0 to  9 step  1 ) do
   S ← S+ T1[i] ;
   End for ;
   write (‘’the  sum  is  :’’, S) ;
   /* calculate the  average */
   M ← S/10 ;
   write (‘’ the  average  is  :’’, M) ;
   End.
```

3.2 The sum of the elements of two arrays: (The two arrays must have the same size and the same data type).

The sum consists of adding the elements which have the same indices: the first element of table 1 with 1st element of table 2, and so on

Exercise 09: write an algorithm that lets you enter the elements of two arrays, calculates and displays the sum of their elements, and saves the result in a third array.

```
Algorithm somme_tab
Type tab = array  [10] of reals ;
Var i : integer ;
    T1, T2, T3 : tab ;
Begin
/* filling the two arrays */
write ('give the 10 reals number for the array  1\n '') ;
for (i =0 to  9 step 1) do
read (T1[i]) ;
end for
write ('give the 10 reals number for the array 2 \n'') ;
for  (i =0 to 9 step 1 ) do
read (T2[i]) ;
end for
/* the sum of two arrays */
for  (i =0 to 9 step 1 ) do
T3[i] ← T1[i] + T2[i] ;
write (T3[i] ) ;
endfor.
End .
```

Note : we can use two arrays and the result will be saved in one of them .

3.3 Show the number of values greater than a given value

Exercise 10: Count averages greater than or equal to 10 in a class of 30 students. (the average must be a correct value)

```
Algorithme exe-10
Type tab = array  [30] of real ;
Var T1 : tab;
i, co : integer ;
Begin
  co← 0;
for  (i= 0 to  29 step 1 ) do
     repeat
       write ('' enter the average of the student number  :'',i+1) ;
       read (T1[i]);
```

```
        until (T1[i]>=0 )and (T1[i]<=20))
         if (T1[i] >=10) then
        co ← co +1 ;
       endif  ;
endfor
write (‘’ there is  :’’, co , ‘students  have greater than  10’’) ;
End .
```

3.4 Show the maximum or minimum of an array and their position (index):

Exercise11: Write an algorithm that allows you to display the maximum real value in an array of size n (here n = 100) and specify the position (index) where it is located.

```
Algorithm Exe10
    Const n =100
    Type tab = array [0..n-1) of real;
    Variable     max : rea l ;
                  i, pos : integer ;
                 P : tab
    Begin
    /* Array filling */
       for (i = 1 to n-1) do
           write ('Enter value ', i, ' : ')
           read (P[i])
       end for
       /* Initialization with the first element */
     max  ← P[0] ;
     pos ← 0 ;
    /* Search for the maximum */
     for  (i= 1 to  n-1) do
      if  (P[i] > max ) then
          pos   ← i ;
          max  ← P[i] ;
      end if  ;
     end for ;
    write (‘‘the maximum  value is = ‘', max, '' it  is  in the position ‘', pos) ;
    end.
```

3.5 Permutation of the elements of an array

This operation consists of reversing the positions of the elements: the last element becomes the first, the first becomes the last, and so on.

Exercise 12:

Write an algorithm that reverses the elements of an array of 8 elements.

Example:

Initial array:

15	9	66	85	18	55	49	74

After permutation:

74	49	55	18	85	66	9	15

Solution :

```
Algorithm  Permutation_array
Variables T : array [0..7] of integer ;
   i, x, k : integer ;
Begin
  for (i =  0 to  7) do
   write ("give the element  ", i+1, " of  array ")
   read (T[i])
  end for
  k ← 7
  // Note that I varies from 0 to 3 in the following loop:
for (i =  de 0 à 3 )do
   x  ← T[k]
   T[k]  ← T[i]
   T[i]  ← x
   k  ← k -1
  end for
  for (i =  0 to 7 )do
   write  (T[i])
  endfor
 end.
```

X= t[7] = 74 , t[7] = t[0] = 15 , t[0]= 74 ; k=6,i=1

X= t[6] = 49 , t[6] =t[1] =9, t[1] = 49 ; k =5,i=2

X=t[5] = 55 , t[5]= t[2] = 66 , t[2] = 55, k=4,i=3

X=t[4] = 18 , t[4]= t[3] = 85 , t[2] = 18, k=4,i=3

3.6 sorting arrays: this operation allows you to sort the elements of an array in an ascending or descending manner.

Exercise 13: Write an algorithm that allows you to enter 4 averages and classify them in ascending order.

```
Algorithme sorting_array
Const  N = 4
Variable Moy : array[0.. n-1] of reals
 i,j: integer;
x : real ;
Begin
// filling the array
for (i = 0 to N-1 )do
write ("enter the average ",i+1, " :")
read(Moy[i])
endfor;
// sorting the elements
for (i = 0 to  N-2) do
 for (j = i +1 to N )do
    if( Moy[i] > Moy[j] )then
     x ← Moy[i];
     Moy[i] ←Moy[j];
     Moy[j] ← x ;
    endif
    endfor
 endfor
// display the elements of array
    for (i from 0  to  N-1) do
      write (Moy[I])
   endfor
End.
```

4. Two-dimensional array:

Consider the following example: a section of 500 students who follow 10 modules, we want to calculate the average of each student.

- What is the data?
- What are the results?
- How should we declare them?

➢ If we use **500 × 10 variables** for the grades of 500 students in 10 modules, plus **500 variables** for the averages… it becomes **unreasonable.**

➢ If we use **10 vectors (one-dimensional arrays) of size 500** for the grades and another for the averages… it is also **illogical.**

• In this case, vector 1D does not respond to this reality,

We need for other structures → **Two-dimensional array(matrix)**

Two-dimensional array(matrix):

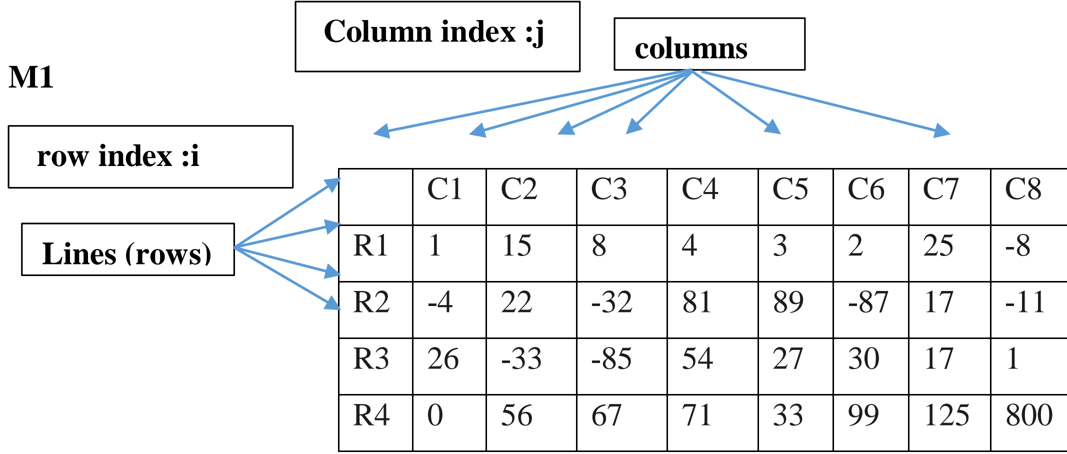
4.1 Definition:

A matrix is a set of data of the same type, stored in central memory and referenced by **two indices (rows and columns).**

A matrix is characterized by:

- ✓ The name
- ✓ The number of columns
- ✓ The number of lines (rows)
- ✓ The size of the matrix =(number of rows * number of columns)
- ✓ The type of the elements of the matrix.
- ✓ Each element in a matrix is characterized by the row index and the column index.

Here is a **matrix** that contains **4 rows and 8 columns.**



	C1	C2	C3	C4	C5	C6	C7	C8
R1	1	15	8	4	3	2	25	-8
R2	-4	22	-32	81	89	-87	17	-11
R3	26	-33	-85	54	27	30	17	1
R4	0	56	67	71	33	99	125	800

Example:

$M=\begin{pmatrix} 2 & 5 & -8 \\ -11.5 & 8.5 & 3 \\ 1 & 7.25 & 4 \end{pmatrix}$; M(0.2) = -8 , M(2.1) = 7.25.

M(i,j) = row number i, and column number j.

M1[3,7] =800;

4.2 Declaration of a 2-dimensional array:

Type mat = array [n, m] Data type

Var

M1 : mat ;

Example :

Type mat = array [10,10] integer ;

Type student = array [1..500,1..10] real ;

Var

M1, M2 : mat ;

N1,N2 : student ;

4.3 Assignment:

Syntax   **Matrix-name[index row, index column]← Expression ;**

Expression can be : a value , variable, operation

Examples:

M1[0,4]←5.5;

M1[1,2]←x;

M1[3,5]←M1[4,4];

M1[2,7]←(x+1)/2;

M1[5,5]←M1[1,1]*M1[2,2]

4.4 Input /output operations:

read(matrix-name[index rows, index column]);

write (matrix-name[index rows, index column]);

Example:

read(M1[I,j]);

write(M1[I,j]);

Exercise 14: fill an integer matrix of 10 rows and 15 columns

Type mat= array [0..9,0..14]of integer ;

var i, j: integer ;

M : mat ;

```
Begin
for( i=0 to 9) do
        for( j=0 to  14) do
                read(M[i, j]);
        endfor
end for
End
```

Exercise 15: fill and display the contents of a matrix with **n** rows and **m** columns

```
Type mat = array [1000,1500]: real;
var i, j: integer
M: mat;
Begin
write ("give the number of rows and columns \n");
read(n,m);
 for( i=1 to n )do
        for (j=1 to m) do
        read (M[i,j]);
        end for
 end for
 for( i=1 to n )do
        for (j=1 to m) do
        write(M[i,j])
        end for
 end for
END.
```

4.5 Matrix in C language :

Declaration syntax :

Type identificateur [nombre de lignes][nombre de colonnes] ;

Example :

int tab [10][15] ; a matrix of 10 lines and 15 columns of integers

float mat [100][100] ; a aquar matrix of 100 lines and 100 columns of reals

Char [50][30] ; a matrix of 10 50 and 30 columns of characters

1. Assignment :

M[i][j] = value /variable /expression

M[1][5] = 5 ; M[3][5] = x ; M[2][2] = x/y , /*with x,y are variables of same type as M*/

Exercise 15 :write a c program that filling and display an integer matrix [3,3].

```
#include<stdio.h>
int main(){
        int mat[3][3];          /* 2D array declaration*/
        int i, j;               /*Counter variables for the loop*/
    for(i=0; i<3; i++)
{     for(j=0;j<3;j++)
    {  printf("Enter value for mat[%d][%d]:", i, j);
       scanf("%d", &mat[i][j]);
     }
  }
  //Displaying array elements
  printf("Two Dimensional array elements:\n");
  for(i=0; i<3; i++)
 {  for(j=0;j<3;j++)
    {  printf("%d ", mat[i][j]);
       if(j==2)
     {  printf("\n");
       }
     }
  }
  return 0;
}
```

Initialization of 2D Array

There are two ways to initialize a two Dimensional arrays during declaration.

```
int mat[2][4] = {
   {10, 11, 12, 13},
   {14, 15, 16, 17}
};
```

OR

```
int mat[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

```
int abc[2][2] = {1, 2, 3 ,4 }      /* Valid declaration*/
int abc[][2] = {1, 2, 3 ,4 }       /* Valid declaration*/
```

```
int abc[][] = {1, 2, 3 ,4 }        /* Invalid declaration – you must specify second dimension*/
int abc[2][] = {1, 2, 3 ,4 }      /* Invalid because of the same reason  mentioned above*/
```

4.6 Matrix properties

- **The diagonal of a matrix:** The column index is equal to the row index. **M[i,i]**

Example :

$$M=\begin{pmatrix} \mathbf{2} & 5 & -8 \\ -11.5 & \mathbf{8.5} & 3 \\ 1 & 7.25 & \mathbf{4} \end{pmatrix};$$

M[1.1] = 2

M[2.2] = 8.5

M[3.3] = 4

- **The anti diagonal:**

$$M=\begin{pmatrix} 2 & 5 & \mathbf{-8} \\ -11.5 & \mathbf{8.5} & 3 \\ \mathbf{1} & 7.25 & 4 \end{pmatrix};$$

M[1.3] = -8 , M[2.2] = 8.5 , M[3.1] = 1

In general case it is : **M[i, n-1-i]**

- **The identity matrix**

$$M=\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix};$$

M[i.i] = 1 , M[i.j] = 0 , for each $i \neq j$;

- **The symmetric matrix: M[i.j] = M[j.i]**

$$M=\begin{pmatrix} 5 & 6 & -4 \\ 6 & 4 & 10 \\ -4 & 10 & -13 \end{pmatrix}$$

Exercise 16:

Write an algorithm that displays a square matrix (n*n) with diagonal values = 1;

Algorithm Matrix

type Mat = array [100,100] of integer

Var i,j: integers

M: matte

Begin

For (i = 1 to 100) do

For (j= 1 to 100) do

read (m[i,j]);

If (i=j) then

Mat[i,i]← 1 ;

End if ;

endfor

endfor

END.

C program:

```
#include<stdio.h>
int main(){
  /* 2D array declaration*/
  int mat[3][3];
  /*Counter variables for the loop*/
  int i, j;
  for(i=0; i<3; i++) {
    for(j=0;j<3;j++) {
      printf("Enter value for mat[%d][%d]:", i, j);
      scanf("%d", &mat[i][j]);
      if(i==j)
      {
        mat[i][j]=1;
      }
    }
  }
  //Displaying array elements
  printf("Two Dimensional array elements:\n");
  for(i=0; i<3; i++) {
    for(j=0;j<3;j++) {
      printf("%d ", mat[i][j]);
      if(j==2){
        printf("\n");
      }
    }
  }
  return 0;
}
```

Exercise 17: calculate the sum of each row and the sum of each column for a matrix of n row and m column.

Algorithm Sum

```
Type tab = array [n,m] of integer
Type tab2 = array [n] of integer
Type tab3 = array [m] of integer
Var i,j: integers
        M:tab; sl: tab2; sc: tab3;
Begin
for (i = 1 to n) do
   for (j= 1 to m) do
    read (m[i,j]);
   End for
End for
/* sum of each line */
For (i = 0 to n-1) do
  Sl[i] ←0 ;
   For (j= 0 to m-1) do
     Sl[i] ← Sl[i]+m[i,j]
   End for
   write (sl[i]);
End for
/*sum of each column*/
For (j = 0 to m-1) do
  Sc[j] ←0 ;
   For (i= 0 to n-1) do
     Sc[j] ← Sc[j]+m[i,j]
   End for
  write (sc[j]);
End for
END
```

4.7 Matrix Operations

Exercise 18: Program to Add Two Matrices

```
#include <stdio.h>
int main() {
  int r, c, a[100][100], b[100][100], sum[100][100], i, j;
  printf("Enter the number of rows (between 1 and 100): ");
```

```
scanf("%d", &r);
printf("Enter the number of columns (between 1 and 100): ");
scanf("%d", &c);

printf("\nEnter elements of 1st matrix:\n");
for (i = 0; i < r; ++i)
  for (j = 0; j < c; ++j) {
    printf("Enter element a%d%d: ", i + 1, j + 1);
    scanf("%d", &a[i][j]);
  }

printf("Enter elements of 2nd matrix:\n");
for (i = 0; i < r; ++i)
  for (j = 0; j < c; ++j) {
    printf("Enter element b%d%d: ", i + 1, j + 1);
    scanf("%d", &b[i][j]);
  }
// adding two matrices
for (i = 0; i < r; ++i)
  for (j = 0; j < c; ++j) {
    sum[i][j] = a[i][j] + b[i][j];
  }
// printing the result
printf("\nSum of two matrices: \n");
for (i = 0; i < r; ++i)
  for (j = 0; j < c; ++j) {
    printf("%d   ", sum[i][j]);
    if (j == c - 1) {
      printf("\n\n");
    }
  }
return 0; }
```

Exercise 19: product of a vector with a matrix

Type tab = array [n,m] of integer

Type tab2 = array[m]integer;

Var i,j: integers

M: tab; T,P: tab2

Begin

For (i = 1 to n) do

Read (T[i]);

For (j= 1 to m) do

Read (M [i,j]);

End pour

End pour

/* product of each vector element by all elements of row i and sum */

For (i = 1 to n) do

P[i] ← 0 ;

For (j= 1 to m) do

P[i]← M[i,j]*T[j] ;

End for

write (P[i]);

End for

End

5. Character strings

5.1 Definition:

A character string is a sequence of characters that ends with the null character **('\0'),** a character string with a length which is the number of characters it contains.

Declaration syntax:

First way: **var variable_name: chaine [length];**

Second way: **var variable_name: c.c;**

Example

Var name: char [25] **or** **var** **name: c.c;**

In C language:

There is no special type for character strings in C language. A character string is declared as a one-dimensional array of char (character vector) whose end is indicated by the character **'\ 0'**. The size of the string is equal to the maximum length of the string plus one so that we can store the character **'\0'** denoting the end.

char variable-name [length] ;

Example :

Char nom[25] ; // it contains 24 characters + one of : '\0'.

Access to the elements of a character string:

Example: For the word string **"hello",** we have: char Mot[6]="hello"

Mot [0]← 'h' ; Mot [1]← 'e' ; Mot [2]← 'l' ; Mot [3]← 'l' ; Mot [4]← 'o' ;

Address	Content
0000	'h'
0001	'e'
0002	'l'
0003	'l'
0004	'o'
0005	'\0'

Reading and display

Algorithm, a character string is read and displayed globally (at once) and not character by character.

Example :

Algorithm

read (word);

write (word);

In C language:

we can read and display a character string in two ways:

First way: **scanf ("%s", word);** **printf ("%s", word);**

Second way: **gets (word);** **puts (word);**

The difference between scanf and gets is that **s**

canf :only brings the text entered before the first blank (space) into the variable to read.

gets :brings all the text entered up to the carriage return (newline \n) into the variable to read.

```
#include <stdio.h>
main ()
{char nom [50] ;
printf ("give your  first name and last name \n"  ) ;
//gets (nom) ; //  mohammed ali
//printf ("%s \n",nom);  // mohammed ali
scanf ("%s\n",nom) ;  //  mohammed ali
//printf ("%s \n",nom);  // mohammed
```

```
printf("\n");
// but if we use gets
gets (nom) ; //  mohammed ali
printf ("\n%s \n",nom);  // mohammed ali
//Example de puts et printf
 int a=5 ;
printf ("the value of a  is : %d\n", a) ;//  'la valeur de a est : 5
//puts ("the value of  a is : %d\n", a) ;//  error: too many arguments to function 'puts'|
puts ("the value of a is \n"); ; // the value of a is .
}
```

Initialization of a character string

To initialize a string, we declare the string variable and assign its initial value.

Example :

Algorithm:

Var word [15] : c.c ;

word ← ''Hello'; // initialize the word variable to the c.c ''Hello''.

In C language:

char word[] = { 'b','o','n','j','o','u','r', '\0'}; /* fair, but to be avoided!! */

char word[] = 'hello'; // the computer automatically reserves the number of bytes needed for the string, i.e.: the number of characters + 1 (here: 8 bytes).

char word[10] = 'hello'; // explicit indication of the number of bytes to reserve, the number of bytes must be greater than or equal to the length of the string.

char word[10]= { 'b','o','n','j','o','u','r'};

Functions used in C language to manipulate the string:

Function	In algorithm	In C	Example
Longueur de la chaine	Long(ch)	strlen(**char** *chaine);	L ← long ('Bonne chance') ; // L=12
Position de chaine dans une autre	Pos (ch1, ch2)	Strchr (ch1, ch)	P ← pos ('gra', 'Programmation') ; // P= 4
Edit an Sting characters	Copy n characters (ch1,ch2,n)	strncpy (ch,ch2,n)	CH1 ← ('Bonjour', mohamed, 4) ; // CH2 = mohamed.

			Ch1='Bonjourmoha
La concaténation entre les chaines	Concat (ch1,ch2,..)	**char** *strcat(**char** *destination, **char** *source);	CH1 ← 'Juillet' ; CH2 ← Concat ('5', '/', CH1, '/', '1962') ; CH2 ←5/juillet/1962
copies the contents of “chaine2” into “chaine1”.	Copy(ch1,ch2)	**char** *strcpy(**char** *chaine1, **char** *chaine2);	Chaine1= ‘’bonjour’’ Chaine2=tous le monde Chaine1=bonjour tous le monde

Exercise 20: display the length of a given string

```
#include <stdio.h>
#include <string.h>
int main(void)
{   printf("length : %zu\n", strlen("hello")); // displays length 5
    return 0;
}
```

Exercise 21: copy s string into another

```
#include <stdio.h>
#include <string.h>
int main(void)
{  char chaine[25] = "hello\n";
   strcpy(chaine, "thank you");
   printf("%s\n", chaine);  //   thank you
   return 0;
}
```

Exercise 22: concatenation of two string

```
#include <stdio.h>
#include <string.h>
int main(void)
{     char chaine[25] = "hello";
   strcat(chaine, " every one");
   printf("%s\n", chaine);  // helloevery one
   return 0;
}
```

Exercise 23: extract a substring

```
#include <stdio.h>
#include <string.h>
 main(void)
{     char chaine[25] = "Bonjour"; char cch[25] ;
    printf  (" the substring of %s is %s ", chaine, strncpy(cch,chaine1,4);  // Bonj
    }
```

Exercise 24: Comparison of two character strings

0 if equal

-1 first char of chaine1 less than the 1st char of chaine2

1 if 1st char of ch1 greater than the 1st char of ch2.

```
#include <stdio.h>
#include <string.h>
 main(void)
{     char ch1[25] = "Bonjour";
       char ch2[25]= "Bonjour";
       char ch3[25]= "hello";
printf((" the comparison between  %s and %s  is %d ", ch1,ch2, strcmp(ch1,ch2));//0
printf((" the comparison between  %s and %s  is %d ", ch1,ch3, strcmp(ch1,ch3));//-1
    }
```

Exercise 24: the revers of character string

```
#include <stdio.h>
#include <string.h>
 main(void)
{     char ch1[25] = "Bonjour";
printf("the reverse of the string is %s\n ", strrev(ch1));//ruojnoB
}
```

Exercise 25:

```
gets(chaine1) ; //my name is
     scanf("%s",chaine2) ;//Mohamed
     printf("%s\n",strncat(chaine1,chaine2,3)) ;//chaine1+3 = my name isMoh
     printf(strcpy(chaine1,chaine2)) ;//Mohamed.
```

Exercise 26:

```
#include <stdio.h>
#include <string.h>
main(){
char chaine[25];
int i,nbr=0;
printf(" give a sentences \n");
gets(chaine);
puts(chaine);
i=0;
while(chaine[i]!='\0')
 if ((chaine[i]=='a')||(chaine[i]=='A')||(chaine[i]=='i')
||(chaine[i]=='I')||(chaine[i]=='e')||(chaine[i]=='E')
||(chaine[i]=='o')||(chaine[i]=='O')||(chaine[i]=='u')||(chaine[i]=='U')||(chaine[i]=='y')||(chaine[i]=='Y'
))
        nbr=nbr+1;
   i++;
}
printf("the number of vowel of the sentence %s is %d",chaine,nbr );
}
```

Exercise 27: count the number of word

```
  #include <stdio.h>
#include <string.h>
main(){
char ch2[50];
int i,x,nbrw=0;
printf(" give a sentences \n");
gets(ch2);
 x=strlen(ch2);
for(i=0;i<=x;i++)
{  if(ch2[i]==' ')
   nbrw++;
}
printf("the number of word of the sentence %s is %d\n",ch2,nbrw+1 ); }
```

Exercises about arrays and Character String

Exercise 01 : what is displayed the follwing program ?

```
#include <string.h>
main ()
{
char ch1[50]=''bonjour'' ;
char ch2[ ] ='' monsieur";
printf ("befor : %s\n ", ch1);
strcat (ch1, ch2);
printf ("after : %s\n'',ch 1);
strncat (ch1,ch2,2);
printf ("after : %s\n'',ch 1);
}
```

With strncat(char s[50],char t [50], int n) add the first n characteres of the string <t> to the end of the string <s>

Solution :

Befor : bonjour

After : bonjourmonsieur

After : bonjourmonsieurmo

Exercise 02: Write a program that reads a word, then displays the reverse of this word.

For example:

- Input: SALUT
- Output: TULAS

Solution

```
#include <stdio.h>
#include <string.h>
int main() {
   char word[100];
   int i;
   // Read a word
   printf("Enter a word: ");
   scanf("%s", word);
      // Display reversed word
   printf("The reversed word is: ");
   for (i = strlen(word)- 1; i >= 0; i--) {
      printf("%c", word[i]);
   }
   printf("\n");
   return 0;
}
```

Exercise 03: Write a program in C that converts a positive decimal number into its binary representation.

For example:

- Input: N = 8
- Output: 1000

Algorithme binary

```
Var   nbr , i, k : integer
T : array [100] of integer ;
begin
write ('' enter a positive decimal number);
read (nbr);
i  ← 1 ;
repeat
T[i]   ←  nbr mod 2 ;
nbr  ← nbr div 2;
i     ←   i+1 ;
until (nbr=0);
write ('' The binary conversion of '',nbr,''is:) ;
for (k = i to 1 step (-1)) o
write (t[k]) ;
end for
end
```

in C :

```
#include < stdio.h>
int main ( )
{ int i,k,nbr, t[100];
printf('enter a positive decimal number \n'') ;
scanf  (''%d'',&nbr) ;
 i= 1 ;
do
  { nbr  = nbr div 2
    t[i]   =  nbr mod 2 ;
      i   =   i+1 ;
}
  while (nbr != 0)
printf ('' (''The binary conversion of %d is \n '', nbr '') ;
for  (k = i ; k=1 ;k--)
{ printf (''%d'', t[k]) ;
}
return 0 ;  }
```

Exercise 04: Hamming Distance

The Hamming distance between two words of the same length is the number of positions where the letters are different.

Example:

JAPON – SAVON → distance = 1

Task:

Write a function that calculates the Hamming distance between two words entered by the user.

Solution

```
#include < stdio.h>
int main ( )
{  char ch1 , ch2 [10];
   int i,d ;
   printf (‘’ enter two strings \n’’ ) ;
   gets (ch1, ch2) ;
    if (strlen(ch1)== strlen(ch2)
     {    for (i = 0 ; i<= strlen(ch1); i++)
            {if ch1[i] != ch2[i]
               {
               d= d+1;
               }
            }
        }  else
        { printf (‘’ The two strings have different lengths.’’);
         }
   printf (‘’ the  distance between the two strings is %d\n’’,d) ;
return 0 ; }
```

Exercise 05

Write in C language a program that reads a real number, then displays its integer part and its fractional part.

Solution:

```
#include <stdio.h>
#include <math.h>
int main() {
   double number, fractional;
   int integer;
   printf("Enter a real number: ");
   scanf("%lf", &number);
   integer = (int)number;                // integer part
   fractional = number - integer;         // fractional part
   printf("The integer part is: %d\n", integer);
   printf("The fractional part is: %lf\n", fractional);
   return 0;
}
```

Exercise 06 :

The following recurrence formula makes it possible to calculate the square root of “A”:

Write in C language a program that asks the user for the value of “A” and the number of iterations, then calculates and displays the *n*th term of U.

$$\begin{cases} U_1 = 1 \\ U_i = \dfrac{\left(U_{i-1} + \dfrac{A}{U_{i-1}}\right)}{2} \end{cases}$$

```
Algorithm Equation

Variables:
A, i, u, n : integer
Begin
  Write ("Enter an integer value:");
  Read (A);
  Write ("Enter the number of iterations:");
  Read (n);
  u ← 1;
  For i ← 2 to n do
    u ← (u + (A / u)) / 2;
  EndFor
  Write ("The final value of u is: ", u);
End.
```

Chapter 06: Records and structs

Introduction

In the previous chapter, we studied **arrays**, which allow us to store and process a **collection of elements of the same type**. Arrays are very useful, but sometimes they are not enough.

In many real situations, we need to group together different kinds of data that belong to the same entity. For example, to represent a **student**, we may need a **name (string**), an **age (integer**), and a **grade (real number).**

Using only arrays or simple variables makes this complicated. That is why we introduce a new data type: **records (called structs in C).** They allow us to **combine different types of data into a single**, well-organized unit.

1. **Definition:** **A record** is a package of variables, possibly of different types. Each variable is a field of the record. In C language, records are called structs, a shorthand for structure.

2. How to declare a record in algorithm and in C language?

In algorithm

```
Struct student =record
      Name : c.c;
      Age :integer;
      Average : real
End struct;
Var E1,E2: student
```

In C language:

```
struct {
   int day;
   int month;
   int year;
 } x;
struct dmy {
  int day;
  int month;
  int year;
};
struct dmy x;
struct dmy y;
```

Examples

```
struct Date {
int jour;
int mois;
int annee;
};
```

```
struct Etudiant {
        char fname [30] ;
        char lname [30] ;
        struct Date date_naissance ;
        };
```

3. Example for Initialization:

struct Student E1 = {"Mohammed", "Ali", {1, 1, 2000}};

4. input /output in algorithm and in C:

Algo: read(E1.fname); read (E1.lname), read(E1.Date.jour);

write (E1.fname);

C:

scanf(“%S”,E1.fname); printf(“%d”,E1.age);

5. Exercises

Exercise 1: write an algorithm and a c program that fill and display the information of an employee

```
#include <stdio.h>
main ()
{   struct employe
   { char fname[30];
     char lname[60];
     int age;
     float salary;
     };
     struct employe E1;
      printf(" filling the information of this employee \n");
      printf(" please enter the first and the last name the age and the salary \n");
      scanf("%s%s%d%f",&E1.fname,&E1.lname,&E1.age,&E1.salary);
      printf("the information about this employee are \n");
      printf("%s\n%s\n%d\n%f\n",E1.fname,E1.lname,E1.age,E1.salary);
 }
```

The algorithm

```
Algorithm exe1
  struct employe
        fname  :c.c
        lname :c.c
        age; integer;
        salary:real
   end struct
  var E1:    struct employe
   Begin
   Write (" filling the information of this employee \n");
   Write (" please enter the first and the last name the age and   the salary \n");
   read(E1.fname,E1.lname,E1.age,E1.salary);
```

```
    write ("the information about this employee are \n");
    write (E1.fname,E1.lname,E1.age,E1.salary);
```

end

Exercise 02 : write an program that can fill a list of 50 students with(fname ,lname, and a list of 4 markes),

```
#include <stdio.h>
#include <stdlib.h>
struct Etudiant {
        char Fname [30] ;
        char Lname [30] ;
        float notes [4] ;
        } ;
struct Etudiant tab [50];
int main ( )
{  for (i = 0; i < 50; i++)
 {
printf ("enter the first name of the student number %d ", i+1);
scanf ("%s", tab[i].fname);
printf ("enter the last name of the student number n°%d ", i+1);
scanf ("%s", tab[i].lname);
 for (j =0; j < 4; j++)
 { printf ("enter the  grade %d of the student n°%d  ", j+1, i+1);
 scanf ("%f", &tab[i].notes[j]);
 } }return 0 ;}
```

BIBLIOGRAPHY:

- BestCours. (n.d.). *Bases essentielles de l'algorithmique et programmation en C* [PDF]. BestCours. https://www.bestcours.com/programmation/algorithme/444-algorithmique-et-programmation-les-bases-c-corriges-pdf.html
- Hassan El Bahi Algorithme - Recherche Vidéos
- Apprenez à programmer en C - OpenClassrooms
- Tshikutu, A. B. (2024). Logique de programmation et algorithme I (Licence). Congo-Kinshasa. HAL. https://hal.science/hal-04553652v1
- Amadio, R. M. (2023). *Notes de programmation (C) et d'algorithmique* (Master's thesis). HAL Open Science. https://hal.science/cel-01957585v4
- Berthet, D., & Labatut, V. (2014). *Algorithmique & programmation en langage C – Vol. 1* (Licence, 232 p.). HAL Open Science. https://hal.science/cel-01176119v2
- Girardot, J.-J., & Roelens, M. (2012). *Langages et concepts de programmation : Structures de données et algorithmes en C (Cours 1A, 2012–2013).* École Nationale Supérieure des Mines de Saint-Étienne.