University of Adrar

Faculty of Material Sciences, Mathematics and Computer Science

Department of Mathematics and Computer Science

Algorithms and Data Structure 1

C
O
U
R
S
E

H
A
N
D
O
U
T

Full Algorithms and Data Structure 1 Course and Tutorials

Level : 1st YEAR LICENCE (LMD) in Mathematics and Computer Science

Semester: 1st Semester (**S1**)

Dr. KADDI Mohammed

Associate professor

University of Adrar

Foreword

This handout, a crucial resource, is specifically designed for first-year LMD students in the Mathematics and Computer Science field. It serves as a comprehensive course manual for the subject "Algorithmics and data structure 1", aiming to introduce the fundamental notions of algorithm and data structure. It's important to note that students should have a solid foundation in computer science and mathematics.

This handout is structured into six chapters as follows:

In the first chapter, a brief history of computer science and an introduction to algorithms are given.

The second chapter presents a simple algorithm's basic notions and its C language translations.

Conditional structures (in algorithmic language and C) are presented in the third chapter.

The fourth chapter describes the different control structures (loops) in algorithmic language and C that can be used in an algorithm (For, While, Repeat, and nested loops).

Chapter five covers the use of arrays and strings in programming.

Finally, chapter six is devoted to custom types such as enums and records.

A list of bibliographical references is given at the end of this manuscript.

Table of contents

Chapter 1: Introduction

1. Introduction

Primitive people were the first to use the counting device. They used stones, bones, and sticks as counting tools. Over time, the human mind and technology leaped forward, and the development of other computing devices began.

Data structures and algorithms (DSA) are essential for any programming language. Every programming language has its data structures and different types of algorithms to handle these data structures.

Data structures organize and store data effectively when performing data operations. The algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a particular order to get the desired output. Algorithms are generally created independently of underlying languages, i.e., an algorithm can be implemented in multiple programming languages.

Almost every enterprise application uses various types of data structures in one way or another. As a programmer, I know that data structures and algorithms are essential to day-to-day programming.

2. Computer Science

2.1 Definitions

- Computer science designates the automation of information processing by a concrete (or an abstract system.
- Computer Science refers to all of the sciences and techniques related to information processing.
- Computer Science can also refer to what relates to computer hardware (electronics), and office automation (bureautique "FR").
- Computer science is the science of automatic and rational processing of information. This processing is done by a machine called a computer.

- Computer science is the science of automatic processing of information through the computer, i.e., automating the information that we manipulate. Its purpose is to develop and formulate the set of commands, orders or instructions allowing the computer to be controlled and oriented during the processing.
- The French translation is « L'informatique ».

2.2 Approch

« Computer Science is the automated processing of data (or information) by an electronic device the computer ».

- ✓ Data or Information: computer manipulates numbers.
  - ➢ can represent various types of information numbers (calculations or accounting), text, letters (characters), images or videos.
  - ➢ converting this information into a series of numbers raises the problem of data format, coding and standardized formats.
- ✓ Automated:
  - ➢ the processing is defined in a program which runs on its own.
  - ➢ the user simply provides processing parameters.
  - ➢ establishing this program is the domain of programming.
- ✓ Processing: these data are:
  - ➢ **creates**: automatic acquisition, type text, draw an image, record sound or video.
  - ➢ **analyse**: number analysis, finding word occurrences, identifying an object, speech recognition.
  - ➢ **modify**: calculations, typing text, modifying existing text, modifying contrast, brightness, colors, applying effects.
  - ➢ **archive** and **restore**.

2.3 Terminology

- ✓ **Informatics** (*science de l'information*): the study of systems, biological or artificial, that record, process and communicate information.
- ✓ Computer science (*l'Informatique théorique*): procedural epistemology, the study of algorithms, software, and computers.
- ✓ **Computer engineering** (*Génie informatique*): the manufacture and use of computer hardware.

- ✓ **Software engineering** (*Génie logiciel ou ingénierie de logiciel*): software modeling and development.
- ✓ **Information technology** (*Technologies de l'information*): represents the evolution of techniques and technologies related to Computer Science.
- ✓ **Information and communication technology**: (*Technologies de l'information et de la communication TIC*).

2.4 Application areas of Computer Science

- ✓ **Computer Science for management**:
  - ➢ guide management and management processes in companies,
  - ➢ areas of activity human resources management, sales administration, purchasing management, marketing, finance.
- ✓ **Computer Science scientifique** :
  - ➢ assist design engineers in industrial engineering fields,
  - ➢ design and size equipment using calculation programs,
  - ➢ used in design offices to simulate scenarios quickly and reliably.
- ✓ **Real time Computer Science** :
  - ➢ define software for controlling systems in direct contact with the world.
  - ➢ aeronautics space, weaponry, miniaturization of circuits.
- ✓ **Knowledge management**: (*L'ingénierie des connaissances*)
  - ➢ consists of managing innovation processes,
  - ➢ bring coherence between the three areas of management, real time, and scientific.
- ✓ **Intelligence** (*Intelligence ou Veille*) **Economic and strategic** :
  - ➢ use information technologies to search for information (search engines).

3. Information

- Information is an element of knowledge that can be represented using conventions to be manipulated (stored, processed or communicated) by a computer. It can be: text, sound, image, video, etc.

- Information is a set of events that can be communicated to the computer. It can be : text, sound, image, video, etc.

4. Computer

- A computer is a set of electronic circuits for manipulating data in binary form, represented by variations of an electrical signal.

- Computer is a machine capable of processing information and is composed of two parts: **HARDWARE** and **SOFTWARE**.

- The computer is a very powerful device that can process information with a very high speed, a high degree of precision and has the ability to store all this information. The computer can receive data as input, then perform operations on this data according to a program, and finally provide results as output.

The computer is made up of **two parts**: the **hardware part** and the **software part**. The combination of these two parts forms what is called: **computer system**.

- ✓ **HARDWARE**: is the physical or hardware part of the computer (keyboard, screen, processor, memory, motherboard, etc.).
- ✓ **SOFTWARE**: is the program part in a computer (operating system, application program, etc.).
- ✓ **Operating System**: It is a set of programs for computer management (memory management, file management, etc). It acts as an intermediary between the user and the machine.

4.1 Types of computers

- ✓ **Computers or PC**: desktop or laptop
  - ➢ composed of a central unit a case containing the motherboard, processor, RAM, power supply, and storage units.
  - ➢ A console: a screen (output), a keyboard and a mouse (input).
  - ➢ Various peripherals: a printer, a scanner, etc..
- ✓ **Workstations**:
  - ➢ Particularly powerful and expensive PCs,
  - ➢ used only for professional purposes.
- ✓ **Mainframes**:
  - ➢ A cabinet houses the CPU and power supply, one or more storage devices (hard drive, backup) while Hardware & Network Equipment (router, hubs, modem) are in the same room (separate racks).
  - ➢ An administration console (screen, keyboard, etc..) is located in this same room.

- ✓ **Servers:**
    - ➢ universal storage location for users connected to servers,
    - ➢ perform tasks serve as a firewall, host a web server, etc...

4.2 Hardware part (Computer Components)

It is the physical and tangible part of the computer system. It is divided into central unit and peripheral devices.

4.2.1. Central unit (UC)

It is the central functional element of any computer and is where most of the information processing takes place. It contains mainly:

- ✓ **Motherboard** : On the motherboard we find all the electronics of the computer, as main components we find the microprocessor (CPU) and the internal memory.
- ✓ **Memory:** there are mainly 3 types of RAM, ROM and storage media.
    - ➢ **RAM (**Random **Access Memory)**: It required for the execution of any program. Sometimes it's called PC memory or just memory. In essence, RAM is your computer or laptop's short-term memory. It's where the data is stored that your computer processor needs to run your applications and open your files. This memory is called RAM because its content is always changing, it is purely volatile, which means it will not retain data if there is no power. It is therefore important to save data to the storage device before the system is turned off.
    - ➢ **ROM** (Read Only Memory) is the memory that contains a program necessary for starting the computer (BIOS), it is called read-only memory since its contents never change by a simple user (neither modify, nor delete, nor add).

Since RAM cannot save user information (volatile memory), as well as ROM (read only memory), then where can one save for example a whole day's work from? this is where the so-called storage media (or external memories, auxiliary memories, mass memories) come into play.

- ➢ **Storage media:** used to save information. Examples: hard disk, floppy disk drive, removable disk drive (Zip or other), CD-ROM drive or burned DVD, USB key, external hard drive, etc.

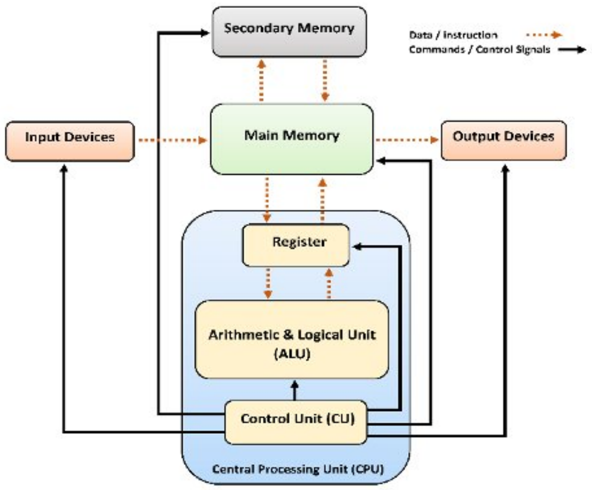
- ✓ **UCT or CPU (Central Processing Unit) or Processor**: It is the brain of the computer, i.e. it is responsible for any operation carried out inside the computer (e.g. printing a page, drawing a table, listening to a song, making a calculation, send an email, etc.). Conventionally, the processor is composed of:

- **Command and control unit (UCC):** its role is to control the operation of the computer. It is a component of a computer's central processing unit (CPU) that directs the operation of the processor.
- **Arithmetic and Logical Unit (UAL):** its role is to perform arithmetic and logical operations (greater, lower, equal, intersection (AND), union (OR), etc.);

4.2.2. Peripherals

There are two types:

- ✓ **Input devices:** The devices which are used to give input from the external world to the computer system are known as input devices. The most widely used and popular input devices in the world of computers are **keyboard**, **mouse**, **scanner, microphone, barcode reader, optical** and **Web camera.**
- ✓ **Output devices:** The devices which are used to give output from the computer system to the external world is known as output devices. The most widely used and popular output devices in the world of computers are **monitor (screen), printer**, and **the audio speakers**, etc.



Basic Architecture of a computer

4.3 Software part

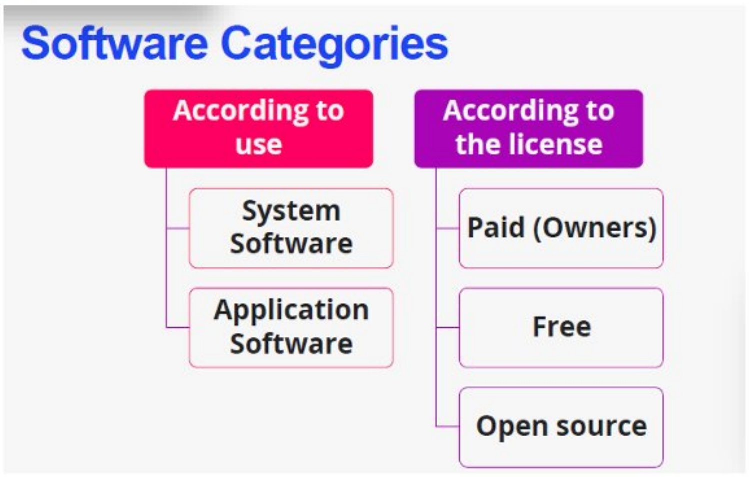
4.3.1 Definitions

- A Software refers to the intangible part of Computer Science, the organization and processing of information: **programs**.

- A Software is a set of programs that allows a computer or computer system to perform a particular task or function.

- Software is a set of instructions, data or programs used to operate computers and execute specific tasks. It is the opposite of hardware, which describes the physical aspects of a computer.

4.3.2 Software Categories



Software Categories

a) Softwares according to use

✓ **System Software:**

- **Operating systems:** MS Windows, Linux Ubuntu, Androis iOS, Mac OS...
- **Device drivers:** Motherboard drivers, graphics card drivers, etc...
- **Firmware:** BIOS, UEFI, Embedded systems, …
- **Programming language tools:** Interpreter compiler and assemblers,
- **Utilities:** Anti-virus (Avast, McAfee), CCleaner WinRAR, …

✓ **Application Software:**

- **Content Management:** MS Word, Google Docs, …
- **Database Management:** MS Access, MySQL, Oracle, …
- **Multimedia:** Adobe Photoshop, VLC Media Player, Inkscape, …

- **Web browser:** Google Chrome, Mozilla Firefox, …
- **Accounting and Management:** SAP, Ciel, …

b) Softwares according to the license

- **Paid software:** MS Office, SAP, Sky, …
- **Freeware:** Adobe Reader, Skype, TeamViewer, …
- **Shareware:** Adobe Acrobat, Winzip, etc…
- **Free Software (Open Source):** Moodle, Mozilla Firefox, Apache Web Server, …

So, the two main categories of software are **application software** and **system software**.

- An application is **software** that *fulfills a specific need or performs tasks*. System software is *designed to run a computer's hardware and provides a platform for applications to be run on top of.* Other types of software include **programming software**, which provides the programming tools software developers need; middleware*, which sits between system software and applications; and* **driver software**, *which operates computer devices and peripherals*.

- Computer program, detailed plan or procedure for solving a problem with a computer; more specifically, an unambiguous, ordered sequence of computational instructions necessary to achieve such a solution.

Software is generally a set of programs designed to perform a complex task or process automatically.

A machine can host any number of software packages. However, an operating system is the basic software that must be installed for the first time. An operating system was a set of programmes that ensured the operation of all the hardware components of a computer and man-machine communication. Exp: MS-DOS (Microsoft Disk Operating System), Windows 95, Windows 98, 2000, XP, vista, Unix and Linux. Once man can communicate with computer, what can he do with this machine? → It is the application software that tells him what he wants to do.

Example of computer applications :

- MS Word.............word processing
- MS Excel...............financial and graphical analysis.
- MS Power Point..........computer-assisted presentation.
- Real one Player...........to play music
- CD playback

- Chat, e-mail, Messenger, ...
- **Programming:** the subject of this course.

In short, we can say:

Programming is all the activities involved in designing, producing, testing and maintaining programs.

- A program is a sequence of instructions or operations designed to solve a given problem, to relieve human effort, to save time and more particularly to avoid errors.

- To program, we first need to know and master what is known as an algorithm. Algorithms are the basis of programming, which will follow you throughout your studies.

If we want to give the definition of an algorithm, we say that it is similar to that of a program? so what's the difference?

- An algorithm is a sequence of finite, ordered instructions or operations for solving a given problem, written in the user's language (a language that is not understood by the computer), whereas a program is written in a language that is understood by the computer; this language is called the programming language.

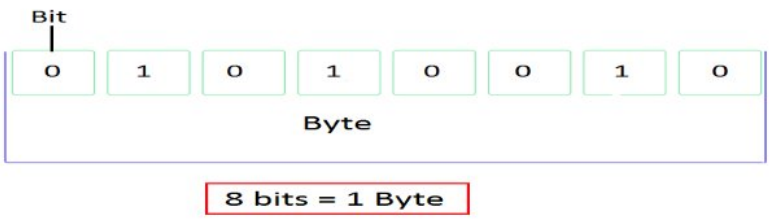
We can say that a program is an algorithm translated into a programming language.

We will look all these concepts in more detail in the chapters that follow.

5. Data units of measurement

A measure of the amount of information or storage capacity.

- ✓ **bit (binary digit) :** It is the minimum unit of information and is equivalent to a binary digit that can be 0 or 1.
- ✓ **Byte (octet)**: It is an 8 bit set. It can contain one character and can take values between 0 and 255.



- ✓ **Word**: A word can have a variable number of bits depending on the computer system we are dealing with. In current computers it varies from 16 bits to 128 bits.

Unit	Shortened	Capacity (Value)	Conversion
Bit	b	1 or 0 (on or off)	
Byte/Octet	B/octet	8 bits	1 Byte= 1 octet =8bits
Kilobyte	KB (Ko)	1024 bytes	1 Ko=2^{10} Octets
Megabyte	MB(Mo)	1024 kilobytes	1 Mo=2^{20} Octets
Gigabyte	GB(Go)	1024 megabytes	1 Go=2^{30} Octets
Terabyte	TB(To)	1024 gigabytes	1 To=2^{40} Octets
Petabyte	PB(Po)	1024 terabytes	1 Po=2^{50} Octets
Exabyte	EB(Eo)	1024 petabytes	1 Eo=2^{60} Octets
Zettabyte	ZB(Zo)	1024 exabytes	1 Zo=2^{70} Octets
Yottabyte	YB (Yo)	1024 zettabytes	1 Yo=2^{80} Octets

Units of Measure && Conversion table for Digital Information

6. Introduction to Algorithmic

6.1 Solving a Problem by Computer

6.1.1 Problem Definition:

Definition 1: A problem is a theoretical or practical question that involves difficulties to be solved or whose solution remains uncertain.

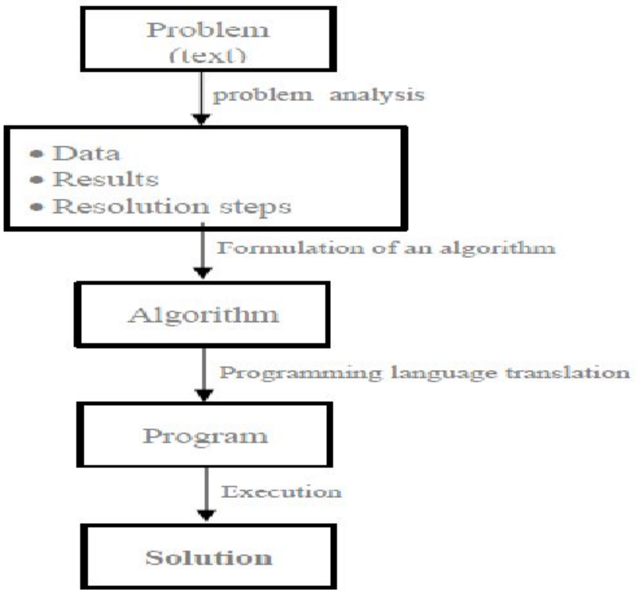
Definition 2: Question to be solved by rational or scientific methods.

Definition 3: An issue that can be resolved from the elements given in the statement.

6.1.2 Steps for solving a problem by a computer

To solve a problem with a computer, you:

- ✓ **Analyzes the problem:** by defining the data (inputs) and the results (outputs) of the problem and determining the procedure to follow (steps of resolution) which allows to obtain a solution to the problem.
- ✓ **Formulates the algorithm**: this step is used to represent the resolution steps in the algorithmic language (pseudo-code).
- ✓ **Translates the algorithm into a program:** the algorithm must translate into a program written in a programming language.
- ✓ **Runs the program:** on the machine to obtain a solution.



Steps for solving a problem by a computer

6.1.3. Algorithm representation

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

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

6.2 Algorithm and algorithmic concept

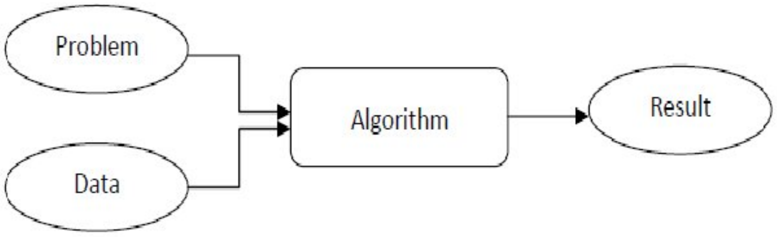
The word algorithm comes from the name of the famous Arab mathematician: Mohamed Ibn Moussa El Khawarizmi (780-850). A computer programme enables the computer to solve a problem, but before we can tell the computer how to solve the problem, we must first be able to solve it ourselves in the form of an algorithm.

- ✓ **Algorithmics**: It refers to the discipline that studies algorithms and their applications in computer science.

- ✓ **An algorithm**: is a sequence of ordered and finished instructions (operations, actions or treatments) to solve a given problem.

The problem solving steps (Algorithm) are :

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



Problem solving steps.

6.3 Program and programming

- ✓ **A program**: is a series of ordered and finished instructions that are executed to achieve a given objective (problem solving). The programme will simply be the translation of the algorithm into a programming language, i.e. a language that is simpler than French in its syntax, without ambiguities, that the machine can use and transform to execute the actions it can describe. Pascal, C, Java and Visual Basic are all names of programming languages.
- ✓ **A computer program**: is a sequence of ordered and finite instructions that are executed *by a computer* in order to solve a given problem.
- ✓ **Programming**: refers to all activities that enable the writing, testing and maintenance of computer programs.

Note: A program is an algorithm written in a programming language.

6.4 Algorithm and program

- The development of an algorithm precedes the programming stage.
- A program is an algorithm.
- A programming language is a language understood by the computer.

- Developing an algorithm is a demanding problem-solving process.
- Writing an algorithm is an exercise in thinking on paper.
- The algorithm is independent of the programming language. For example, the same algorithm will be used for an implementation in Java, C++ or Visual Basic.
- The algorithm is the crude resolution of a computer problem.

6.5. Creating a program

Solving a given problem involves a succession of stages, as follows:

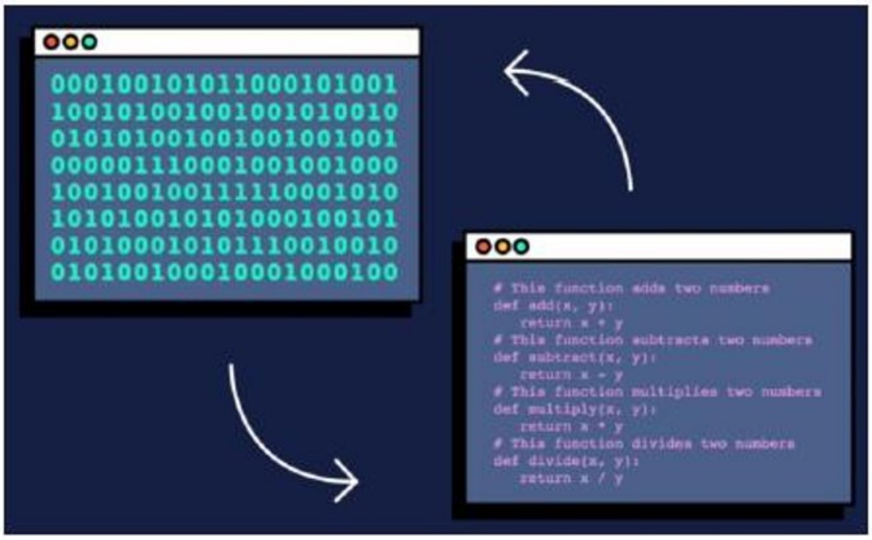
Problem → Explicit statement → Algorithm → Program

When writing a program, two types of error can occur:

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

6.6 Programming languages (computer languages)

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



```
000100101011000101001
100101001001001010010
010101001001001001001
000001110001001001000
100100100111110001010
101010010101000100101
010100010101110010010
010100100010001000100
```

```
# This function adds two numbers
def add(x, y):
    return x + y
# This function subtracts two numbers
def subtract(x, y):
    return x - y
# This function multiplies two numbers
def multiply(x, y):
    return x * y
# This function divides two numbers
def divide(x, y):
    return x / y
```

Programming languages fall into two different classifications — **low-level** and **high-level.** **Low-level** programming languages are closer to machine code, or binary. Therefore, they're harder for humans to read.

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

Two types of High-level programming languages are distinguished:

- ✓ **Procedural languages:** Fortran, Cobol, Pascal, C, …
- ✓ **Object-oriented languages:** C++, Java, C#,…

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

7. Basics of an algorithmic language

7.1 Basic structure

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

```
Algorithm name_of_the_algorithm
CONST { Defining constants }   ⎤
TYPE  { Defining types }       ⎬ they are optional (if there is no var or const or type,
                               ⎦                         they are not written)
VAR   { Declaring variables}
BEGIN
      { Sequence of instructions }
END.
```

- **Header part:** this is the first line of the algorithm, starting with the word algorithm followed by a space followed by the name of the algorithm.
- **Declarations section:** in this section, all the objects used to solve the problem must be declared.
- **Processing section:** this section begins with the word Begin and ends with the word End. It contains the actions used to solve the problem.

7.2 Basic instructions

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

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

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

7.3 Constants and variables

Objects are divided into two classes: **constants and variables**.

7.3.1 Notion of variable

The data manipulated in an algorithm are called variables. In a programming language, a variable is used to store the value of a datum. It designates a memory location whose contents can change during the course of a program (hence the name variable).

Each memory location has a number that allows it to be referred to uniquely: this is the memory address of that cell.

Rule: The variable must be declared before being used, and must be characterized by:

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

Identifiers : rules

The choice of name for a variable is subject to a few rules which vary according to the language, but in general:

- A name must begin with an alphabetical letter. Example: E1 (1E is not valid)

- Must consist solely of letters, numbers and underscores ("_") (avoid punctuation characters and spaces). Examples: SMI2008, SMI_2008. (SMP 2008, SMP-2008, SMP;2008: are invalid).

- Must be different from the language's reserved words (for example in C: int, float, double, switch, case, for, main, return, ...).

- The length of the name must be less than the maximum size specified by the language used (in C, the name must be no longer than 32 characters).

- The C language is case-sensitive: a and A are two different names.

Tip: to make the code easier to read, choose meaningful names that describe the data being manipulated. Examples: Student _Mark, Price_TTC, Price_HT

Variable types

The type of a variable determines the set of values it can take. The types offered by most languages are :

- Numeric type (integer or real).
- Byte (encoded in 1 byte): from [-27,27[or [0, 28[.
- Short integer (coded on 2 bytes): [-215,215[
- Long integer (coded on 4 bytes): [-231,231[
- Single-precision real (coded on 4 bytes): precision of order 10^{-7}
- Double precision real (coded on 8 bytes): precision of order 10^{-14}
- Logical or Boolean type: two values TRUE or FALSE
- Character type: upper case letters, lower case letters, digits, symbols, ... Examples: 'A', 'b', '1','?', ...

- String type: any sequence of characters. Examples: " " Last name, First name", "postal code: 1000", ...

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

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

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

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

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

- Logical operators: NOT, AND, OR

Truth table :

A	B	A ∧ B	A V B
FALSE	FALSE	FALSE	FALSE
FALSE	VRAI	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

A	NON A
FALSE	TRUE
TRUE	FALSE

Comparison operators: = , ≤ , ≥ , ≠

D) Character

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

E) String

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

Variable declaration

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

***Variable** list of identifiers : type*

Example:

Variable

i, j, k : integer

x , y : real

OK : boolean

Ch1, ch2 : string

In C: basic types :

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

In general, their declaration is as follows:

```
<type><name> ;
<type><name1>,<name2>,<name3> ;
```

Examples :

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

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

7.3.2 Notion of constant

A constant is a variable whose value does not change during program execution. It can be a number, a character or a string of characters. In pseudo-code,

Constant identifier = value (by convention, constant names are in uppercase)

Example: to calculate the area of circles, the value of *pi* is a constant but the radius is a variable.

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

In C :

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

7.4 Basic operations

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

7.4.1 Assignment

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

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

Assignment only modifies what is to the left of the arrow.

Example 1 : i ←1, j ←i , k ←i+j , x ←10.3 , OK ←FAUX , ch1 ←"SMI" , ch2 ←ch1 , x ←4 , x ←j (avec i, j, k : entier; x :real; ok :boolean; ch1,ch2 : string)

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

Example 2

a ←10 a receives the constant 10

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

d ←'m' d receives the letter m

Remarks

- The C programming language uses the equal sign = for the assignment ←
- When an assignment is made, the expression on the right is evaluated and the value found is assigned to the variable on the left. Thus, A←B is different from B←A
- Assignment is different from a mathematical equation.

- The operations x ← x+1 and x ← x-1 have meaning in programming and are called incrementing and decrementing respectively.
- A+1 ← 3 is not possible in programming languages and is not equivalent to A ← 2.
- Some languages give default values to declared variables. To avoid any problems it is preferable to initialise declared variables.

Example in C

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

Some useful operators

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

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

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

7.4.2. Reading

read (variable)

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

Examples of reading

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

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

In C :

Input with scanf.

- Reads from the standard input (the keyboard)

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

Examples

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

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

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

7.4.3 writing

Write (expression)

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

writing examples

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

write(12) Displays the value 12

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

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

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

Examples in C

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

Main format codes

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

Formatting characters

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

7.5 General syntax of the algorithm

```
Example algorithm
Constant  var1 = 20 , var2 = "hello! "
Variable
var3, var4 : real
var5: character
Begin // body of the algorithm
        Instruction 1
        Instruction 2
        .............
        Instruction n
End
```

Notes:

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

7.6 Expressions and operators

An expression can be a value, a variable or an operation made up of variables linked by operators.

Examples: 1, b, a*2, a+ 3*b-c, ...

Evaluating the expression provides a single value, which is the result of the operation.

The operators depend on the type of operation:

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

Note:

- ✓ An integer and a character cannot be added together.
- ✓ However, in some languages, an operator can be used with two operands of different types, as is the case with arithmetic types (4 + 5.5).

- ✓ The meaning of an operator can change depending on the type of operands, for example:
  - ✓ The + operator with integers performs addition, 3+6 is 9
  - ✓ With character strings, it performs concatenation: "hello" + "everyone" is "hello everyone".

7.7 Operator priority

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

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

Example : 9 + 3 * 4 is 21

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

7.8 Boolean operators

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

Example : let a, b, c and d be any four integers:

(a<b)||((a>=b)&&(c==d)) ≡ (a<b)||(c==d)car (a<b)||(!(a<b)) always true.

Chapter 2: Simple Sequential Algorithm

1. Introduction

When the computer was invented in the 1940s, it operated using electronic tubes, and its programming (writing commands) was done by entering a series of numbers composed of zero (0) and one (1). It was difficult for programmers. However, with the creation of the transistor, computers became much smaller, and their capabilities increased. Easier-to-use programming languages were invented, which are very similar to human language. These languages are called high-level languages. Any program written in these languages can be quickly and automatically translated into machine language (0s and 1s) using a compiler.

In this chapter, we will cover the concept of language, algorithm structure, the concept of variables and constants, and also explore some types of simple instructions such as assignment, input, and output.

2. Notion of Language and Algorithmic Language

2.1. Language

Language is a means of communication and understanding among human beings. In the case of a computer, it is the way a computer understands human commands. Language consists of an alphabet, symbols, vocabulary, grammatical rules, and meanings.

- Alphabet: It is a set of letters, numbers, and symbols.
- Vocabulary: A collection of symbols and words, whether reserved words or those defined by the programmer.
- Syntax: Rules or laws that govern the grouping and placement of symbols and vocabulary.
- Semantics: Specifies the meaning of each instruction that can be constructed in the language, particularly what it will produce during execution.

2.2. Programming Language

A programming language provides us with a framework to develop algorithms and produce programs that a computer can execute. It allows us to describe the data structures that will be processed by the computer and the operations that will be performed. It serves as an intermediate language between human language and machine language. Humans can understand it, and computers can translate it into the 0s and 1s of machine language that they understand.

There are many programming languages, each with its own advantages and rules, making them suitable to varying degrees for specific types of software. Programmers must know some of these languages and know which language is appropriate for each type of application.

2.2.1. Types of Programming Languages

Programming languages can be divided into two parts:

- Interpreted: Interpreted: If the program is not fully translated into machine language, but rather translated and executed instruction by instruction. Examples: Matlab and web languages.
- Compiled: If the program is fully translated into machine language before execution. Example: C.

Examples of Programming Languages:

- C and C++: which will be used in this course and are considered the parent language of many others.
- Pascal: used for educational purposes and closely related to algorithms.
- Delphi and WinDev: good for developing management software.
- Java: suitable for network and mobile applications.
- C#(C-Sharp) and Visual Basic: useful for developing software specific to the Windows environment.
- Objective C (Xcode): used for software development for Apple products (Mac, iPad, and iPhone).
- PHP: specific to web development.
- Matlab and Python: specific to data analysis and used by engineers.

2.2.2. Compiler

A computer program that converts the source code written in a particular programming language in to targeted code that can be executed directly by a computer. There are many programming languages that a programmer cannot know all of them. To enable programmers to work in teams and exchange solutions, they need to formulate these solutions in algorithmic language. Algorithmic language is the common language of all programmers.

2.2.3. Integrated Development Environment (IDE)

To write a program, you can use any text editor such as Notepad. However, this method makes the development process very difficult. Therefore, there is a set of programs that provide all the necessary tools for the development process called IDE (Integrated Development Environment). IDE provides all the necessary tools for designing, developing, testing, debugging, and deploying applications, which makes development easier and faster. The IDE includes all the necessary tools to start designing applications, such as:

- Code editor: for writing and editing program code. It performs automatic formatting, making the process of reading easier.
- Project manager: to manage the files that make up a single project.
- Debugger: detects and corrects errors in the code.
- Shortcuts to compile and run the program.
- Other tools...

Examples of IDEs: Dev-C++,Embarcadero,Visual Studio...

2.3. Basic Syntactic Elements

2.3.1. Reserved Words

Reserved words are words that have preexisting meaning in the programming language and cannot be used by the programmer to create new elements. Each language has its own reserved words, such as "algorithm" "begin," and "end" in the algorithm, and "if" and "while" in C.

2.3.2. Values

Values can be numbers, characters (always enclosed in single quotes ''), strings of characters (always enclosed in double quotes ""), true, or false. See types.

example: -2,7,3.12,6e-7,'k','خ','1','!',"azerty","سلام",true,false

2.3.3. Identifier

An identifier is the name given by the programmer to any element of the algorithm that they want to create. Examples include the name of the algorithm, variable name, type name, constant name, function name, etc. There are rules and conditions in the C language that we adopt in the algorithm for naming identifiers:

Rules for Naming Identifiers:

- The identifier name can only contain literal symbols and numerals from A to Z, a to z, and 0 to 9, as well as the underscore symbol "_".
- It must be a single word, meaning the name cannot contain spaces " ".
- It must start with a letter or "_", not with a digit.

- It must not be a reserved word.
- The identifier must be unique; it is not possible to define more than one element with the same name.
- It is recommended to use meaningful names, for example, we use "Width" instead of "x".

Examples of Valid Identifiers:

x, pi, Mat_info, isEmpty, n5, _if, _0a (it's better to avoid it)

Examples of Invalid Identifiers:

α,ج , π, é, élève (unacceptable symbols)

3a (starts with a digit)

Mat info (contains a space) begin, end, if (reserved words)

2.3.4. Operations

- **Arithmetic Operations:**

Operation	**C**	**comment**	**example**
- +	- +	Sign	+3 -7 -a
- +	- +	The two operands are integers, the result is an integer. one is real, result is a real number.	5+3.0 real
*	*	For the product, like addition and subtraction.	5*3 integer
/	/	for real division. The result is always a real number	5/3 or 5.0/3 in C
mod	%	To calculate the remainder of the division. Both operands are integers The result is always an integer.	5**mod**3 or 5%3 in C
div	/	To calculate the quotient (integer division). Like the remainder	5div 3 or 5/3 in C
^		to calculate the power of a number. In C, the function pow() is used, the result is a real number	5^2orpow(5,2)inC
√		to calculate the square root of a number. In C, the function sqrt() is used.	√5or sqrt(5) in C

- **Relational Operators:**

Algo	**C**	**comment**
>, >=, <,<=		The same in C
=	==	In C « == » twice = is read as equal, while « = » is used for assignment and read receives.

≠	!=	Not equal

The result of the comparison is always a logical **Boolean**, i.e., **true** or **false**.

- **Logical Operators:**

Operation		**C**	**observation**
not	negation	!	true if the operand is false, and false if it's true.
and		&&	true only if both operands are true, otherwise, it returns false
or		||	true if at least one of the operands is true, otherwise, false.
xor	exclusive OR		true if one is true and the other is false, otherwise, it returns false.
=	equivalence	==	true if both operands are equal..

- **String Operators:**

+: used to concatenate two strings. For example, "hello" + "world" gives "helloworld" without adding spaces. "hello" + " " + "world" gives "hello world".

- **Operator Priority:**

When evaluating an expression, we follow the priority summarized in the following table.
If the priorities are equal, priority is given to the operation on the left.

priority	the operation
0	()
1	+ and - sign,not.
2	* / mod div
3	- +
4	>, >=, <,<=
5	≠, =
6	and
7	or

Parentheses are used to change priority (and sometimes for readability).

2.3.5. Expression

An expression is a structure of values and identifiers, connected by operations. When evaluated, it results in a single value. Expressions are created using values, variables, parentheses, and operations.

Example:

Assuming a=2, b=3, and ok=true,

expression	result	expression	result	expression	result
5	5	a+3	5	ok	True
a	2	"In"+"fo"	Info	a*(b-7)>8et ok	False

2.3.6. *Instruction*

An instruction (statement) is a command or step in the solution, meaning it is the action that will be executed.

2.3.7. *Block of Instructions*

A block of instructions is a set of instructions that begins with the word "Begin" and ends with "End," or begins with a reserved word defining the beginning, such as "if," and ends with "End" + the corresponding starting word, e.g., "End If." In C, it starts with "{" and ends with "}".

Example:

algorithm	C

Begin
Instruction 1;
…
Instruction n;
End
	{
	Instruction1;
	…
	Instructionn;
	}

2.3.8. *Comments*

Comments are texts that are ignored during the translation of the program and are not part of the algorithm. They are added to programs to provide explanations and facilitate understanding.

In C, comments can be added using «//» for single-line comments. It begins with // and ends with a line break.

Comments can also be added, starting with « /* » and ending with « */ », which can extend over several lines.

example:

```
//One-line commentary
/* Comment
It can span multiple lines*/
```

2.3 Expressing the Algorithm

An algorithm can be expressed by writing it in natural language, such as Arabic, French, or English. However, natural language is ambiguous and imprecise. Therefore, we write the algorithm using Pseudocode, flowcharts, or programming languages.

1- **Pseudocode**: Describes the algorithm in human languages like Arabic, French, or

English, in a manner similar to programming languages. Some use many details (to be closer to programming languages), while others use fewer details (closer to human language). There is no specific rule for writing this type of code.

2- **Flowchart**: An illustrated representation of the algorithm that shows the steps to solve the problem from start to finish while abstracting away the details to provide an overview of the solution. Arrows and agreed-upon geometric shapes are used to represent the steps.

3- **Code**: Where the algorithm is written in a programming language directly, such as C, so that the computer can translate it into binary language for direct execution by the processor.

3. Parts of an Algorithm

An algorithm consists of two main parts: data and instructions. An algorithm's structure is very similar to a recipe for cooking. It typically consists of a title, followed by ingredients, and finally, the preparation method.

An algorithm takes the following form:

```
Algorithm name;
  Declaration of the data needed;
Begin
  Instructions;
End.
```

The algorithm is composed of three basic parts:

- **Header:** Comprised of the word "Algorithm," followed by the name that explains the problem to be solved. The name should be a valid identifier.
- **Declarations:** Reserved for reserving memory space for data (constants and variables) that will be used as input and output.
- **Instructions:** A set of steps or commands that will be executed during the algorithm's execution. It starts with "Begin" and ends with "End." There are five main types of instructions:
    1. Assignment instruction.
    2. Read (input) instruction.
    3. Write (output) instruction.
    4. Conditional instruction.
    5. Iterative (loop) instruction.

4. Data: Variables and Constants

4.1 Constant

A constant is a value (numeric or symbolic) that has a name and cannot be changed during program execution.

Constant declaration:

```
  Const Identifier=value
```

Const or **Constant**: These are two reserved words that allow constant declaration.

Identifier: The name given to the constant.

Value: The value assigned to the constant.

Example:

```
Const
Pl= 3.1415926
DEP = "قسم الاعلام الالي"
```

Advantages of Constants :

− Condenses the code, where a long phrase can be replaced with a short word, such as using "PI" instead of 3.1415926.

− Helps avoid errors by providing a meaningful name. For example, "PI" instead of 3.1415926.

− Simplifies code maintenance, as the value needs to be changed in one place only.

4.2 Variable

A variable is a location in memory used to store data. It has a name, a type, and a value (address in the second semester).

- **Name**: Identifier used by the programmer to refer to and manipulate the variable. For example, "weight."
- **Type**: In computers, everything is represented as 0 and 1. The type determines how it is translated, as well as the size of memory to reserve. For example, "int" (32 bits).
- **Value**: The content of the bits that make up the variable, i.e., its value. Typically, this is the part that changes during program execution. For example,1101represents the number 13, or -5 if we consider the leftmost 1 as the sign "-".

Variable Declaration:

```
  Var Identifier: Type;
```

Var or **Variable**: These are two reserved words that allow variable declaration.

Identifier: The name given to the variable.

Type: The type of the variable.

A comma "," can be used to declare multiple variables of the same type.

Examples:

```
Var
   age: integer
   gender: character
   x, y, z: real
```

Note: By convention, constant names are written in uppercase, and variable names are written in lowercase.

5. Data Types

A data type represents the domain to which data belongs, such as numbers, text, images, audio, or video. The data type determines how the bits, which compose the variable, are translated and the size of memory to reserve, i.e., the number of bits and allowed operations. When defining a variable, its type must be specified. There are five basic data types in the algorithm:

1. Integer such as: -5, 0, 1,13
2. Real: -7, 0,1, 3.14, 2.7e03
3. Boolean Contains only true or false.
4. Character: Includes all symbols on the keyboard, such as digits, letters in all languages, and printed (visual) and unprinted symbols. They are always enclosed in single quotes (e.g., 'a', 'M', ' س' ,',' ,'+' ,' 1 ').
5. String: A set of symbols, with a length of 0 or more, always enclosed in double quotes (e.g., "computer," "Good luck\n," "1", "3.14").

Notes:

− We use "." instead of "," to express decimal numbers.

− 1 is not the same as 1., not the same as '1', and not the same as "1." The first is an integer, the second is a real number, the third is a character, and the last is a string.

− 'a' is not the same as "a" The first is a character, and the second is a string of length 1.

− Lowercase letters are not the same as uppercase letters. For example, 'a' is not the same as 'A'

− Some symbols (keys) do not print. For example, space ' ' or newline '\n'.

− The backslash (\) is used to represent some invisible or special symbols visually. For example, newline '\n' and tab '\t'. To print double quotes, we use '\"' and to print a backslash, we use \\.

− There is an empty string, denoted as "", which contains no characters and has a length of 0.

6. Basic Instructions

6.1 Assignment

This is the process that allows us to store a value in a variable.

Syntax:

```
Variable ← exp ;
```

- variable: This is the name of a variable.
- exp: It is an expression (identifiers, values, and operations, see 2.3.4), calculated to obtain a unique value placed in the variable.

← read as "gets" in English. The arrow always points to the variable.

A variable can hold only one value at each point in the program's execution. When the operation is performed, only the left variable changes. It loses its old value and takes on the new one. For the assignment process to work correctly, the value of the right expression and the left variable must be of the same type or at least compatible types.

Example:

`a←5`	a gets 5
`b←a*2`	b gets 10
`a←0`	a gets 0
`b←b-1`	b gets 9
`c←'b'`	C gets the letter b
`d←b>a`	D gets true
`s←"name"`	S gets the word "name"

Before a variable can be used, it must be declared and assigned an initial value. To obtain the value of any variable or constant, simply write its name.

6.2 Input/Output Instructions

To interact with the user, the programmer has two instructions: **read ()** and **write ()**.

6.2.1 Input: Read ()

read () is a ready-to-use function in algorithms. You input a value from the user, via the keyboard, and assign it to the variable inside parentheses. It is always used for data entry.

Syntax:

```
Read(variable)
```

- variable: This is the name of a variable. read () can only be used with variables.

When the program is executed and the input instruction read () is encountered, execution is paused until the user enters data. The input process ends by pressing the Enter key. The program will continue to execute. Several variables can be entered at once, separated by a comma ",". In this case, the user enters the value of the first variable, then presses the space key, then enters the value of the second variable, and presses the Enter key only after entering the value of the last variable.

Example:

`read(name)`	The user enters a series of letters, for example, <Muhammad>, then presses the Enter key.
`read(a,b)`	They enter a number, for example, "15", then press space, then enter the second number, for example, "20", then press the Enter key.

6.2.2 Output: write()

`write()` is a ready-to-use function in the algorithm. It displays on the screen whatever we put inside its parentheses. It is always used to print results.

Syntax:

```
Write(exp);
```

or

```
Write("message");
```

- `exp:` This is an expression, calculated to obtain a single value, to display on the screen.
- `"message":` Any text you want to display as is on the screen. It is not calculated. It can be in any language or set of letters. It must be enclosed in double quotes, which are not displayed on the screen.

Several values and texts can be displayed at once, separated by a comma ",".

Example:

`Write(name);`	The value of the variable name appears on the screen, for example, <Mohammed>.

`a←5;`
 `Write(a+3);`
	Displays 8 without changing the value of a.

`Write("square of",a,"is",a*a);`	Displays: square of 5 is 25
`Write("b=",a);`	Displays: b=5

Notes:

− Always before the Read() instruction comes the Write() instruction, to explain to the user what is expected to be entered.

− The user writes on the keyboard, while the program (computer) reads <read()> from the keyboard, and the program writes <write()> on the screen, while the user reads from the screen.

− <read()> can be generalized for the input of all input units, and <write()> for the output of all output units.

7. Building a Simple Algorithm

After seeing that the algorithm consists of 3 parts, namely: header, declaration, and instructions, and we have learned to declare constants and variables, and we have learned 3 types of instructions, namely: assignment, reading, and writing. Now we can write simple algorithms.

To know the variables, we ask the question: "What data is needed and what is the expected result?" The instruction part usually consists of three basic steps:

− The first step, "Inputs": the data needed for implementation is entered using the <read()> instruction.

− The second step, "Processing": It contains a set of instructions necessary to solve the problem using the assignment instruction.

− The third step, "Outputs": the results are presented using the <write()> instruction.

Example1:

Write an algorithm that calculates the area of a circle.

```
Algorithm circle_area;
Const
  P=3.14;
Var
  r,s:integer;//r is the radius and s is the surface
Begin
  write("Enter the radius");
  read(r);
  s←p*r*r;
  write("The area of the circle is:", s);
End.
```

Example 2:

Write an algorithm that calculates the average for ADS1.

```
Algorithmavg_ADS1;
Var
exam,td,tp,avg:real;
Begin
write("Enter the exam score, tutorial score, and practical
score"); Read(exam, td, tp);
avg←(exam*3+td+tp)/5;
write("The average is:",avg);
End.
```

7.1 Execution of an Algorithm

The execution of the algorithm aims to know the value of each variable after each instruction. Where execution starts from the word Begin to the word End. At the beginning, the values of the variables are undefined (empty), then after each assignment or reading, the value of the variable changes.

Example

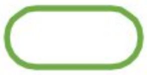
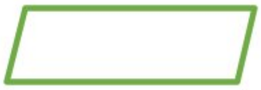
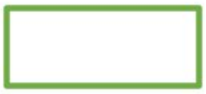
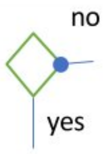
Algorithm mirror;	Execution		
Var			
a,b,c: integer;	a	b	c
Begin			
a←357;	357	?	?
c←0;	357	?	0
b←a **mod** 10;	357	7	0
c←c*10+b;	357	7	7
a←a **div** 10;	35	7	7
b←a **mod** 10;	35	5	7
c←c*10+b;	35	5	75
a←a **div** 10;	3	5	75
b←a **mod** 10;	3	3	75
c←c*10+b;	3	3	753
End.			

8. Representing an Algorithm with Flowcharts

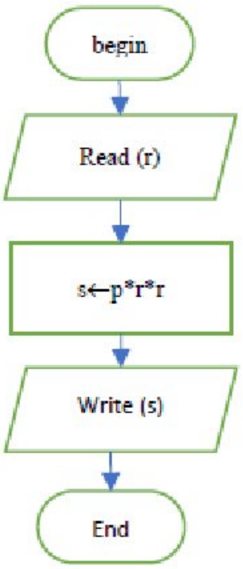
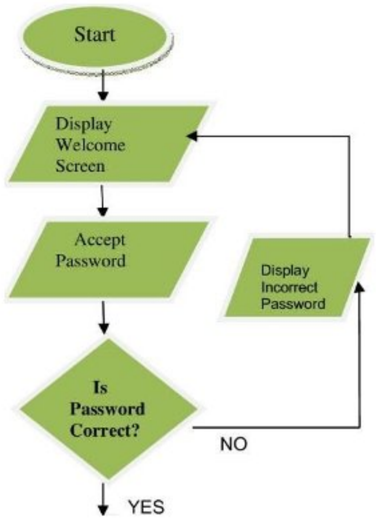
A flowchart is a visual representation of an algorithm before its programming. It shows us the sequence of operations and gives us the overall structure of the algorithm's components.

Flowcharts have many advantages: they provide a better visualization of ideas and are easily understood by everyone, facilitating teamwork.

Several shapes are used in a flowchart, with the most important being:

Symbol	Use
	Start, end, interruption
	Input - Output read() / write()
	Processing Symbols like assignment
 	Logical Symbols: Choices with conditions

Example:

Algorithm to calculate the area of a circle	the algorithm to check the accuracy of the password
	 Start Display Welcome Screen Accept Password Is Password Correct? NO YES Display Incorrect Password

9. Translation into C Language

"C" is a fully compiled high-level imperative language. It is one of the most widely used programming languages in the world and is considered the parent language of many programming languages. In this course, we will use Dev-C++ as an IDE. It's worth noting that "C" is case-sensitive, distinguishing between uppercase and lowercase letters. <main> is not the same as <Main>, <MAIN>, or <mAin>. Therefore, we recommend always writing in lowercase.

All simple statements (declaration, assignment, I/O, return) must end with a semicolon ";".

9.1 The Preprocessor

Before the program is actually compiled, the source code files are processed by a preprocessor, which resolves certain directives given to it. For example, including other files (libraries), replacing words with other phrases (macros).

A directive given to a preprocessor always starts with a #.

9.1.1 #include

The `#include` directive instructs the compiler to include the content of another file into the current program's code.

```
#include <filename>
```

Generally, these files are libraries of predefined and ready-to-use functions. Example:

Example:

- To use I/O functions (scanf and printf), we use the stdio.h library.
- To use mathematical functions (sin, cos, exp, pow, sqrt, ...), we use the math.h library.
- To use string functions (strlen, ...), we use the string.h library.

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

9.1.2 Macro

A macro, in its simplest form, is defined as follows:

```
#define macro_namereplacement_text
```

Example:

```
#define N 10
```

The preprocessor replaces all occurrences of the word N with 10.

9.2 Types

9.2.1 Predefined Types

The following tables summarize the basic types in algorithms and their equivalents in C.

- Integers in algorithms from-∞ to +∞:

Types	Size(bytes)	Size(bits)	Range
char	1	8	$-2^{7}, 2^{7}-1$
short	2	16	$-2^{15}, 2^{15}-1$
long	4	32	$-2^{31}, 2^{31}-1$

int	4	32	$-2^{31}, 2^{31}-1$

Natural numbers in algorithms from 0 to +∞. Since integers contain natural numbers, we usually use integers to express them. You can also add unsigned before a type in C to express only natural numbers.

Types	Size(bytes)	Size(bits)	Range
unsignedchar	1	8	$0, 2^{8}-1$
unsignedshort	2	16	$0, 2^{16}-1$
unsignedlong	4	32	$0, 2^{32}-1$
unsignedint	4	32	$0, 2^{32}-1$

- Real numbers:

Types	Size(bytes)	Precision	Range
float	4	6	3.4×10^{-38} to $3.4 \times 10^{+38}$
double	8	8	1.7×10^{-308} to $.7 \times 10^{+308}$
Long double	10	8	1.7×10^{-4932} to $1.7 \times 10^{+4932}$

- **Boolean** type: There is no Boolean type in C, but int is used instead. True is represented by the number 1 and false by 0. Any number other than 0 is translated as true.
- Character type is char.
- String type: To express strings in C, we use arrays (Chapter 5) of char[] or pointers (second semester) of char*.

9.2.2 Notes

− The ***int*** type is the generic type for integers.

− The ***char*** type is used for both integers and characters, where each character is associated with a number.

− In C++, there's the ***bool*** type for Boolean and ***string*** for string.

− In this course, we use ***char*** for characters, ***int*** for integers and Boolean, and ***float*** for real numbers.

9.2.3 Type Conversion

- **Implicit**: This is done automatically by the compiler. It goes from a smaller type to a larger type without losing information. For example, char to int, int to float, or float to double. Converting 5 to float becomes 5.0.
- **Explicit** (casting): When the conversion could lead to losing information, it's necessary to declare that the programmer is performing the operation. You need to specify the destination type in parentheses before the expression to convert it. For example, **(int) 3.1416** converts it to 3.

9.2.4 Defining New Types

To create a new type or rename a specific type, the typedef keyword is used:

```
Typedef old_name new_name;
```

Example: To declare a new type named Banane with an underlying type of int, you use:

```
typedef int Banane;
```

9.3 Declaration of Variables and Constants

9.3.1 Declaration of Variables

Syntax:

```
Type variable;
```

example:

```
int age;
char gender;
float x, y, z;
Banane b;
```

9.3.2 Declaration of Constants

Syntax:

```
const Type Identifier = value;

or


Type const Identifier = value;
```

example:

```
int const N = 10;

const float Pl = 3.1415926;

const char[] DEP = "قسم الاعلام الالي";
```

Note: Macros can be used to declare constants as well, like:

```
#define DEP "قسم الاعلام الالي"
```

9.4 Assignment

We use = instead of ←, and read "receives" rather than "equals".

Syntax:

```
variable = expression;
```

example:

`a=5;`	`a gets 5`
`b=a*2;`	`b gets 10`
`a=0;`	`a gets 0`
`b=b-1;`	`b gets 9`
`c='b';`	`c gets the letter b`
`d=b>a;`	`d gets 1`
`s="name";`	`s gets the word name`

Declaration with initialization

```
        float x, y=3, z;//y is initialized with 3
```

Multiple assignments with right priority

```
        b=3;
        a=b=5+3;
```

b takes 8, then a takes the value of b, which is 8

Assignment shortcuts in C.

expression	comment	example

`v+=exp ; ⇔ v=v+(exp) ;`	Parentheses are important	`x=2 ;`
		`x*=5+3 ; ⇔ x=x*(5+3) ;`
		`≠ x=x*5+3 ;`
		`x become 16`

`v-=exp ; ⇔ v=v-(exp) ;`		
`v*=exp ; ⇔ v=v*(exp) ;`		
`v/=exp ; ⇔ v=v/(exp) ;`		
`v%=exp ; ⇔ v=v%(exp) ;`		

`v++ ; ⇒ v=v+1 ;`	If contained in another instruction, the calculation is performed with the current value of the variable, and after completion, 1 is added to or subtracted from the variable depending on the operation.	`x=2 ;`
		`y=3+x++ ;⇔ y=3+x ;x=x+1 ;`
		`In other words, y becomes 5 and x 3`

`v-- ; ⇒ v=v-1 ;`		

`++v; ⇒ v=v+1 ;`	If included in another instruction, 1 is added or subtracted according to the operation before the expression is calculated, using the new variable value.	`x=2 ;`
		`y=3+ ++x ;⇔ x=x+1 ;`
		`y=3+x ;`
		`In other words, y becomes 6 and x 3`

`--v ; ⇒ v=v-1 ;`		

Note:

- v++ is not identical to v+1, but v=v+1.
- v++, ++v, v=v+1, v+=1 are all equivalent if presented in a separate instruction.
- The difference between v++, ++v when it appears in another sentence, is the pre- or post-addition.

Empty instruction: It's an instruction that does nothing, like the semicolon instruction ""and instructions such as i + 1, where the result is calculated and ignored, without any change in the program state.

9.5 Input and Output

9.5.1 printf (print formatted)

The printf function, defined in the stdio.h library, is used to write formatted data to the standard output unit, which is typically the screen.

Syntax:

```
    printf(format,expression_1,…,expression_n);
```

format: A text or string that is displayed as is on the screen, except for the "%" symbol to indicate expression formats and the "\" character for escape sequences.

• **expression:** Computed to obtain a single value, which will be displayed on the screen in the specified format <format>.

The format follows this structure:

```
%[flags][width][.prec]type_char
```

where it always starts with %.

- flags
  - -: Left-align the output.
  - +: Show the sign of the number.
  - Empty: Display a space i stead of + for positive numbers.
- **width**: Represents the minimum number of digits to display for a specific value. If this number is greater than the required size, the difference is filled with spaces or zeros, depending on whether the number starts with 0 or not.
- **.prec**: The number of digits after the decimal point for floating-point numbers.
- **type_char**: A character representing the type of value to output.

format	use
%d	To input or output a number in the **d**ecimal system (10)
%o	To input or output a number in the **o**ctal system (8)
%x	To input or output a number in the he**x**adecimal system (16)
%u	To input or output an unsigned natural number **u**nsigned
%i	To input or output an integer (**i**nt) like %d
%f	To input or output a real number (**f**loat)
%c	To input or output a character (**c**har)
%s	To input or output a **s**tring (char[], char*)
%e	To input or output a nbr in scientific format such as 3**e**-2

– Some characters are special, so we must use an escape technique to use them. In C, the escape character is (), backslash, and we use it to add a new line '\n' or a tabulation (a large space) '\t'. To print double quotes ("), we use ", to print a backslash () we use \, and to print % we use %%.

example:

`printf("Hello");`	displays Hello

`int a=13;`
`printf("a=(%d)10\ta=(%o)8\ta=(%X)16\n",a ,a ,a);`
	a=(13)10 a=(15)8 a=(D)16

`a=66;`
`printf("a=%i\ta=%f\ta=%c\ta=%c\n",a ,a ,a ,a+32);`
	a=66 a=0.000000 a=B a=b
	Because 66 is not necessarily 66 in float
	And 66, if we see it as a character, represents the letter B, while 66 + 32 = 98 represents the coding of the letter b in lower case.

`float pi=3.1415926;`
`printf("%f\t%.4f\t%06.2f" ,pi ,pi ,pi);`
	3.141593 3.1416 003.14

9.5.2 scanf (scan formatted)

The **scanf** function, defined in the *stdio.h* library, is used to read formatted data fromthe standardinput unit, typically the keyboard. The function copies the value entered by the user to the variable's memory location. Therefore, & is used before the variable name.

Syntax:

```
scanf(format,&variable_1,…,&variable_n);
```

- **format**: A string representing the reading format.
- **variable**: The variable name. It's preceded by **&**, except for string variables (pointer types).

The format takes the following form:

% [width] **type_char**

- **width:** A number that controls the maximum number of characters to be read in the current input field.
- **type_char:** A character representing the type of value to be entered. The same symbols in the table for printf.

Example:

`scanf("%s",name);`	**&** is not used to read a string.
`scanf("%d%f",&a,&b);`	Enter a number, then press *space*, then enter the second number, then press *enter*.

Note: It is recommended to enter only one value per scanf instruction.

Avoiding problems with characters and strings in scanf

When reading a character using scanf("%c"...), the user enters the initial character and subsequently presses the Enter key. This action results in the creation of the '\n' character, which remains stored in memory. When the program encounters the scanf("%c", ...) instruction a second time, it doesn't await the user's input; instead, it directly assigns the '\n' character to the second variable.

To overcome this predicament, in the second scanf, we introduce a space ' ' after the % symbol, as demonstrated here: scanf("% c"...). Alternatively, we can utilize the getch function from the string.h library to resolve this issue.

Example: We'll try to enter 3 letters a, b, c in the variables c1, c2 and c3. We use:

```
scanf("%c",&c1);
scanf("%c",&c2);
scanf("%c",&c3);
```

The user presses a, then enter. The program assigns a to c1 and \n to c2, and waits for the user to enter the second character to be assigned to c3. To avoid this problem, we use:

```
scanf("%c",&c1);
scanf("%c",&c2);
 scanf("%c",&c3);
```

The issue with scanf arises when attempting to input a string containing spaces into a variable. For instance:

```
scanf("%s%s",v1,v2);
```

Upon entering the words "math info" and pressing Enter, the program assigns the first word to v1 and the second word to v2. However, if we intend to input both words, such as a compound name, into a variable, we use:

```
scanf("%s",v1);
```

Upon entering the words "math info" and pressing Enter, the program assigns only the first word to v1, while the second word is lost.

To circumvent this problem, we employ the **gets** function, defined in the **string.h** library:

```
#include<string.h>
gets(v1);
```

10. Structure of a C Program

```
#include<stdio.h>
  // Public declarations (constants, types and variables)
```

```
3.  Int main()
4.  {
5.    //Local declarations(constants and variables)
6.    //instructions
7.    return 0;
8.  }
```

explanation:

1. Include the stdio.h library which contains scanf and printf.
2. Place for public declarations.
3. main(): Every program must have a starting function called main which indicates the entry point of the program.
4. Start of the main function body, corresponding to the "begin" in algorithms.
5. Place for local declarations.
6. Instructions.
7. return: The main function must return an integer (int). It returns 0 to the operating system to indicate successful execution.
8. End of the main function and the program, corresponding to the "end" in algorithms.

Observations:

- Declarations can be made either in the place of public or local declarations.
- Code formatting, alignment, margins, spaces, and line breaks after special characters ([{}]) =+, ;: etc. are not significant in the program. Much of the program can be composed on a single line, with semicolons serving as separators between statements.
- Code formatting, alignment, margins, spaces, and line breaks should be used for program readability.

An example demonstrating the process of translating an algorithm into C.

algorithm	C	comment
`Algorithm circle_area`		The algorithm name becomes the file name circle_area.c
`Const P=3.14`	`Const float P=3.14;`	can use #define P 3.14
`Var r, s:entier`	`int r, s;`	There is no var word in C
`//r radius and s area`	`// r the radius and s the surface`	This is not part of the program, but only for explanation.
`begin`	`int main()` `{`	Variables can be declared after {.
`Write ("Enter radius ")`	`printf("Enter radius \n");`	\n to return to the line
`Read (r)`	`scanf("%d", &r);`	Don't forget & before the variable
`s←p*r*r`	`s=p*r*r;`	Each instruction ends with ;
`Write ("The area of the circle is:" , s)`	`printf("The area of the circle is: %d" , s);`	The format must be given
`End`	`}`	

example2

Write a program to calculate the average for ADS1.

```
#include <stdio.h>
int main() {
    float cont, td, tp, moy ;
    printf("Enter the exam score \n") ;
    scanf("%f", &cont) ;
    printf("Enter TD score \n") ;
    scanf("%f", &td) ;
    printf("Enter TP score \n") ;
    scanf("%f", &tp) ;
    moy = (cont * 3 + td + tp) / 5 ;
    printf("The average is %.2f" , moy) ;
    return 0;
}
```

University of Adrar

Faculty of Material Sciences, Mathematics and Computer Science

Department of Mathematics and Computer Science

Module: Algorithms and data structure 1

Promotion: 1st Year Math and CS **Academic year: 2024-2025**

Tutorial 01

Exercise 01:

a) Write an algorithm in natural language to prepare an omelette.

Exercise 02:

a) Write an algorithm in natural language to solve an equation of type $ax+b = 0$.

b) Write an algorithm to solve and find solutions to this equation.

Exercise 03:

Problem: given an equation of type $ax^2 + bx + c = 0$

a) Write an algorithm to solve and find solutions to this equation.

b) Adopt a similar approach to obtain an algorithm in tree form.

University of Adrar

Faculty of Material Sciences, Mathematics and Computer Science

Department of Mathematics and Computer Science

Module: Algorithms and data structure 1

Promotion: 1st Year Math and CS **Academic year: 2024-2025**

Tutorial 02

Exercise 01: Determine the error if it exists for each identifier 5TD, _3, bonne chance, TP, mathématiques, Δ, D-A, end, TP

Exercise 02: In this algorithm indicate the variables and give their type:

```
Algorithm ex2 ;
var  a, b, c : integer ;
begin
        a←5;
        b←12;
        c←2*a-b;
        b←2*b-c*3;
        a←b-a*4+c*5;
        write ('A=',a,' B=',b,' C=',c);
end.
```

What do these variables contain?

Exercise 03:

```
Algorithm ex3 ;
var  a, b, c : integer ;
begin
      read (a,b);
      a←a+b ;
      b←a-b;
      a←a-b;
      write('A=',a,' B=',b);
end.
```

Exercise 04: Give the type and result of the following expressions, in algorithm and in C language.

a) 5-3.*2+2

b) 10/5*5

c) (7+6) mod 5

d) 12 div 2 > 17 mod 5 *2

e) 1380 div 60 mod 60

f) 'h'>'Q' and 17>5

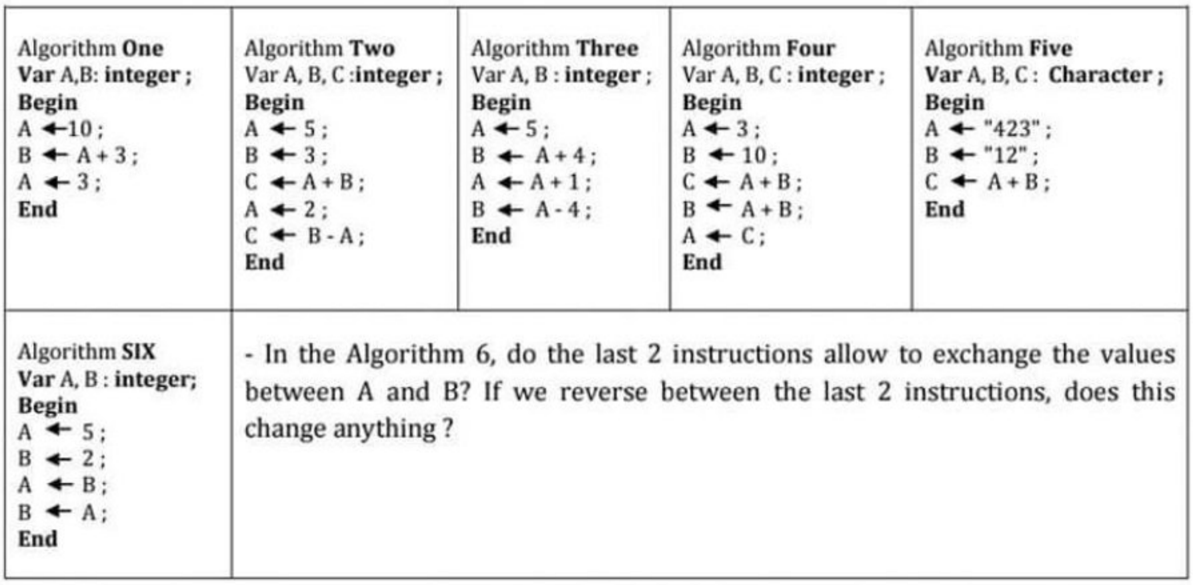
g) non ('h'>'Q')

Rewrite previous expressions in C language.

Exercise 05: Give the values of the variables after the execution of each instruction of this algorithm.

```
Algorithm Exo5 ;
var A, B: integer ;
 begin A←7 ;
B←A-4 ;
A←A-1 ;
B←A+5 ;
 End.
```

Exercise 06: What will be the values of the variables A,B and C after execution of the following instructions for each algorithm.



Algorithm **One**
Var A,B: **integer ;**
Begin
A ←10 ;
B ← A + 3 ;
A ← 3 ;
End
	Algorithm **Two**
	Var A, B, C :**integer ;**
	Begin
	A ← 5 ;
	B ← 3 ;
	C ← A + B ;
	A ← 2 ;
	C ← B - A ;
	End
		Algorithm **Three**
		Var A, B : **integer ;**
		Begin
		A ← 5 ;
		B ← A + 4 ;
		A ← A + 1 ;
		B ← A - 4 ;
		End
			Algorithm **Four**
			Var A, B, C : **integer ;**
			Begin
			A ← 3 ;
			B ← 10 ;
			C ← A + B ;
			B ← A + B ;
			A ← C ;
			End
				Algorithm **Five**
				Var A, B, C : **Character ;**
				Begin
				A ← "423" ;
				B ← "12" ;
				C ← A + B ;
				End

Algorithm **SIX**
Var A, B : **integer;**
Begin
A ← 5 ;
B ← 2 ;
A ← B ;
B ← A ;
End
	- In the Algorithm 6, do the last 2 instructions allow to exchange the values between A and B? If we reverse between the last 2 instructions, does this change anything ?			

Exercise 07: Give the values of the variables after execution.

```
Algorithm Exo7 ;
var A, B: integer ;
begin
A←7 ;
```

```
B←5 ;
A←B ;
B←A ;
end.
```

- Does this algorithm allow to exchange the values of A and B?

- Propose changes to exchange the values of A and B.

Exercise 08: Supposing we have three variables A,B and C.

- Write an algorithm transferring the value of A to B, The value of B to C and the value of C to A.

Exercise 09: Evaluate the following expressions

3*(2-5) / (3*2)-5 / 3*2-5 / 3+(5*3) / (3+5)*3 / 3+5*3 / 3+5/2

5+2 > 4 / 5+3>=7 AND NOT (5-3=8) / NOT (6+3<=5) OR (65-7=458-95)

Exercise 10: Let's consider the following algorithm:

```
Algorithm Ex10 ;
var a, b: integer ;
begin
 a←7 ;
 b←5 ;
 a←a * b ;
 b←a/b ;
 a←a/b ;
end.
```

- What does this algorithm do?

Exercise 11: Run this algorithm:

```
Algorithm assignment ;
Variables a, b, c, x, y, z : integer ;
 d, e, f, g : Boolean ;
 h, i : character ;
begin
 a ←2 ;   h ←'c' ;   b ←3 * a ;   c ←10 ;   i ←'r' ;   d ←(b - c) = a ;    e ←Not d ;   c ←b – c –
a ;
 f ←(c ≠ 12) and ( e ) ;    y ←c ;   x ←b ;   g ←h > i ;
end.
```

Chapter 3: Conditional structures

1. Introduction

A program consists of instructions. Most of these instructions are executed in the order in which theyappear. Once an instruction is executed, it moves to the next instruction (sequential). Instructions are often separated by a semicolon ";". However, some instructions modify the program flow, known as control structures. For example: conditional structures, loops, function calls, and unconditional jump instructions. The conditional structure comes in three types: the simple conditional structure (if then), the complex conditionalstructure (if then else), and the multiple-choice structure (switch).

2. Simple Conditional Structure "if then"

In programming, we often encounter cases where we need to decide whether an instruction should be executed or not based on whether a condition is true or false. For example, in a dessert recipe, you might findinstructions like: If you have almonds, add them to the recipe, or if you like lemons, add a little more. The program's execution flow will change as the input changes. To express conditions in programming, we use theif...then test, which is the simplest conditional instruction. It consists of two parts:

- Condition: A Boolean expression with a value of either true or false.
- Block of instructions: executed if the condition is true, or ignored if the condition is false.

2.1 Syntax:

Algorithm	C

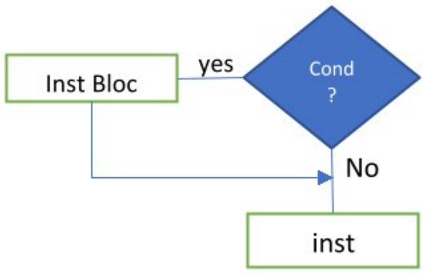
`if Condition then`
`  Block of instructions`
`EndIf`
`The rest of the instructions`
	`if (Condition)`
	`{`
	`   Block of instructions`
	`}`
	`The rest of the instructions`

The words "if," "then," and "EndIf" (or "fi") are reserved words in the algorithm. The same applies to "if" in C. In the algorithm, the condition is always between "if" and "then," while in C, it is always enclosed in parentheses (). To build the condition, we use comparison operations (>, <, =, ≠, ...) and logical operations(and, or, not, ...).

Instructions belonging to the "if" in C are enclosed in curly braces {} which can be omitted if they contain only one instruction. (Optional {}). If we find a set of instructions after the "if," and we don't find thecurly braces, then only the first instruction is associated with the condition, and the rest of the instructions willalways be executed regardless of the condition. However, if there is more than one instruction inside the curly braces {}, then both curly braces are mandatory.

Note:

- In C, the Boolean type is represented by an int. False is represented by 0, and true is any number otherthan 0.
- There is no ";" after }.



2.2 Flowchart

2.3 Execution

The execution process of the conditional instruction is performed by evaluating the condition, which results in a Boolean logical value. If the result is true, the block of instructions between "then" and "EndIf" in the algorithm, or between {} in C, is executed, followed by the rest of the program instructions. If the result is false, the instructions between "then" and "EndIf" are ignored, and the rest of the program instructions are executed directly.

2.4 Example

Write a program that reads an integer, then displays a warning if it's negative, and finally shows its square.

Algorithm	C	screen

```
algorithm
root var  x  :
integer**begin**
  write  ("Enter a number: ")
  read  (x)
  If x<0 then
   write  ("nbr is negative ")
  End If
  write  ("the square is "  , x*x)
end
```
	```
	#include <stdio.h>
	int main()
	{
	int x ;
	printf("Enter a number: \n");
	scanf("%d", &x) ;
	if ( x<0 )
	**{**// can be removed printf("nbr is
	                 negative
	   \n") ;
	}
	printf("the  square  is  %d" ,x*x)
	;
	}
	```
		```
		Enter a number
		2
		the  square  is 4

		*******
		Enter a number
		-3
		nbr    is     negative
		the  square  is 9
		```

3. The Complex Conditional Structure "if then else"

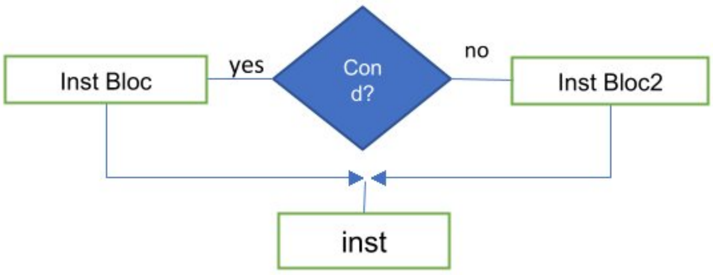
In a simple conditional, "if" specifies what to do if the condition is true, but not what to do if it's false. However, sometimes it's necessary to decide what to do in both cases. This leads to the "if else" (if...then...else) structure, which is an extension of the simple "if." The complex conditional structure "if else" consists of threeparts:

- Condition: A Boolean expression with a true or false value.
- First block of instructions: executed if the condition is true, or ignored if false.
- Second block of instructions: executed if the condition is false, or ignored if true.

3.1 Syntax:

Algorithm	C
`If Condition then`	`if (Condition)`
`  instruction block 1`	`{`
`else`	`   instruction block 1`
`  instruction block 2`	`}`
`End If`	`else`
`The rest of the instructions`	`{`
	`   instruction block 2`
	`}`
	`The rest of the instructions`

The word "Else" is a reserved word in the algorithm. The same applies to "else" in C. In C, {} can be omittedif it contains only one instruction. If there is a set of instructions after "if" or after "else," and curly braces are not found, it means that only the first instruction is related to "if" or "else."



3.2 Flowchart

3.3 Execution

Execution The conditional instruction is executed by evaluating the condition, which results in a Boolean value. If the result is true, the first block of instructions between "then" and "else" in the algorithm, or between {} before "else" in C, is executed, followed by the rest of the program instructions. If the result is false, the second block of instructions between "else" and "EndIf" in the algorithm, or between {} after "else" in C, is executed, followed by the rest of the program instructions.

3.4 Example

Write a program that calculates the absolute value of an integer and displays it on the screen.

Algorithm	C	Screen
algorithm absolute	**#include** <stdio.h>	Enter a nbr
var x, y : integer	**int** main(){	-5
begin	**int** x, y ;	|-5|=5
write ("Enter a nbr: ")	printf("Enter a nbr:\n") ;	
read (x)	scanf("%d", &x) ;	
if x>=0 **than**	**if** (x>=0)	
y←x	{ // can be deleted	**if** (x>=0)
else	y=x ;	y=x ;
y←-x	}	**else**
End If	**else**	y=-x ;
write ("|" , x ,"|=",y)	{ // can be deleted	
end	y=-x ;	
	}	
	printf(("|%d|=%d", x, y)) ;	
	}	

3.5 Conditional assignment in C

If we have a variable v takes one of the values v1 or v2 depending on condition b, i.e. :

```
if (b)
  v=v1 ;
else
  v=v2 ;
```

In this case, the “? : ” can be used and its

syntax is as follows:condition ?

expression_true : expression_false

- condition is a Boolean condition
- expression_true The expression returned if the condition is true.
- expression_false The expression returned if the condition is false

Example

```
v=b ? v1 :v2 ;
result = average >=10 ? " Admitted" : " Adjourned" ;
```

3.6 If-else extension

" If-else" can be used to test multiple conditions and to select the appropriate treatment for each case.For instance: to determine whether a student is accepted or not, there are several cases. Either they are accepted without compensation, or they are accepted with compensation, or they are accepted but with debts, or they are postponed. To determine this, one must examine the averages of the first and second semesters (s1 and s2), the annual average (MA), and the total earned credits (Crd).

The solution

Algorithm

```
algorithm absolute
var s1, s2, MA: real
   Crd : integer
begin
  write ("Enter first and second semester averages ")
  read (s1,s2)
  write ("Enter annual average ")
  read (MA)
  write ("Enter total credits ")
  read (Crd)
  If s1>=10 and s2>=10 then
   write("admitted without compensation ")
  else
   if MA>=10 then
    write("admitted with compensation")
   else
    if Crd>=45 then
     write ("admitted with debts ")
    else
     write ("adjourned ")
    end if
   end if
  end if
end
```

C

```
#include <stdio.h>
int main(){
float s1, s2, MA ;
int Crd ;
printf("Enter first and second semester averages \n") ;
scanf("%f%f", &s1, &s2) ;
printf("Enter annual average \n") ;
scanf("%f", &MA) ;
printf("Enter total credits \n") ;
scanf("%d", &Crd) ;
if ( s1>=10 && s2>=10 )
   printf("admitted without compensation") ;
else if ( MA>=10 )
   printf("admitted with compensation") ;
else if ( Crd>=45 )
   printf("admitted with debts ") ;
else
   printf("adjourned ") ;
}
```

4. The Multiple-Choice Conditional Structure "switch"

To choose an action among multiple options, we use the "switch" statement. However, when dealing withmore than two options, nested "if" statements can be used, resulting in nested "if" statements for each choice. This can make the program harder to read. The "switch" test is a special case of nested "if else" statements. Itdetermines which block of code to execute based on the value of the tested variable. It is used when we havemultiple outcomes and the condition is tested multiple times using the same variable. The "switch" statementis more readable and consists of:

- The expression to test, typically a variable.
- The values to test with corresponding blocks of instructions.
- An optional default block if there is no match with any value.

4.1 Syntax

Algorithm	C

`case expression of`
`val_1 : instruction block 1`
`val_2 : instruction block 2`
`…`
`val_n : instruction block n`
`else`
`  another instruction block`
`End case`
`The rest`
	`switch ( expression ) {`
	`case val_1 : instruction block 1`
	`   break ;`
	`…`
	`case val_n : instruction block n;`
	`   break ;`
	`default:`
	`   another instruction block`
	`}`
	`The rest`

The words "Case" "**else**" and "End Case" (or " EndCase") are reserved words in the algorithm. The sameapplies to "switch" "case" and "default" in C.

- **Expression:** An expression that calculates an integer or character value. It's typically a variable.
- **val_1, ..., val_n** : Values or constants of the same type as the expression.
- **Block of instructions**: One or more instructions executed if the expression value matches Value_i.

Note: switch is used instead of nested “if” we're going to test a single instruction or variable, of integer orcharacter type, several times with constant values.

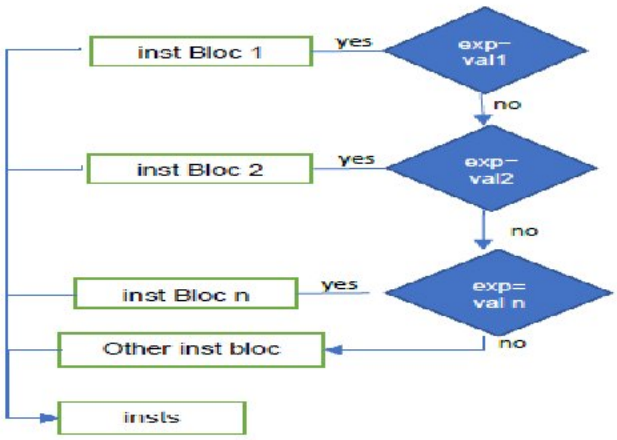
4.2 Rules regarding switch

- The curly braces {} of the switch and the parentheses () are necessary and cannot be omitted.
- Each Value_i must be different from the others. For example, writing "case 1" twice is illegal.
- Value_i can be placed in any order. However, it's recommended to order them in ascending order forbetter readability.
- A block of instructions can contain any number and type of instructions.

The "break;" statement is optional. It's used to exit a switch immediately, moving the program flowout of the switch.

- The default block is optional. If no Value_i matches, the execution context will move to the defaultblock. It should be the last case.

4.3 Flowchart



4.4 Execution

The "switch" statement is executed by evaluating the expression, then jumping to the block of instructions corresponding to the matched Value_i. After that block is executed, the execution will continue until it encounters a "break;" statement or reaches the end of the switch. If there is no match, the execution will move to the default block (if present), then continue with the rest of the program instructions.

The execution of "switch" in C slightly differs from the algorithm's "case" In C, after executing the block for Value_i, if no "break;" statement is encountered, the execution will continue with the subsequent block until a "break;" statement is reached. The execution will then move to the rest of the instructions outsidethe switch.

To make "switch" equivalent to "case algo" a "break;" statement should be added at the end of each block.

If multiple values share the same block of instructions, the algorithm can use a comma. In C, the first

value should not have instructions or a "break;" statement. Assuming values 7 and 9 have the same treatment:Algorithm:

Algorithm	C

`7 ,9: instruction block`	`case 7:`
	`case 9:`
	`   instruction block`
	`   break;`

4.5 Example

Write a program that reads an integer less than 10 and displays the corresponding English word on the screen.

Algorithm	C
`algorithm conversion;`	`#include <stdio.h>`
`var nb : integer;`	`int main(){`
`begin`	`int nb ;`
`  write ("enter a nbr ");`	`printf("enter a nbr \n") ;`
`  read (nb);`	`scanf("%d", &nb ) ;`
`  case nb of`	`switch ( nb ) {`
`  0 : write ("zero");`	`case 0 : printf("zero") ;`
`  1 : write ("one");`	`         break ;`
`  2 : write ("two");`	`case 1 : printf("one") ;`
`  …`	`         break ;`
`  9 : write ("nine");`	`…`
`  else`	`case 9 : printf("nine") ;`
`   write ("not treated");`	`         break ;`
`  End case`	`default:`
`end.`	`   printf("not treated") ;`
	`}`
	`return 0 ;`
	`}`

5. Branching Instructions

Branching is the process of moving between executed program instructions by the processor, where itperforms a "jump" to a specific address instead of continuing to execute instructions sequentially. There are four instructions in C that can unconditionally modify the execution flow of a program: **break, goto, continue,**and **return**.

5.1 Break Statement

We've already seen it with "switch," where it ends the "switch" instruction, moving the flow to the firstinstruction after "switch." In the case of a nested "switch," it only exits the immediate enclosing "switch." It'salso used to exit loops (covered in the next lesson). In this case, "break;" is usually within an "if."

Example

```
switch (grade){
  case 'A' :
  case 'a' : printf("excellent\n") ;
      break ;
  case 'b' : printf("good\n") ;
  case 'c' : printf("you can do better\n") ;
      break ;
  default : printf("try again\n") ;
}
```

- If grade contains the letter a or A, excellent.
- If it contains b ،it will appear good and you can do better
- If it contains the letter c, it only shows that you can do better.

- If it contains another character, try again.

5.2 Goto Statement

It transfers the program execution to a named instruction. This name, or "label," is preceded by a colon":". Any instruction can be named with a valid identifier followed by a colon.

Label syntax: `label : instruction;`

where label is a valid identifier. Such as:

```
here : printf("zero") ;
```

Syntax for calling: To access this instruction from anywhere, use the following syntax:

```
goto label ;
```

where "label" is the name of the instruction, e.g.: To access the instruction "here" from anywhere, use:

```
goto here ;
```

Note:

- "**case**" and "**default**" are special naming methods used within a "switch."
- **Goto** can be used to repeat instructions without the need for loops.
- It's advisable not to use "goto" and labels extensively, as it makes the program difficult to understandand maintain for humans.

Example:

again :

…

goto again **;**

5.3 continue Statement

It's used with loops to move the flow to the end of the loop and directly to the next iteration, withoutcompleting the loop instructions. It's typically within an "if."

Syntax `continue ;`

5.4 Return Statement

It's used to exit functions and return a result. (semester 2)

Syntax: `return expression ;`

Example: `return 0 ;`

As commonly used at the end of the main() function

University of Adrar

Faculty of Material Sciences, Mathematics and Computer Science

Department of Mathematics and Computer Science

Module: Algorithms and data structure 1

Promotion: 1st Year Math and CS **Academic year: 2024-2025**

Tutorial 03

Exercise 01:

Write an algorithm that reads the name and birth year of a person, as well as the current year. Then, it displays the age of that person.

Display example:

```
Name: Said
Year of birth: 2005
Current year: 2023
Hello Said, you are 18 years old.
```

Exercise 02:

Write an algorithm and its C program to calculate the average of the analysis module.

Exercise 03:

Write an algorithm and its C program that receives an angle in degrees, then displays this angle in grades and radians.

N. B. : rad = deg° × π/180 gr=π/200 rad

Exercise 04:

1) Write an algorithm that displays addition, subtraction, division and multiplication of real constants

2) What will happen if the two numbers are declared as variables which will be entered by keyboard?

Exercise 05:

1) Write an algorithm allowing you to exchange the values of two variables A and B, of the same type.

2) A variation of the previous one: we have three variables A, B and C. Write an algorithm transferring the value of A to B, the value of B to C and the value of C to A.

Exercise 06:

1) What does the following algorithm produce?

```
Algorithm ex6 ;
var A, B, C : char ;
begin
A ← "423" ;B ← "12" ; C ← A + B;
end.
```

2) What does the following algorithm produce?

```
Algorithm ex6 ;
var A, B, C : char ;
begin
A ← "423" ; B ← "12" ; C ← A & B ;
end.
```

Exercise 07:

Write an algorithm that reads the price excluding taxes (HT) of an item, the number of items and the VAT rate, and which provides the corresponding total price including taxes (TTC).

Exercise 08:

We consider the following algorithm:

```
Algorithm Operation;
Var A,B,C,D,E: boolean; X: Real;
Begin
Read(x); A ← (X > 5); B ← (X < 2); C ← (X > 0); D← (A AND B) OR C; E← A AND (B
AND C); Write (D); Write (E);
end.
```

For X=7, what are the values displayed by the algorithm?

University of Adrar

Faculty of Material Sciences, Mathematics and Computer Science

Department of Mathematics and Computer Science

Module: Algorithms and data structure 1

Promotion: 1st Year Math and CS **Academic year: 2024-2025**

Tutorial 04

Exercise 01:

Write an algorithm that allows entering 3 integers. Then, this algorithm only displays the even numbers.

Exercise 02:

Write a program that calculates the maximum between 2 numbers and another one that calculates the maximum between 3 numbers.

Exercise 03:

Write an algorithm with its C program that calculates the alms (aumône) or zakat. This algorithm receives a person's wealth along with the price of one gram of gold. Then, it displays the zakat amount. Knowing that the zakat rate is 2.5% and the Nisab threshold is 85 grams of gold.

Exercise 04:

Write an algorithm that reads a year A and informs us if this year is a leap year (February has 29 days) or not.

N.B.:

- If A is not divisible by 4, the year is not a leap year.
- If A is divisible by 4, the year is a leap year unless A is divisible by 100 and not by 400.

Exercise 05:

Write an algorithm that calculates the average of the analysis (exam and tutorials). Then, it calculates the final average, and if the average is below 10/20, it asks the user to provide the make-up grade. In this case, the calculation of the final average considers the better grade between the original exam and the make-up exam, and finally, the algorithm displays the final average.

Exercise 06:

Write an algorithm and its program for a minicalculator that offers the user to perform one of the following operations (addition of two numbers, subtraction of two numbers, division of two numbers, multiplication of two numbers, square root of a number, and power of a number).

Exercise 07:

Write a program that reads a character, and if it is a letter, it displays it in uppercase; otherwise, it displays it as it is.

Exercise 08:

Write an algorithm with its C program that allows reading two integers A and B and checks if A is divisible by B or not.

Exercise 09:

A store sells 3 products, p1, p2, and p3, with respective prices of 24 DA, 32 DA, and 43 DA. A discount of 1% is granted if the pre-tax amount exceeds 220 DA, and a discount of 2% is given if it surpasses 560 DA. Write an algorithm with its C program that reads the quantity purchased for each product, and then it displays: • The total price of each product, • The total pre-tax price, • The total pre-tax price after the discount, • The amount of VAT, knowing that the VAT rate is 19%, • The total amount to be paid.

Exercise 10:

A cinema offers the following rates for groups:

• 8 DA per place for the first 5;

• 6 DA per place for the following ones up to 10;

• 5.50 DA per place, those over 10.

1. The head of an association comes to buy tickets; how much will he have to pay for 4 places? for 9 places? for 15 places?
2. Write an algorithm to obtain the amount to pay when the number of places is given.
3. Test it for the values from question 1.

Chapter 4: Loops

1. Introduction

A loop is a control structure aimed at executing a set of instructions repeatedly multiple times. It is executed based on either a known number of iterations in advance (iterative loop) or until a condition allows the loop to exit (conditional loop). There are three types of loops:

- Conditional loop with a pre-condition: The condition is checked before the first iteration.
- Conditional loop with a post-condition: The condition is checked after the first iteration.
- Iterative loop: A counter keeps track of the number of iterations.

A programming error can lead to a situation where the exit condition is never satisfied. This results in the program running indefinitely, which is called an infinite loop.

2. The "While" Loop

The "While" loop is a pre-condition loop where a set of instructions is repeatedly executed based on a Boolean condition. The "while" loop can be seen as a repetition of the "if" statement. It is used when there's a set of instructions that need to be repeated with the possibility that they may not be executed at all (0 or more times), depending on the predefined condition. The loop consists of two parts:

- Condition: This is a logical expression with a value of either true or false.
- Instruction Block: It is executed as long as the condition is true.

2.1 Syntax

Algorithm	C

WhileCondition**Do**
  InstructionBlock
EndWhile
Restoftheprogram
	while(Condition**)**
	{
	  InstructionBlock
	}
	Restoftheprogram

"While" "**do**" and " **EndWhile**" are reserved keywords in algorithms. Similarly, in C, the term used is "while."

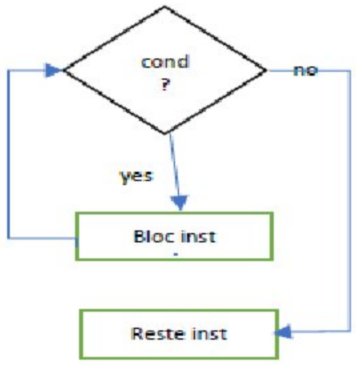
The condition is always placed between the keywords " **While** " and " **do**" in algorithms, while in C, it's always enclosed in parentheses. To create the condition, we use comparisons (>, <, =, ≠, ...) and logical operations (and &&, or ||, not !, ...).

In C, the instructions for the "while" loop are enclosed in curly braces {}, and they can be omitted if they contain only a single instruction (optional {}). If we encounter a set of instructions without the curly braces, it means that only the first instruction is repeated.

Notes:

- In C, the Boolean type is expressed using an **int**. False is represented by 0, and true is represented by any non-zero number.
- No ";" is needed after the closing curly brace "}".

2.2. Flowchart:



2.3 Execution

The execution process of the "While" loop begins by evaluating the condition expression, which results in a Boolean value. If the result is true, the instruction block between "Do" and "EndWhile" in the algorithm, or between the curly braces in C, is executed. The condition is re-evaluated, and the process repeats. When the test result becomes false, the loop exits, jumping to the instruction immediately following the loop. The condition is often referred to as the "continuation condition."

Notes:

Since the **While** loop checks the condition **before** the first iteration, it's possible that the condition isn't checked the first time, and so its instructions aren't executed at all.

2.4 Example

Write a program that reads two integers and then displays the quotient of the first divided by the second, without using the division operator (/ or div).

Note: Division is repeated subtraction.

Algorithm	C	Screen

`Algorithm quotient;`
`Var x,y,q,r:integer;`
`/*x first nbr, y second, q quotient, r remainder */`
`begin`
`  write("enter2nbrs") ;`
`  read (x, y);`
`  q←0;`
`  r←x;`
`while r>=y Do`
`      r←r-y ;`
`      q←q+1;`
`  Endwhile;`
`  write ("the quotient of", x, "on", y, "is", q, "the remainder is", r)`
`end`
	`#include<stdio.h>`
	`int main()`
	`{`
	` int x, y, q, r ;`
	` printf("enter 2 nbrs\n");`
	` scanf("%d%d", &x, &y) ;`
	` q=0;`
	` r=x;`
	` while (r>=y)`
	` {`
	`  r-=y;`
	`  q++ ;`
	` }`
	` printf("the quotient of %d over %d is %d the remainder is %d\n"), x, y, q, r) ;`
	`}`
		`enter 2 nbrs`
		`17 5`
		`the quotient of 17 over 5`
		`is 3 the remainder is 2`

The algorithm takes two numbers x and y, and returns the quotient q and remainder r.

At the beginning, let's assume that the remainder is x, and at each iteration we decrease the nominator "y" until it becomes less than the denominator. Each time we decrease "y", we add 1 to the quotient q. The **while** loop can never be executed if x is less than y from the start. In this case q=0 and r=x.

Example2

Write a program that doesn't stop until the user presses the Enter key.

```
#include<string.h>
int main()
{
while(getchar()!
='\n');    return
0;
}
```

3. The "Do...While" Loop

The "Do...While" loop is a post-condition loop where a set of instructions is repeatedly executed based on a Boolean condition. It's used when a set of instructions needs to be executed repeatedly at least once, regardless of the condition (1 or more times). The loop consists of two parts:

- Instruction Block: It's executed as long as the condition is true, except for the first time when it's executed regardless of the condition.
- Condition: A Boolean expression with a true or false value.

3.1 syntax:

Algorithm	C

`Do`
`  InstructionBlock`
`whileCondition`
`Restoftheinstructions`
	`do{`
	`  InstructionBlock`
	`}`
	`while(Condition) ;`
	`Restoftheinstructions`

The words "`Do`" and "`while` " are reserved keywords in algorithms and C. The condition always comes after "`while` " in algorithms and is always enclosed in parentheses () in C. To construct the condition, we use comparison operations (>, <, =, ≠, ...) and logical operations (and &&, or ||, not !, ...).

The instructions for "do…while" loops in C are enclosed in two curly braces {} within the "do" and "while" statements. The curly braces {} can be omitted if they contain only a single instruction (optional {}).

Observation :

- The "do…while" loop in C always ends with a semicolon ";".

The " do…while " construct can be expressed in algorithmic as "Repeat…Until" (until the condition is satisfied). In this case, the condition becomes a termination condition, not a continuation condition, which is the negation of the " while" loop condition.

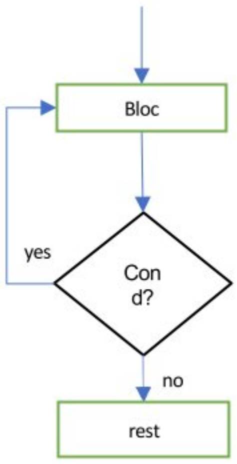
Example:

do…while	Repeat…Until

`Do`
`  Instruction block;`
`While x>y`
`The rest of the instructions;`
	Repeat
	`  Instruction block`
	`Until x≤y`
	`The rest of the instructions`

Note that the negation of > is ≤.

3.2 Flowchart:



3.3 Execution

The execution process of the conditional loop "Do...While" involves executing the instruction block between "**Do**" and "**While**" in the algorithm, or between "do" and "while" in C. After that, the condition expression is calculated, resulting in a Boolean value. If the result is true, the instruction block is executed again, and the process continues until the test result becomes false, at which point the loop exits, jumping tothe instruction immediately following the loop.

Observation :

Since the "**Do... While** " loop executes the instructions of the first iteration **before** the condition is verified, the loop executes at least one iteration, even if the condition is not satisfied from the start.

3.4 Example:

Write a program that reads a set of integers using a single variable, stops at the first 0 that reads it, then displays the number of integers entered.

Algorithm	**C**	**Screen**
`algorithm readNbrs;`	`#include<stdio.h>`	`enter a nbr:5`
`var x,nb:integer`	`Int main()`	`enter a nbr:7`
`/*xto read nbrs, nb to count`	`{`	`enter a nbr:-2`
`nbrs */`	`Int x,nb;`	`enter a nbr:0`
`begin`	`nb=0;`	`The number of`
`nb←0;`	`do{`	`numbers is 3`
`Do`	`printf("enter a nbr:");`	
`write("enter a nbr:");`	`scanf("%d", &x) ;`	
`read (x);`	`nb++;`	
`nb←nb+1;`	`}`	
`while x≠0`	`while(x!=0);`	
`write("The number of numbers`	`printf("The number of`	
`is", nb-1);`	`numbers is %d"), nb-1) ;`	
`end.`	`}`	

The algorithm needs variable x to read the numbers and variable nb to count the numbers. We set nb 0 as the initial value, then enter a number x and add 1 to nb. If x is 0, we stop, otherwise we repeat the loop until the user enters the number 0. The loop will run at least once. Finally, we show the value of nb-1 so that the number 0 is not counted.

4. The "for" loop

The "For" loop is an unconditional iterative loop where a set of instructions is executed iteratively a predetermined number of times. The loop consists of two parts:

- Counter: Used to count the number of iterations. It's a variable of integer or character type, with aninitial value, a final value, and a method of incrementing or decrementing.

- Instruction Block: Executed in each iteration.

4.1 Syntax (Algorithm)

Algorithm
For Counter←Initial_Value **To** Final_Value **Step** Step_Value **Do**   Instruction Block; **EndFor**; Rest of the instructions

The words **For**, **To**, **Step, Do** and **EndFor** are reserved words in the algorithm.

– `Counter`: A name for an integer or character variable.

– `Initial_Value`: This is the initial value taken by the counter variable.

– `Final_Value`: This is the final value the counter variable can take.

– `Step_Value`: This is the value of the counter variable at the end of each iteration. where `Counter ← counter+step`. Generally equal to 1.

Observations :

- The `Final_Value` termination value is calculated once before the loop is executed.
- The (Step `Step_Value`) part is optional and, in its absence, means that `Step_Value` is 1.
- If Step_Value is positive, it is added to counter until `counter ≥ Final_Value`. In the case of a negative `Step_Value`, it is decremented to `counter ≤ Final_Value`.
- `counter = Final_Value` is executed.
- If `Initial_Value` is greater than `Final_Value` and `Step_Value` is positive, the for loop is not executed.
- If `Initial_Value` is less than `Final_Value` and `Step_Value` is negative, the **for** loop is not executed.
- The `counter` value cannot be modified inside the loop.

4.2 Syntax« for» in C

The "for" loop in C is more general than the "for" loop in the algorithm. It's closer to the "**while**" conditional loop than to the " for " loop.

The general form	Its algorithme quivalent

for(initialization;test;iteration)
{
  Instruction block
}
The rest of the instructions
	for(Counter=Initial_Value;Counter<=
	        Final_Value; Counter += Step_Value)
	{
	  Instruction block;
	}
	The rest of the instructions

for is a reserved word in C.

The first line of for consists of three parts enclosed in parentheses (), all optional, separated by a semicolon ";".

− ***Initialization*** : This part is executed once before the loop is executed. It is generally used to assign an initial value to the counter. Ex : i=0

− ***condition***: a Boolean expression. Its value must be true to execute the loop. If the condition is false, the loop is exited. It is evaluated at the start of each loop iteration. Usually, the counter is tested. Like: i<10.

− ***Iteration***: It is executed at the end of each iteration. It is generally used to increment or decrement the counter. Like : i++ or i--.

C "for" statements are enclosed in {} . They can be omitted if they contain only one instruction ({} optional). If we find a set of instructions after for and we don't find the two braces, only the first instruction is repeated.

Notes:

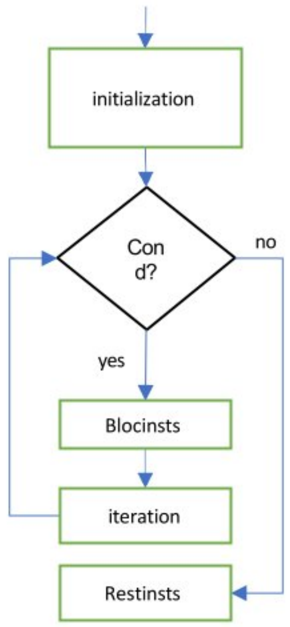
- The variable (the counter) can be declared in the initialization part, in which case the scope of its definition is only inside the for loop, not outside it.
- The counter value in the iteration section can be incremented or decremented, or modified in any other way.
- All "for" parts (initialization, test, iteration) are optional, and can be omitted and left empty. but ";" is mandatory and cannot be omitted. The following script is valid **for** (; ;)
- The initialization part and the iteration part can contain several instructions separated by commas ','.
- The instruction "; "is the empty instruction.

The following example codes are equivalents

```
Int i=0;
j=10;
for( ; ; ){
  if(!(i<j)
  )break;
  i++;
  j--;
}
```

```
for(int i=0,j=10;i<j;i++,j--);
```

4.3 Flowchart:



4.4 Execution:

The execution process of the "For" loop involves assigning the initial value to the counter variable. If the counter's value is less than or equal to the final value (for positive steps) or greater than or equal to the final value (for negative steps), the instruction block between "Do" and "EndFor" (algorithm) or between curly braces in C is executed. After each iteration, the counter is incremented or decremented by the step value. The process continues until the counter value no longer satisfies the condition, at which point the loop exits, jumping to the instruction immediately following the loop.

In C, the initialization expression is only executed once before the loop is executed. The command then passes to the condition. It is tested before each iteration. If the result is true, it executes the block of instructions between {} in C, then executes the iteration part, then re-evaluates the test and starts again. The iteration part is executed at the end of each iteration. When the test result becomes false, we exit the loop by jumping to the instructions immediately following it.

4.5 Example:

Write a program that reads two integers and then displays all the integers in between.

Algorithm	C	Screen
Algorithm numbers;	#include<stdio.h>	enter 2 nbrs
var x,y,i:integer;	**Int** main()	5 9
/*I is the counter*/	{	5 6 7 8 9
begin	**Int** x,y,i;	
write("enter2nbrs");	printf("enter 2 nbrs\n");	
read (x, y);	scanf("%d%d", &x, &y) ;	
fori←x **to** y **Do**	**for**(i=x;i<=y;i++)	
write(i);	printf("%d\t", i) ;	
Endfor;	**return**0;	
End.	}	

The algorithm takes two numbers x and y, and needs a variable i, which acts as a counter. Where it takes successive values from the interval x to y. At the end of each iteration, 1 is added to counter i. Since the step is implicitly 1. If x is greater than y, no number is displayed. In C, it must be written.

{} has been omitted from the for loop because it contains only one instruction.

5. Nested Loops

A loop can contain any type and number of instructions, including another loop. When a loop is inside another, it's called a nested loop. In nested loops, the execution proceeds as follows:

- Enter the outer loop
- Enter the inner loop
- Execute the inner loop until it's finished
- Return to the outer loop to execute the remaining instructions
- Repeat the process of executing the outer loop until it's finished.

Example

Write the program that reads the number of lines n, then displays on the screen in the first line *, in the second **, in the third ***, and so on until it displays in the last line n *.

Algorithm	C	Screen

Algorithm asterisk;
Var n,i,j: integer;
/*i,j counters*/
Begin
  write("enter no. of lines");
  read(n);
  for i←1 **to** n **Do**
    for j←1 **to** I **Do**
      write("*");
    EndFor;
  EndFor;
End.
	#include<stdio.h>
	int main()
	{
	 int n,i,j;
	 printf("enter no.of lines");
	 scanf("%d", &n) ;
	 for(i=1;i<=n;i++){
	  for(j=1;j<=i;j++)
	    printf("*") ;
	  printf("\n");
	 }
	 return0;
	}
		enter no.of lines
		5
		*
		**
		***
		****
		*****

The external for(i) contains two instructions: for(j) and printf("\n"). The internal for(j) contains a single instruction, printf("*"). printf("\n") is repeated n times. printf("*") is repeated 1+2+...+n times.

6. Loop Equivalence

- The "**While**" loop is used when the number of iterations is unknown in advance and when there's a possibility of not executing the instruction block at all.
- The "**Do...While**" loop is used when the number of iterations is unknown in advance, and the instruction block must be executed at least once.
- The "**For**" loop is used when the number of iterations is known in advance, or when the starting and ending values of the counter range are known.
- In general, any "**While**" loop can be expressed using "**Do...While**" by adding a condition before "**Do**," andany "**Do...While**" loop can be expressed using "**While**" by adding the instruction block before "**While**." Any "**For**" loop can beexpressed using "**While**" by initializing the counter before the loop, using the final value as the exit condition, and adding the instruction that modifies the counter's value at the end of the loop. However, it's not always possible to express "While" or "**Do...While**" loops using "**For**," unless there's a counter involved.
- In C, "**While** "or" **Do...While**" loops can be expressed with "**For**," and all loops can be expressed using "**Goto**" and "**If**."

Examples:

While

while	do…while	for	goto +if

`...`
`r=x ;`
`while ( r>y ) {`
`   r-=y ;`
`   q++ ;`
`}`
`printf(…) ;`
`}`
	`...`
	`r=x ;`
	`if ( r>y )`
	`  do {`
	`    r-=y ;`
	`    q++ ;`
	`  } while ( r>y )`
	`printf(…) ;`
	`}`
		`...`
		`r=x ;`
		`for ( ;r>y; ) {`
		`   r-=y ;`
		`   q++ ;`
		`}`
		`printf(…) ;`
		`}`
			`...`
			`r=x ;`
			`again :`
			`if ( r>y ) {`
			`   r-=y ;`
			`   q++ ;`
			`   goto again ;`
			`}`
			`printf(…) ;`
			`}`

do…while

do…while	while	for	goto +if

`...`
`nb=0 ;`
`do {`
` printf("entrer    un`
` nbr") ;`
` scanf("%d", &x) ;`
` nb++ ;`
`} while ( x!=0 ) ;`
`printf(…) ;`
`}`
	`...`
	`nb=0 ;`
	`printf("entrer     un`
	`nbr") ;`
	`scanf("%d", &x) ;`
	`nb++ ;`
	`while ( x!=0 ) {`
	` printf("entrer    un`
	` nbr") ;`
	` scanf("%d", &x) ;`
	` nb++ ;`
	`}`
	`printf(…) ;`
	`}`
		`...`
		`nb=0 ;`
		`printf("entrer    un`
		`nbr") ;`
		`scanf("%d", &x) ;`
		`nb++ ;`
		`for ( ;x!=0 ; ) {`
		` printf("entrer    un`
		` nbr") ;`
		` scanf("%d", &x) ;`
		` nb++ ;`
		`}`
		`printf(…) ;`
		`}`
			`...`
			`r=x ;`
			`again :`
			`printf("entrer       un`
			`nbr") ;`
			`scanf("%d", &x) ;`
			`nb++ ;`
			`if ( r>y )`
			`  goto again ;`
			`printf(…) ;`
			`}`

for

for	while	do…while	goto +if

`...`
`for (i=x;i<=y;i++)`
`  printf("%d\t",i);`
`...`
	`...`
	`i=x ;`
	`while ( i<=y ){`
	`  printf("%d\t",i);`
	`  i++ ;`
	`}`
	`...`
		`...`
		`i=x ;`
		`if ( i<=y )`
		`  do {`
		`    printf("%d\t",i);`
		`    i++ ;`
		`  } while ( i<=y )`
		`...`
			`...`
			`i=x ;`
			`again :`
			`if ( i<=y ) {`
			`  printf("%d\t",i);`
			`  i++ ;`
			`  goto again ;`
			`}`
			`...`

7. Loop Termination Commands

These commands are used within loops to perform an early exit from the loop based on a condition. They are generally used when checking a condition. Any "for," "while," or "do...while" loop can be terminated by executing any jump instruction like "break," "return," or "goto" (to a label outside the loop). The "continue" instruction only ends the current iteration and proceeds to the end of the loop, starting the next iteration. These instructions are often used within an "if" statement. In the case of nested loops, "break" and "continue" only exit the inner loop.

Example:

`for (int i=1 ;i<10 ;i++){`	`for (int i=1 ;i<10 ;i++){`
`  if(i%3==0) continue ;`	`  if(i%3==0) break ;`
`  printf("%d\t", i) ;`	`  printf("%d\t", i) ;`
`}`	`}`
All numbers will appear except for multiples of 3:	The loop stops at the first multiple of 3:
1 2 4 5 7 8	1 2

University of Adrar

Faculty of Material Sciences, Mathematics and Computer Science

Department of Mathematics and Computer Science

Module: Algorithms and data structure 1

Promotion: 1st Year Math and CS **Academic year: 2024-2025**

Tutorial 05

Exercise 01:

Write an algorithm with its C program that calculates the factorial of a number.

N.B. : $0!=1$ et $n!=1\times2\times\ldots\times n$

Exercise 02:

Write an algorithm to display all the common divisors of two numbers.

Exercise 03:

Write an program with its C program that determines if a number is prime or not.

- Using the for loop.
- Using the while loop.
- Generalize this program to display all prime numbers less than or equal to N ($\leq$N).

Exercise 04:

Write an algorithm/program in C that asks the user for a number between 1 and 3 until the answer matches.

Exercise 05:

Write an algorithm/program in C that asks for a starting number, and then displays the next ten numbers. For example, if the user enters the number 17, the program will display the numbers 18 to 27.

Exercise 06:

Write an algorithm / a program in C which requires a starting number, and which then writes the multiplication table of this number, presented as follows (case where the user enters the number 7) Table of 7:

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
…
7 x 10 = 70
```

Exercise 07:

Write an algorithm/program in C that requires a starting number, and that calculates the sum of the integers up to that number. For example, if you enter 5, the program must calculate:

1 + 2 + 3 + 4 + 5 = 15

NB: we only want to display the result, not the breakdown of the calculation.

Exercise 08:

Write an algorithm / program in C which successively asks the user for 20 numbers, and which then tells them which was the largest among these 20 numbers:

Enter the number number 1: 12

Enter the number number 2: 14

etc.

Enter the number number 20: 6

The largest of these numbers is: 14

Then modify the algorithm / program so that the program also displays in which position this number was entered:

It was number number 2

Exercise 09:

Rewrite the previous algorithm/program, but this time we do not know in advance how many numbers the user wants to enter. Number entry stops when the user enters a zero.

Exercise 10:

Write an program with its C program that calculates the GCD (Greatest Common Divisor). Given that:

$$PGCD(a,b) = \begin{cases} PGCD\big(b, (a \% b)\big), & b \neq 0 \\ a, & b = 0 \end{cases}$$

Exercise 11:

Write an algorithm to calculate the nth term of the Fibonacci sequence defined by:

$$u(n) = \begin{cases} 0 & si\ n = 0 \\ 1 & si\ n \quad 1 \\ u(n-2) + u(n-1), & si\ n > 1 \end{cases}$$

Exercise 12:

If you knew that

$$\pi = 4\sum_{k=0}^{n} \frac{(-1)^k}{2k+1} = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11}$$

Write a program that calculates the approximate value of π.

N.B. : make sure that n is strictly positive.

Exercise 13:

If you knew that

$$\exp(x) := \sum_{k=0}^{n} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} +$$

Write a program that calculates exp(x) (x is a real number and n is an integer).

N.B. : make sure that n is strictly positive.

Chapter 5: Arrays and Strings

1. Introduction

In programming, data is organized as constants and variables in certain ways to facilitate processing and quickaccess. There are different types of data, which can be divided into two parts: Simple types, such as integers, floats, characters, and Booleans. Composite types: arrays, structures, or records.

Let's say we want to input grades for 1000 students, analyze them, and calculate some statistics. In this case, it would be unreasonable to use 1000 variables to store grades and write 1000 input instructions in the program.It's better to use a single variable that can hold all the grade values and use a loop to input them. A structure that can store multiple values simultaneously is called an array.

In this chapter, we'll cover two types of static arrays: one-dimensional and multi-dimensional arrays. We'll also see that strings are a special case of arrays.

2. The Array Type

2.1 Definition

Array: A complex data structure consisting of a finite set of homogeneous elements (of the same type),accessible by indexes indicating their location.

An array can be seen as a group of variables of the same type with the same name.

Dimension : The dimension of an array is the number of indices needed to identify a single element.

Index: When data is stored in an array, the element is identified by an index which, in C, is a non-negativeinteger (≥ 0). The index ranges from 0 to N - 1 (where N is the array size).

One-Dimensional Array

It's also called a vector: you can access any of its elements using a single index, where each index value selectsan element from the array.

2.2 Representation

The array is represented in memory as a sequence of adjacent cells. A new cell cannot be removed or addedto the array after its creation (static). The following figure represents an array of 8 real numbers.

index	0	1	2	3	4	5	6	7
value	15	7	-3	0	9	2	0	-3

There's no difference in drawing the array vertically or horizontally.

2.3 Declaration

Algorithm	C
`arrayName [Size] : Array of elementType`	`elementType arrayName [Size] ;`

The phrase "Array of" is a reserved word in the algorithm, used to indicate that the variable is an array.

- `arrayName`: The identifier name given to the array (the variable name).
- `Size`: The number of elements in the array.
- `elementType`: The type of elements in the array, which can be of any type like

integer (int), float, ... To declare multiple arrays of the same type, use a comma "," while specifying the size of each array betweensquare brackets [].

Example

`const N=100`	`const int N=100 ;`
`marks [N] : array of real`	`float marks [N] ;`
`tab1[50],tab2[20] : array of integer`	`int tab1[50],tab2[20];`

2.4 Initialization

In C, you can specify initial values for all array elements using curly braces { and } during array declaration.Values are separated by commas ",", and these values must be of the same type.

Example

```
int tab[] = {15, 7, -2, 0, 9, 2, 0, -3};
```

index	0	1	2	3	4	5	6	7
value	15	7	2-	0	9	2	0	3-

Note: You can specify the number of elements between the two square brackets "[ ]", or leave them empty forautomatic calculation.

2.5 Usage

An array cannot be treated as a single block like array * 10; each element must be treated separately. To access a single element of the array, we use the array name with an index inside square brackets [and], and the expression inside the brackets evaluates to an integer value. To access the first element of array 'tab', we use tab[0]. To access the fourth element, we use tab[3].

```
tab[5-3]←tab[tab[3]+1]   ⇔   tab[2]←tab[0+1]   ⇔   tab[2]←7
```

the table becomes

Indice	0	1	2	3	4	5	6	7
valeur	15	7	7	0	9	2	0	-3

Note: Accessing an element that doesn't exist (if the index is greater than or equal to the array size or negative)will cause the program to terminate.

2.6 Reading an Array

To fill an array of N numbers, we use "read" N times like:

Algorithm	C
`Read (tab[0])`	`scanf("%d", &tab[0]);`
`read (tab[1])`	`scanf("%d", &tab[1]);`
…	…
`read (tab[N-1])`	`scanf("%d", &tab[N-1]);`

However, we notice that the read instruction is repeated **N** times, iterating from 0 to N-1. Therefore, the "**for**"loop can be used, with the counter playing the role of the index.

Algorithm	C

`for i←0 to N-1 do`
`write("nb", i, "⇒")`
`// Just to clarify`
`read(tab[i])`
`end for`
	`for(i = 0; i < N; i++){`
	`printf("nb %d ⇒", i);`
	`// Just to clarify scanf("%d",`
	`&tab[i]);`
	`}`

The "read" instruction is to fill the table, and the "write" instruction is to show the user what is required.

2.7 Displaying an Array

Like reading, "write" repeats.

Algorithm	C

`for i←0 to N-1 do`
`write(tab[i])`
`end for`
	`for(i = 0; i < N; i++){`
	`printf("%d\t", tab[i]);`
	`}`

2.8 Observations

- To visit all elements of an array, we generally use the "**for**" loop.

- The array size must be specified during programming (declaration), but we can give the user the impression that the array size can be changed by declaring a large array and using only part of it. We ask the user for the desired size, it must not exceed the actual array size.

2.9 Example

Write a program that receives the averages of N students, where N is determined by the user, then calculates the number of students who failed the subject (average less than 10).

Algorithm

```
Algorithm nb_adjourned;
Const MAX=200;
var avg[MAX] :array of real;
    i, aj, N : integer;
begin
do
  write ("enter number of students (<", MAX, ")");
  read (N);
while N>MAX
for i←0 to N-1 do
  write ("mark ", i, "⇒");
  read (avg[i]);
end for;
aj ←0;
for i←0 to N-1 do
  if avg[i]<10 then
    aj←aj+1;
  end if;
  end for;
write("the number of adjourned is ", aj);
end.
```

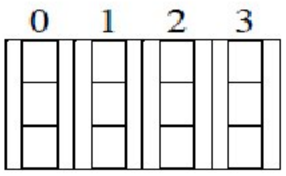
C

```
#include<stdio.h>
#define MAX 200
int main(){
float note[MAX] ;
int i, N, aj=0; // aj is nb of adjourned
// retrieve number of students
do{
  printf("enter number of students (<%d)",MAX) ;
 scanf("%d",&N) ;
}while (N>MAX) ;
// Fill in the table
for(i = 0; i < N; i++){
  printf("avg %d ⇒", i);
  scanf("%d", &note[i]);
}
// calculate number of adjournments
for(i = 0; i < N; i++)
  if(avg[i]<10) aj++ ;
// result display.
printf("the number of adjourned is %d", aj);
}
```

3. Multi-Dimensional Arrays

3.1 Definition

A two-dimensional array (also called a matrix) is essentially a simple array (one-dimensional) whose elementsare themselves one-dimensional arrays. We see this in the illustration below,



The elements are accessible via two indices, the first specifying the row number and the second specifying theelement number in that row (column).

This mechanism can be generalized to create matrices with more than two dimensions. We can create an n- dimensional array, and accessing its elements requires n indices. The arrangement of indices is crucial. The element M[3][2] (36) is different from the element M[2][3] (28).

3.2 Representation

The matrix is represented in memory by a sequence of adjacent cells. A cell cannot be removed or added to the matrix after its creation (static). The following figure represents a matrix with 3 rows and 5 columns of real numbers.

Column Number

Row Number	0	1	2	3	4
0	15	7	-3	0	9
1	6	12	4	33	85
2	2	-8	17	28	-52
3	14	42	36	49	-12

3.3 Declaration

Algorithm	C
matrixName[Rows][Columns] : **Array of** elementType	elementType matrixName [Rows][Columns] ;

The term "Array of" is a reserved word in the algorithm, used to indicate that the variable is an array.

- \`matrixName\`: The identifier given to the matrix (variable name).
- \`Rows\`: Number of rows.
- \`Columns\`: Number of columns.
- \`elementType\`: The type of elements. It can be any type, such as integer (\`int\`), float (\`float\`), ...

The number of elements is the product of the number of rows and the number of columns.

Example

`const R=100`	**`const`** `int R =100 , C=200 ;`
`const C=100`	**`float`** `M[R][C] ;`
`M[R][C]  :` **Array of** `real`	
`mat1[50][30],mat2[30][20] : tableau d'entier`	**`int`** `mat1[50][30],mat2[30][20];`

3.4 Initialization

In C, the initial values of all matrix elements can be specified by specifying the elements of each row between

\`{\` and \`}\`, along with the matrix declaration. The values are separated by commas \`,\`, and each row isseparated by a comma \`,\`. All values must be of the same type.

Example:

int mat[][] = **{**{15, 7, -3 ,0 ,9}**,**{6, 12, 4,33,85}**,**{2, -8, 17 ,28,-52}**,**{14, 42, 36, 49, -12}**}**;

column number

Row Number	0	1	2	3	4
0	15	7	-3	0	9
1	6	12	4	33	85
2	2	-8	17	28	-52
3	14	42	36	49	-12

Note: You can specify the number of rows and columns between the two brackets or leave them empty to becalculated automatically.

3.5 Usage

To access a single element of the matrix, we use the matrix name with an index inside two brackets \`[\` and \`]\`specifying the row number, and another index inside two brackets \`[\` and \`]\` specifying the column number. To access the element in the first row and first column of matrix \`mat\`, we use \`mat[0][0]\`.

Syntax

```
mat[line][column]
```

example

```
mat[1][3]←mat[1][3]+2
```

the matrix becomes

Column number

Row Number	0	1	2	3	4
0	15	7	-3	0	9
1	6	12	4	35	85
2	2	-8	17	28	-52
3	14	42	36	49	-12

3.6 Reading a Matrix

To fill the matrix `M[Rows][Columns]`, we fill in `Rows` rows. Since each row is a one-dimensional arraywith `Columns` elements.

Algorithm	C
`for j←0 to C-1 do`	`for (j=0 ;j<C ;j++)`
` read(M[0][j]);`	` scanf("%d", &M[0][j]);`
`end for;`	`for (j=0 ;j<C ;j++)`
`for j←0 to C-1 do`	` scanf("%d", &M[1][j]);`
` read(M[1][j]);`	…
`end for;`	`for (j=0 ;j<C ;j++)`
…	` scanf("%d", &M[L-1][j]);`
`for j←0 to C-1 do`	
` read(M[L-1][j]);`	
` end for;`	

However, we notice that the `for` loop is repeated `Rows` times. In other words, it iterates from 0 to `Rows -1`. Therefore, the outer `for` loop can be used.

Algorithme	C
`for i←0 to L -1 do`	`for(i = 0; i < L; i++)`
` for j←0 to C-1 do`	` for(j = 0; j < C; j++){`
`    read(M[i][j])`	`  printf("M[%d, %d] ⇒", i,j);`
`  end for`	`  scanf("%d", &M[i][j]);`
` end for`	` }`

The `read` statement is used to fill the matrix, and the `write` statement is used to explain to the user what isrequired. The `for(j ...)` loop contains two `printf` statements to explain and a `scanf` to enter values. The `for(i ...)` loop contains only one `for(j ...)` loop statement.Explanation of `write`: Let's assume `i = 3` and `j = 5`

write(	"M["	i	","	j	"]⇒"
screen	M[	3	,	5	]⇒

3.7 Displaying a Matrix

Similar to reading, the `write` statement is repeated.

Algorithm	C
`for i←0 to L-1 do`	`for(i = 0; i < L; i++){`
`for j←0 to C-1 do`	`  for(j = 0; j < C; j++)`
`    write(M[i][j]);`	`    printf("%d\t", M[i][j]);`
` end for;`	`  printf("\n") ;`
` end for;`	`}`

The `for(j ...)` loop contains the `printf` statement to print element `M[i][j]`. The `for(i ...)` loop contains two `for(j ...)` loops for printing row `i`, and `printf("\n")` to move to the next line at the end of each row `i` of thematrix.

Note: To visit all elements of the matrix, we use two `for` loops.

3.8 Example

Write a program that reads hourly temperatures for 30 days as a matrix (30 by 24), then displays them on thescreen. After that, display the highest temperature and when it was recorded.

Algorithm

```
Algorithm temperatures
Const Dy =30   Hr=24
var T[Dy][Hr] :array of real
    maxT :real
    i, j,maxDy,maxHr :integer
begin
for i←0 to Dy-1 do
 for j←0 to Hr-1 do
  write ("T[", i+1, ",", j, "]⇒")
  read (T[i][j])
 end for
end for
for i←0 to Dy-1 do
 for j←0 to Hr-1 do
 write  (M[i][j])
 end for
end for
maxT←T[0][0]
maxDy←0
maxHr←0
for i←0 to Dy-1 do
 for j←0 to Hr-1 do
  if (T[i][j]>maxT) then
   maxT←T[i][j]
   maxDy←i
   maxHr←j
 end if
end for
end for
write("the maximum temperature is ", maxT,"
   and was recorded on ", maxDy+1, " at ",
   maxHr )
end
```

C

```
#include<stdio.h>
#define Dy30 // nb lignes
#define Hr 24 // nb colonnes
int main(){
float T[Dy][Hr] ,maxT; // max température
int i, j, maxDy,maxHr;
// Fill in temperatures
for(i = 0; i < Dy; i++)
 for(j = 0; j < Hr; j++){
  printf("T[%d, %d] ⇒", i+1,j);
  scanf("%d", &T[i][j]);
 }
// display all temperatures
for(i = 0; i < Dy; i++){
 for(j = 0; j < Hr; j++)
   printf("%d\t", M[i][j]);
 printf("\n") ;
}
// search for maximum temperature
maxT=T[0][0];
maxDy =0 ;
maxHr=0 ;
for(i = 0; i < Dy; i++)
 for(j = 0; j < Hr; j++)
  if (T[i][j]>maxT){
   maxT=T[i][j];
   maxDy =i;
   maxHr=j;
  }
//display of results
printf("the maximum temperature is %d and
   was recorded on %d at %d", maxT, maxDy
   +1, maxHr) ;
}
```

The program takes the temperature matrix `Temperatures` and outputs the highest temperature recorded `maxTemp`, the day it was recorded `maxDay`, and the hour it was recorded `maxHour`. After filling the matrix and displaying it, we assume that the highest temperature is in row 0 and hour 0. Then we go throughall elements of the matrix, and if we find a temperature higher than the one stored in `maxTemp`, `maxTemp`changes, and so do `maxDay` and `maxHour`. In the end, we display the results on the screen, incrementing `maxDay` by 1 since rows start from 0 and days start from 1.

4. Strings

4.1 Definition

A string is an ordered set of characters, zero or more. They are always enclosed in double quotation marks `"`such as "computer", "Good luck\n", "1", "3.14". In C, character arrays are used to create strings. When you read a string from the keyboard, each character is placed in a location, and when the characters are finished, the character '\0' is added to the end of the text to indicate its end. The character '\0' is called "null," with a code of 0. There is a constant declared in the stdio.h library called **NULL** in uppercase.

```
#define NULL 0
```

Note: Since each character has a code, for example, the code of 'A' is 65, the code of 'a' is 97, similarly, the character '\0' has a code which is 0.

```
NULL ⇔ '\0' ⇔0
```

4.2 Declaration

In algorithms, we use the string, while in C, we use a character array. Suppose we have the string `str`, whichcan contain a maximum of 30 characters, including '\0'. It is declared as follows:

`var str : string`
 	
`var str[30] :array of characters`
	`char str[30] ;`

4.3 Initialization

In the following example, we create a character array and initialize it with the word "Welcome."

```
char greeting[] = {'W', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};
```

This statement creates an 8-character array (7 slots for the word "Welcome" and one slot containing the character '\0'). However, there is a simpler and faster way to create and initialize a string:

```
char greeting[] = "Welcome";
```

This leads to the same result, which is creating an 8-character array, ending with the character '\0'.

	0	1	2	3	4	5	6	7
greeting	W	e	l	c	o	m	e	\0

The size of the array can also be specified:

```
char greeting[30] = "Welcome";
```

	0	1	2	3	4	5	6	7	8	…	28	29
greeting	W	e	l	c	o	m	e	\0		…		

4.4 Assignment

Since strings are arrays, a string cannot be assigned to a variable directly after its declaration. The following operation is incorrect:

```
char slt[30];
~~slt = "Welcome";~~// error : assignment to an array.
```

To assign a string to a variable or copy one variable to another, we use the `strcpy()` function.

```
strcpy(slt , "Welcome" );
```

4.5 Displaying Strings

The `%c` format can be used to display the string character by character until we reach the `\0` symbol.

```
i←0
while str[i] ≠ '\0' do
   write (str[i])
   i←i+1
end while
```

```
for (i=0 ;str[i] != '\0' ;i++)
    printf("%c",str[i]) ;
```

The string can also be displayed directly using the %s format.

`write(str)`	`printf("%s",str) ;`

4.6 Reading Strings

The string can be directly entered using the `%s` format and without using `&` before the string's name.

`read(str)`	`scanf("%s",str) ;`

To enter text containing spaces, in C, we use the `gets` instruction, defined in the `string.h` library, because `scanf` stops at the first space.	`#include<string.h>`
	 …
	 `gets(slt) ;`

4.7 Some String-Specific Functions

C supports a wide range of functions that deal with strings, which are defined in the `string.h` library. Someof them are:

`strcpy(s1, s2);`	Copies string `s2` into string `s1`.
`strcat(s1, s2);`	Appends string `s2` at the end of string `s1`.
`strlen(s1);`	Returns the length of string `s1`.

`strcmp(s1, s2);`	Returns 0 if `s1` and `s2` are identical; lessthan 0 if `s1` < `s2`; greater than 0 if `s1` > `s2`.

4.8 Examples

Example 1:

An empty string str = "", which has a length of 0.

	0	1	2	3	4	5	6	…	28	29
str	\0							…		

The contents of the slots don't matter after `\0`, the string ends at the first `\0`. Thus, any string can be convertedto an empty string by placing `str[0] = '\0'`.

Example 2:

A string containing a single character, it is different from the character type. So, `"w"` ≠ `'w'` because `"w"`is an array.

	0	1	2	3	4	5	6	…	28	29
str	w	\0						…		

Example 3:

Write a program that takes text, then converts uppercase letters to lowercase and lowercase letters touppercase.

Algorithm

```
algorithm inverse
var txt :string[200]
    i  : integer
begin
 write(("enter text")
 read(txt)
 i←0
 while txt[i]≠'\0' do
   if txt[i]>='A' and txt[i]<='Z' then
    txt[i]=txt[i]+'a'-'A'
   else
    if (txt[i]>='a' and txt[i]<='z') then
      txt[i]=txt[i]-('a'-'A');
    end if
   end if
 end while
 write(txt)
end.
```

C

```
#include<stdio.h>
#include<string.h>
int main(){
 char txt[200] ;
 int i ;
 printf("enter a
 text") ;gets(txt)
 for(i=0 ;txt[i] !='\0' ;i++)
  if (txt[i]>='A'&&txt[i]<='Z')
   txt[i]+='a'-'A' ;
  else
   if (txt[i]>='a'&&txt[i]<='z')
     txt[i]-='a'-'A' ;

 printf("%s",txt) ;
 return 0 ;
}
```

University of Adrar

Faculty of Material Sciences, Mathematics and Computer Science

Department of Mathematics and Computer Science

Module: Algorithms and data structure 1

Promotion: 1st Year Math and CS **Academic year: 2024-2025**

<u>Tutorial 06</u>

Exercise 01:

Write an algorithm/program in C that fills an array T with n ($5<n<=10$) integers between 1 and 20. Then, it calculates and displays the sum, product, and arithmetic mean of the elements of T.

Exercise 02:

Write an algorithm/program in C that fills the T array with n letters ($2<n\leq20$). Then it displays, without redundancy, the elements of T.

Exercise 03:

Let T be an array containing N integers ($10\leq N\leq50$). We propose to write an algorithm/program in C that allows T to be split into two arrays: TN (containing the negative elements of T) and TP (containing the positive elements of T).

Exercise 04:

Let T be an array containing N integers ($10\leq N\leq50$). We propose to write an algorithm/program in C that allows the reverse of the elements of T (permute T[1] and T[n], then T[2] and T[n-1],...).

Exercise 05:

Let T be an array containing N integers ($10\leq N\leq50$). We propose to write an algorithm/program in C that allows grouping the even elements at the beginning and the odd elements at the end of T without changing the order of entry of the even and odd values.

Exercise 06:

Let T be an array containing N integers ($10\leq N\leq50$). We propose writing an algorithm/program in C that allows us to determine and display the maximum and minimum values of T.

Exercise 07:

Let T be a matrix containing N; M integers (N=10 and M=5). We propose writing an algorithm/program in C that allows us to determine and display T's maximum and minimum values.

Exercise 08:

Let T be a matrix containing N; M integers (N=10 and M=5). We propose to write an algorithm / a program in C that allows us to determine and display the sum of all the elements of T.

Exercise 09:

Write an algorithm / a program in C that allows you to enter a square matrix, and then it searches and displays its transpose.

Chapter 6 Custom Types

1. Introduction

In algorithms, in addition to predefined types (integer, real, Boolean, character, strings, and arrays), userscan define new types (custom types).

In our course, we are primarily interested in types such as enumeration and record.

2. Enumerations: (Enumerated Type)

- An enumerated type, as the name suggests, allows for the exhaustive definition of possible values.
- An enumerated type is defined by an identifier name and a range of values.
- Any enumerated type must be declared (defined) before its use.

2.1 Declaration:

Type NewType enum = (Value1, Value2, Value3, ...)

Example:

```
Type
 Day enum = (Saturday, Sunday, Monday, Tuesday, Wednesday, Thursday, Friday) Month
 enum = (January, February, March, April, May, June, July, August, September,
              October, November, December)
Variable
 J1, J2: Day;
 M: Month;
// Variables J1 and J2 can only take one of the values: Saturday, ..., Friday.
// Variable M can only take one of the values: January, ..., December.
```

Note:

- Constants in an enumeration are related by an **order** defined by the position of values in theenumeration. Thus, the order in which identifiers are listed is significant.

  Example: Saturday < Monday and December > January.
- Names assigned to different constants (values) in an enumeration cannot be reused.

  Example: Saturday: integer ; // error

2.2 Operations:

Functions defined on enumerated types include:

- Ord(x)`: This function returns a positive integer corresponding to the rank of x in the list.

- Succ(x)`: This function provides the constant immediately following the value of x in the enumeration.

  The successor of the last value is not defined.
- Pred(x)`: This function provides the constant immediately preceding the value of x in the enumeration.

  The predecessor of the first value is not defined.

Example:

Ord(Saturday) = 1, Ord(Sunday) = 2, ..., Ord(Friday) = 7.

Succ(Saturday) = Sunday, Succ(Sunday) = Monday, ..., Succ(Friday) = ? (not defined). Pred(Friday) = Thursday, Pred(Thursday) = Wednesday, ..., Pred(Saturday) = ? (not defined).

3. Structure type (or record)

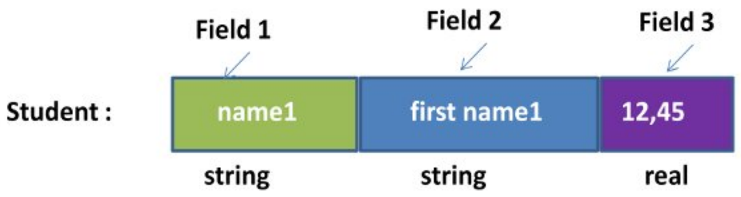
Issues : we want to safeguard students' averages :

For each student we need to store :

- The name (string)
- First name (string)
- The average (real)

Solution: We have to use a customised type of data! We define a new type and call it **student** containing:

- the name (string)
- First name (string)
- The average (real)



3.1 Definition

- A record (or structure) is a type used to store several items of data, of the same or different types.
- A record is made up of components called fields, each of which corresponds to a piece of data.

3.2 Declaration of a Record: Type

```
Structure RecordTypeName
  Field1: Type1;
  Field2: Type2;
```

```
        …..
        Fieldn: Type n;
      EndStructure;
```

- The keyword "Type" is common to all new types that one wants to add to the compiler.
- The keyword "Structure" indicates the beginning of the definition of a structure.
- "RecordTypeName" is the name of the new type.
- The keyword "EndStructure" indicates the end of the record.

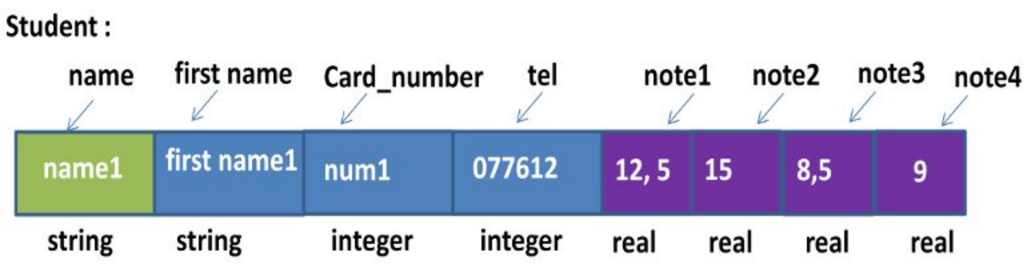
Example: Suppose that we want to write a program that manipulates information about students, including the name, first name, card number, phone number, and the 4 grades for each student.

The following declarations are necessary for this manipulation:

```
Type
  Structure StudentIdentity
    Name: String;
    FirstName: String;
    EndStructure;

Structure Student
    CardNumber: Integer;
    Identity: StudentIdentity;
    PhoneNumber: String;
    Grades[4]: Array of Real;
EndStructure;
//variables
student1, student2, student3 : Student;
```

The three variables *student1, student2, and student3* are of type *Student*, so each of these three variables hasthe following structure:



3.3 Handling records

a) Accessing fields in a record

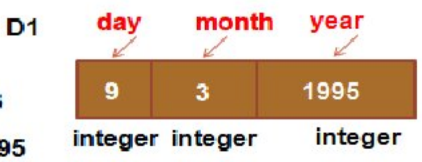
Access to a field is done by specifying the name of the record type variable followed by the field identifierseparated by a dot (.):

Example :

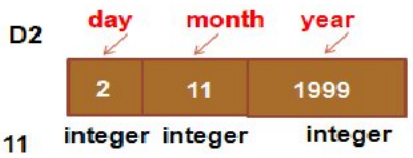
```
Type
Structure date
    day: integer;
    month: integer;
    year: integer;
End Structure ;

Variables
D1,D2: date;
```

```
D1. day = 9
D1. month = 3
D1. year = 1995
```



```
D2. day = 2
D2. months = 11
D2. year = 1999
```



b) Reading and writing records

By analogy with arrays, the fields of a structure are read or displayed one by one, because only the simpletypes defined by the compiler can be read or displayed.

Example : **reading information from a student**

```
……………………
Stud1 : student ; // Variable
Write('Please enter the card number:') ;
Read (Stud1.Card_number) ;
Write('Please enter the name:') ;
Read (Stud1.ident.Name) ;
Write('Please enter the first name:') ;
Read (Stud1.ident.Firstname) ;
Write('Please give the Telephone Number:') ;
Read (Stud1.Tel) ;
For i = 1 to 4 (step=1) do
    Ecrire('please enter the note N° ', i ) ;
    Read(Stud1.Note[ i ] );
```

```
End For
Write('Please give date of birth day month year' ) ;
Read(Stud1. date_birth.day);
Read(Stud1. date_birth.month);
Read(Stud1. date_birth.year);
………….
```

Writing:

- To display the variable "stud1" from the previous example, the procedure is as follows:

```
write (Stud1.Card_number) ;
write (Stud1.ident.Name) ;
write (Stud1.ident.Firstname) ;
write (Stud1.Tel) ;
For i = 1 to 4 (step=1) do
Write (Stud1.Note[ i ] );
End For
Write (Stud1. date_birth.day);
Write (Stud1. date_birth.month);
 Write (Stud1. date_birth.year);
```

Notes :

- Unlike arrays, there is no possibility of using a loop to manipulate all the elements of a record.
- In practice, the number of fields is very limited (5 to 20 fields).
- A record can be the subject of an assignment (with a variable of the same type):
- Stud2 = stud1; /* This means that all the fields of stud1 are copied to the
- corresponding fields of stud2 */

3.4 Arrays of Records:

It is possible to declare an array whose elements are of record type.

Thus, we first define the structure, and then declare the existence of an array whose elements are of thistype.

```
Type
Structure name_Typerecord
variable1 : type1;
variable2 : type2;
………
variablen : type n;
```

EndStructure

Name-Array[size] : ***Array of*** *name-Typerecord ;*

To select the third field of the fifth element of the array, the syntax is used as follows:

Name-array[5].variable3.

- **Reading:**

```
read (Tab[2].Name);
Tab[4].moyenne ← 10.5;
```

- **writing**

```
write (Tab[2].Name);
```

- **Comparison:**

```
if (Tab[2].moyenne < 10) then
```

University of Adrar

Faculty of Material Sciences, Mathematics and Computer Science

Department of Mathematics and Computer Science

Module: Algorithms and data structure 1

Promotion: 1st Year Math and CS **Academic year: 2024-2025**

Tutorial 07

Exercise 01:

Write a C program that defines a point structure containing the two coordinates of a point on the plane. Then, it reads two points and displays the distance between them. The distance between two points (x1,y1) and (x2,y2) is:

$$\text{Distance } = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Exercise 02:

Write a C program that defines a student structure where a student is represented by their first name, last name, and grade. Then, it reads a list of students entered by the user and displays the names of all students with a grade greater than or equal to 10 out of 20.

Exercise 03:

Complete the C code below to construct a deck of 32 cards. The deck is represented by an array of 32 distinct elements, each type of Card.

```
#include<stdio.h>
#include<stdlib.h>
enum Couleur {trefle,carreau , pique, coeur};
enum Face {sept , huit , neuf , dix , valet , dame, roi , as};
struct Carte
{
Couleur couleur;
Face face ;
};
int main()
{
Carte jeu [32] ;
// Code pour construire le jeu
...
}
```

Exercise 04:

Write a C program that reads a set of people of different ages into an array of structures and then deletes all those who are twenty years old and older.

Exercise 05:

Create a union that stores an array of 21 characters and six integers (6 since 21 / 4 = = 5, but 5*4 == 20, so you need 1 more for this exercise), and set the integers to the 6 given values, then display the array of characters as both a series of characters and a string.

References

1. Claude Pair, Marie-Claude Gaudel, Les Structures de données et leur représentation en mémoire, édition Iria, 1979.
2. Damien Berthet et Vincent Labatut. Algorithmique & programmation en langage C - vol.1: Supports de cours. Licence. Algorithmique et Programmation, Istanbul, Turquie. 2014.
3. D.M. Ritchie and B.W. Kernighan. The C programming language. Prentice Hall Inc., Englewood clis, New Jersey, Mars 1978.
4. Jacques Courtin, Initiation à l'algorithmique et aux structures de données, Edition DUNOD, 1998.
5. Mc Belaid, Algorithmique et Structures de données, Edition les pages bleus, Edition Mai 2008.
6. Mc Belaid, Initiation à l'algorithmique, Edition les pages bleus, Edition 2012.
7. Michel Divay, Algorithmes et structures de données génériques - 2ème édition, Edition Dunod
8. https://elearning.univ-msila.dz/moodle/course/view.php?id=10217&lang=fr
9. Thabet Slimani, Programmation et structures de données avancées en langage C, cours et exercices corrigés, 2014.
10. Thomas H. Cormen, Algorithmes Notions de base Collection : Sciences Sup, Dunod, 2013.
11. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest Algorithmique - 3ème édition - Cours avec 957 exercices et 158 problèmes Broché, Dunod, 2010.
12. Zegour Djamel eddine, Structures de données et de fichiers. Programmation Pascal et C, édition CHIHAB.
13. Paul J. Deitel and Harvey Deitel, C++ How to Program (10th Edition). Pearson Education, 2016.
14. Bjarne Stroustrup, C++ Programming Language, Addison-Wesley Professional, 2013.
15. https://cplusplus.com/doc/tutorial/
16. https://en.wikiversity.org/wiki/C%2B%2B/Introduction