الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة محمد بوضياف – بالمسيلة
Mohamed BOUDIAF University – M'Sila

جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila

كلية التكنولوجيا
Faculty of Technology

نيابة العمادة لما بعد التدرج والبحث العلمي والعلاقات الخارجية
Vice Deanship of Post-Graduation, Scientific Research and External Relations

رقم: 1416 ن.ع.ب.ع /ك.ت /2025

المسيلة في: 02 ديسمبر 2025

شهادة ادارية

المصادقة على تقارير خبرة للموافقة على مطبوعة بيداغوجية

بعد الإطلاع على تقارير لجنة الخبراء للموافقة على المطبوعة البيداغوجية للأستاذ : **بلوطي عادل– أستاذ محاضر قسم أ**، بالقاعدة المشتركة بكلية التكنولوجيا بجامعة محمد بوضياف بالمسيلة والتي كانت كلها ايجابية ، تمّ تقرير التّالي:

1–المصادقة على تقارير لجنة الخبراء للموافقة المطبوعة البيداغوجية والمعنونة بـــ:

Cours d'informatique 1

1ère Année ST

2– حيث تمّ تشكيل هذه اللجنة بناء على اجتماع المجلس العلمي للكلية المنعقد بتاريخ **2025/10/07** المكونة من السّادة الآتية أسماؤهم:

– **بوخالفة عبدالوهاب**، أستاذ محاضر "أ"، جامعة محمد بوضياف– المسيلة

– **خطاب خثير**، أستاذ، جامعة محمد بوضياف– المسيلة

– **بلهوشات نوري**، أستاذ محاضر "أ"، جامعة فرحت عباس– سطيف 1

وتمت الموافقة بالاجماع على هذه المطبوعة.

رئيس المجلس العلمي للكلية

أ. علي جريوي

Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

University Mohamed Boudiaf of M'sila

Faculty of Technology

Department of Electronics

Pedagogical handout of:

informatics 1

Level: First Year LMD

Common Base Technology – ST -

Elaborated By: Dr. Ballouti Adel

MCA - Electronics département

Email : Adel.ballouti@univ-msila.dz

Academic Year: 2023/2024

Semestre: 1
Unité d'enseignement: UEM1.1
Matière 3: Informatique1
VHS: 45h00 (Cours: 1h30, TP: 1h30)
Crédits: 4
Coefficient: 2

Objectif et recommandations:

L'objectif de la matière est de permettre aux étudiants d'apprendre a programmer avec un langage évolue (Fortran, Pascal ou C). Le choix du langage est laisse a l'appréciation de chaque établissement. La notion d'algorithme doit être prise en charge implicitement durant l'apprentissage du langage.
Les TP ont pour objectif d'illustrer les notions enseignées durant le cours. Ces derniersdoivent débuter avec les cours selon le planning suivant :

- TP's initiatiques de familiarisation avec la machine informatique d'un pointde vu matériels et systèmes d'exploitation (exploration des différentesfonctionnalités des OS)
- TP's d'initiation a l'utilisation d'un environnement de programmation (Edition, assemblage, compilation etc.)
- TP's applicatifs des techniques de programmation vues en cours.

Contenu de la matière:

Chapitre 1. Introduction à l'informatique (5 Semaines)

1- Définition de l'informatique
2- Evolution de l'informatique et des ordinateurs
3- Les systèmes de codage des informations
4- Principe de fonctionnement d'un ordinateur
5- Partie matériel d'un ordinateur
6- Partie système
Les systèmes de base (les systèmes d'exploitation (Windows, Linux, Mac OS,...)
Les langages de programmations, les logiciels d'application

Chapitre 2. Notions d'algorithme et de programme (7 Semaines)

1- Concept d'un algorithme
2- Représentation en organigramme
3- Structure d'un programme
4- La démarche et analyse d'un problème
5- Structure des données
Constantes et variables, Types de données
6- Les operateurs
L'operateur d'affectation, Les opérations arithmétiques, Les operateurs relationnels, Les operateurs logiques, Les priorités dans les opérations
7- Les opérations d'entrée/sortie
8- Les structures de contrôle
Les structures de contrôle conditionnel, Les structures de contrôle répétitives

Chapitre 3 Les variables Indicées (3 Semaines)

1- Les tableaux unidimensionnels
Représentation en mémoire, Operations sur les tableaux
2- Les tableaux bidimensionnels
Représentation en mémoire, Operations sur les tableaux bidimensionnels

Mode d'évaluation:

Contrôle continu: 40% ; Examen: 60%.

FOREWORD

The handout is dedicated to the first year students of the ST Common Core. This is a handout of an introductory computer science course with basic concepts allowing students to learn how to program advanced languages, namely Turbo Pascal.

The first part is devoted to an introduction to computer science, in this introduction, the soft part and the hard part of a microcomputer as well as the representation of information systems were discussed.

The second part is devoted to the concepts of ALGORITHM & PROGRAM to design the latter with its flowchart representations, the design of a program, the data structure and the resolutions of fundamental problems related to algorithms. As well as, the concepts of the different types of operators, the different existing control structures, namely the conditional and repetitive control structures, and the knowledge of indexed variables containing the types of tables (matrix). This part also contains exercises with detailed solutions.

Chapter 1:
INTRODUCTION TO COMPUTER SCIENCE

Summary

I.1. Definitions:

I.1.1. Informatics: The term "informatics" first created in 1962 by "Philippe Dreyfus" to designate the automatic processing of information using computers.

I.1.2. Computer: Programmable electronic machine that executes instructions organized in program.

I.1.3. Areas of use: The microcomputer has revolutionized the world through its use. Thanks to advances in science, its use is now spreading to several areas of activity.

The main areas of application of the microcomputer are:

- The military domain: during surveillance of a territory or during air and maritime navigation.
- The civil domain: which has three parts which are the economic domain, the communication domain and the scientific domain**.**

I.2. History and technological advances of computers:

- Generation 0: From the 17th century to 1945

Mechanical calculators.

- First generation: 1945 – 1955

These are the first electronic computers (calculators) built on the basis of vacuum tubes. Example: ENIAC.

- Second generation: 1955 – 1965

Computers of this generation were made based on transistors which replaced vacuum tubes. These computers are 100 times faster than those of the 1st generation and consume less electrical energy and are less bulky (take up less space).

First commercial series of computers

Examples:

– IBM 7030 (1961).

- Third generation: 1965 – 1980

This generation uses integrated circuits which make it possible to place a large number of transistors on the same silicon chip; it is characterized by increasing power and miniaturization.

Example: Intel 4004 (in 1971) First calculation unit (on 4 bits) integrated entirely on a single chip.

First microprocessor.

Performance identical to ENIAC.

A size of less than 11mm2.

2300 transistors, 740 kHz.

90 000 opérations par seconde.

- Fourth generation: 1980 to the present.

More power and miniaturization at an ever lower cost. This generation is characterized by the integration of thousands to billions of transistors on the same chip through the use of (VLSI / ULSI: Very / Ultra Large Scale Integration).

Example:

– MICRAL 8008 (1973).

– ALTAIR 8008 (1973).

– APPLE1 (1976).

– IBM PC (1981).

– PENTUIM etc...

Oracle Sparc M7 (2015), 32 cores, 10 billion transistors, Frequency of 4.13 GHz.

- Fifth generation: Quantum computers?

I.3. Information coding systems

Information is any form of data representing one or more pieces of information. We generally distinguish different types of information: texts, numbers, sounds, images, etc. Information in its raw form cannot be directly processed and analyzed by a logical machine such as computer.

So to remedy this problem, we translated it into an alphabet specific to the latter. This is the coding operation.

Coding of information:

The coding of information consists of establishing a correspondence between the external (usual) representation of the information (e.g. the character A or the number 36), and its internal representation in the machine, which is a sequence of 0 and 1.

b). Transcoding of information: is the passage from a code to another one for the same information.

I.3.1. Number Systems:

In general, numbers may be represented in different numeration systems. The decimal system is commonly used in routine transactions while the binary system is the basis for digital electronics. Every number (or numeration) system is defined by a base (or radix), which is a collection of distinct symbols. The representation of a number in a numeration system may be considered as a change in base. In a positional number system, a value of a number depends on the place occupied by each of its digits in the representation.

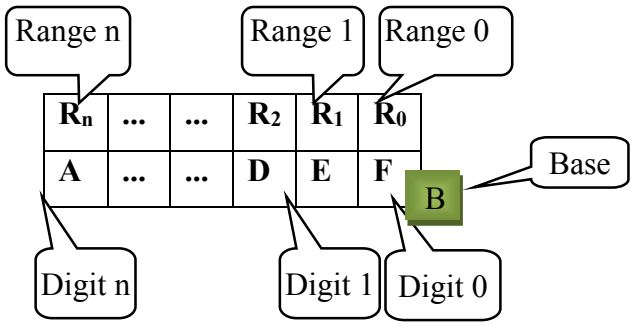
Three concepts are used in a system numbers:

• The base B of the system, the base **B** can be any **integer**.

• The digits of the system: which are all different characters; there are therefore B in total.

• The weight of the digit according to its range.

Generally speaking, any base N is composed of N digits from 0 to N-1.



I.3.1.1. Decimal Numbers (DEC)

The decimal number system uses the following 10 numbers or symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. the radix or the base is thus 10.

Example:

Decompose the numbers 1234 into powers of 10. N = 1234.

The decomposition of the number 1234 takes the form:

$$N = (1 * 1000) + (2 * 100) + (3 * 10) + (4 * 1)$$
$$= (1 * 10^3) + (2 * 10^2) + (3 * 10^1) + (4 * 10^0)$$
$$= 1000 + 200 + 30 + 4$$

Depending on its position, each number is multiplied by the appropriate power of 10. Regardless of the base, the digit on the right is the one with the lowest weight, and the digit on the left is the digit with the highest weight.

I.3.1.2. Binary numbers (BIN):

The binary number system uses the following 2 numbers or symbols: 0, 1. it is a system with a radix of two.

Binary number system is based on two-level logic, conventionally noted as 0 (low level) and 1 (high level).

- The coefficients or numbers (0 or 1) used in the binary representation of a number are called **BIT**S (**B**inary digit**)**.
- An 8-bit word is called BYTE.
- The binary code that is then obtained for a positive number is called a natural binary code.
- The right-most bit is called the **L**east **S**ignificant **B**it (LSB), while the left-most bit is called the **M**ost **S**ignificant **B**it (**MSB**).

Example:

Convert the binary number 1101101 into decimal number.

The decomposition of the number 1101101in powers of 2 is written as:

+	2^6 *	+	2^5 *	+	2^4 *	+	2^3 *	+	2^2 *	+	2^1 *	+	2^0 *
	1		1		0		1		1		0		1

64	32	16	8	4	2	1
2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	1	0	1

Most **S**ignificant **B**it (MSB)

Least **S**ignificant **B**it (LSB)

$(1101101)_2 = 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

$= 1*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 1*1$

$= (109)_{10}$

I.3.1.3. Octal Numbers

In the octal number or base eight 8 representation, we have the following eight symbols: 0, 1, 2, 3, 4, 5, 6, 7.

Example: In radix 8 representation, the number 345 takes the form:

$(345)_8 = 3 * 82 + 4 * 81 + 5 * 8^0$

$(345)_8 = 3 * 64 + 4 * 8 + 5 * 1$

$(345)_8 = (229)_{10}$

I.3.1.4. Hexadecimal Numbers (HEX):

In hexadecimal number system, we have 16 symbols: 10 digits 0, 1..9 and 6 symbols A, B, C, D, E, F. So in hexadecimal system, the base B equal to 16. 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; A;B;C;D;E; F With: A = 10, B = 11, C= 12, D= 13, E =14 and F= 15.

Example:

$(ABC)_{16} = A * 16^2 + B * 16^1 + C * 16^0$

$= A * 256 + B * 16 + C * 1$

$= 10 * 256 + 11 * 16 + 12 * 1$

$= (2748)_{10}$

I.3.2. Transcoding:

A. Conversion Octal to Binary:

We can observe that 8 is equal to 2^3;

We can therefore correspond to each digit of a number expressed in octal a set of 3 bits of the same

number expressed in binary.

Example:

$(6573)_8 = (??????)_2$

$(6573)_8 = ((110)(101)(111)(011))_2$

$=(110101111011)_2$

Octal number	**6**	**5**	**7**	**3**
	4 2 1	4 2 1	4 2 1	4 2 1
Binary number	**1 1 0**	**1 0 1**	**1 1 1**	**0 1 1**

B. Conversion Binary to Octal:

Octal numeration may be deduced from binary numeration by grouping, beginning from the right, consecutive bits in triplets.

Example:

Determine the radix 8 representation for the binary number N= $(1011001100100110011111)_2$.

N= $(1011001100100110011111)_2 = (????????)_8$

Solution:

Radix 8 representations are obtained by replacing each group of three bits by the equivalent octal number. We can therefore write:

By decomposing the binary number N, in increments of three, starting from the right we obtain:

N(Bin)	**1 0 1**	**1 0 0**	**1 1 0**	**0 1 0**	**0 1 1**	**0 0 1**	**1 1 1**
	4 2 1	4 2 1	4 2 1	4 2 1	4 2 1	4 2 1	4 2 1
N(Oct)	**5**	**4**	**6**	**2**	**3**	**1**	**7**

N = $(101\ /100/\ 110/\ 010/\ 011/\ 001/\ 111)_2 = (5462317)_8$

5	4	6	2	3	1	7

C. Hexadecimal to Binary conversion:

In the same way as before, we can observe that 16 is equal to 2^4. We will therefore correspond to each digit of a hexadecimal number a 4 bits of the corresponding binary number.

Example:

Determine the radix 2 representation for the HEX number N= $(6A28)_{16}$.

N= N= $(6A28)_{16}$. = $(??????)_2$

Solution:

$(6A28)_{16}=(?????)_2$

N(Hex)	6				A				2				8			
	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
N(Bin)	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	0

$(6A28)_{16}$= ((0110/1010/0010/1000))$_2$ =$(0110101000101000)_2$

6	A	2	8

D. Binary to Hexadecimal Conversion:

The reverse conversion, binary to hexadecimal, is done by decomposing the binary number by a set of 4 bits from the right and matching each set to its corresponding hexadecimal.

Example:

Determine the radix 16 representation for the BIN number N= $(1111111010000)_2$.

Solution:

By decomposing the binary number N, in increments of 4 BITs, starting from the right we obtain:

$(1111111010000)_2 = (????????)_{16}$

N(Bin)	0001				1111				1101				0000			
	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1
N(Hex)	1				F				D				0			

$(1111111010000)_2 = (0001/1111/1101/0000)_2$

1	F	D	0

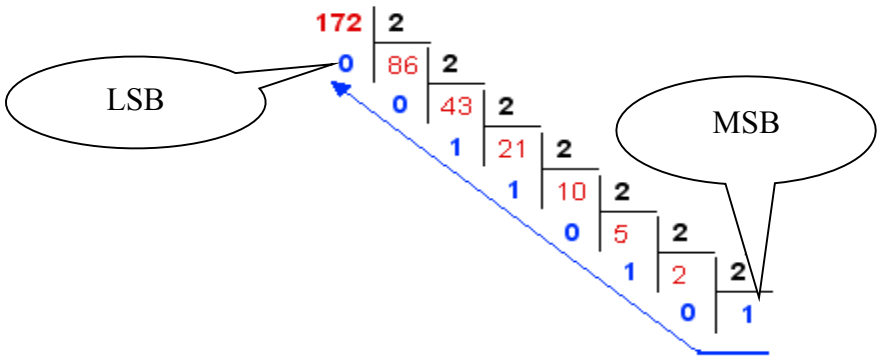
$= (1FD0)_{16}$

E. Decimal to any other bases Conversion.

The conversion of a decimal number into a binary, octal or hexadecimal number is based on finding multiples of the successive powers of the base (2, 8 or 16 depending on the case).

The practical method consists of carrying out successive divisions: of the decimal number by the target base, then of the quotient obtained by the target base, then of the new quotient by the target base… until the quotient becomes zero. The obtained result consists of all the successive remainders of the divisions, read backwards.

Example: Find the binary representation of the decimal number 172.



$(172)_{10} = (10101100)_2$

Observation

The same method would be applicable for conversions:

- Decimal to Octal (successive divisions by 8).
- Decimal to Hexadecimal (successive divisions by 16).
- The following table gives a summary of the BIN - DEC - HEX conversion and vice versa on 4 bits.

BIN				DEC	HEX
2^3	2^2	2^1	2^0		
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	B
1	1	0	0	12	C
1	1	0	1	13	D
1	1	1	0	14	E
1	1	1	1	15	F

I.3.2. Unsigned and signed integers:

So far we have talked about natural numbers. They can by nature only be positive or zero. Now consider relative integers or in other words, provided with a '+' or '-' sign.

In decimal,

+1, +2, +3 etc. are positive numbers. They are superior than 0 (n >0)

-1, -2, -3 etc. are negative numbers. They are less than 0 (n < 0)

In the same way:

+1, +10, +11, +100, +101 etc. are positive binary numbers,

-1, -10, -11, -100, -101 etc. are negative binary numbers.

The problem is that digital electronic circuits can only record 0 or 1 but not (+) or (-) signs. The only way is then to agree that if a number is likely to be negative, we reserve a bit for it to indicate the sign. It remains to determine which bit in a binary number would be best suited to symbolize the sign and which value of this bit (0 or 1) would be best suited to represent the "plus" sign or the "minus" sign.

Let's first observe the fact that machine-coded numbers have a fixed dimension:

- ✓ On paper, numbers have variable dimensions:
- Adding two 2-digit numbers gives a 2 or 3-digit number.
- Multiplying two 2-digit numbers gives 3 or 4-digit numbers.
- ✓ In machines, however, numbers are not extensible. They have fixed dimensions.

A. Integers:

When considering unsigned integers, the machine representation corresponds to the n-bit base 2 representation.

Example: Consider n equal to 8 bits:

The number 34 is written 100010. This will therefore give the 8-bit coding (00100010),

The number 252 is written 11111100 and will therefore be represented on 8 bits by (11111100).

But how to calculate the binary codes of negative numbers?

The calculation is done in two steps:

1) Calculation of the 1's complement = Replace all 0 with 1 and all 1 with 0.

2) Calculation of 2's complement = Add 1 to one's complement.

Example:

How to calculate (– 4) in binary?

We have $(+ 4)_{10} = (0000\ 0100)_2$

The one's complement of (0000 0100) 2 is (1111 1011)2

$$(1111\ 1011)_2$$
$$+\ 1$$
$$= (1111\ 1100)_2$$

Finely $(-4)_{10} = (1111\ 1100)_2$

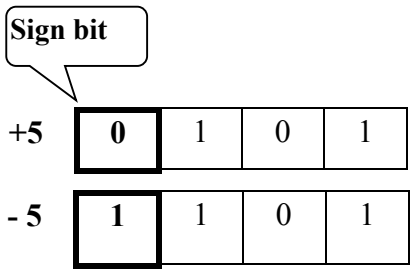
B. Signed binary numbers:

In this case, the most significant bit indicates the sign (0 if positive, 1 if negative) and the remaining bits representing the value of the number.

With n bits we can represent integers between: - (2n-1-1) and +(2n-1).

Example:

We want to represent on 4 bits, the signed number (+5) and the number (-5).



I.4. Principle of operation of a computer:

I.4.1. General Structure of a computer

In a computer, data introduced by an input device (keyboard, disks, mouse, etc.) is processed in the central unit CPU by a program which provides results to an output organ (screen, printer, disks, etc.). Generally speaking, a computer is made up of a CPU (Central Process Unit) and peripherals. The CPU is made up of a MicroProcessor (or Processor) and Central Memory. The latter is made up of a RAM (Random Access Memory) and a ROM (Read Only Memory). The processor consists of two units: control unit (CU) and arithmetic logic unit (ALU). The devices are composed of input devices, output devices and backup devices (external memories) as shown in Figure I.1.

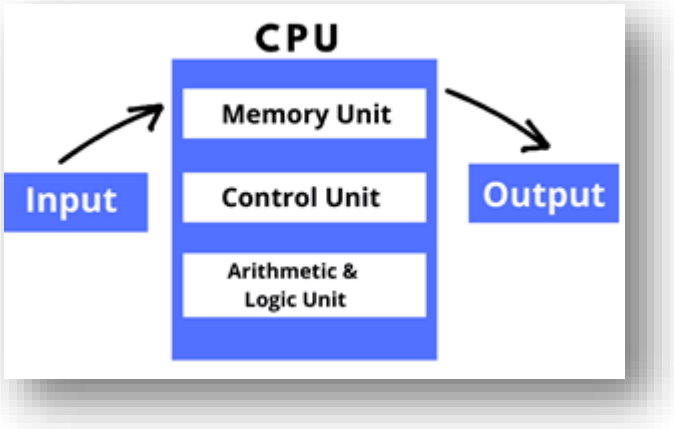


Figure I.1 : Structure of a computer.

A. Central Processing Unit C.P.U:

The role of the C.P.U is:

- Search for instructions in memory.
- Interpret or decode instructions.
- Check if the instructions are complete.
- Execute instructions.
- Put the results in reserve (in memory).

The CPU is essentially composed of:

B. Central Memory (MC):

It memorizes the program and the data during its execution, the Central Memory MC is made up of two memories: a Random Access Memory (RAM: Random Access Memory) and a Read Only Memory (ROM: Read Only Memory).

B.1. RAM memory:

Random Access Memory called random access memory or volatile memory (in the absence of current it will be erased), the RAM store the program to be executed as well as the results data. A program can only be executed if it is loaded into central memory. This memory is in the form of strips, it resides on the motherboard of the computer.

B.2. ROM memory:

Read-only memory: memory that is not erased by power failure. It is used to store code and system parameters necessary for the operation of the computer (BIOS: Basic Input/Output System Basic input/output program).

Main memory is currently measured in thousands of megabytes.

We have:

– 1 bit: (electronic state 1 or 0) = elementary unit of information (Binary Element).

– 1 Byte = 8 bits.

– 1 Kilobytes = 1024 bytes.

– One Megabyte (1MB) = 1024 KB.

– One Gigabyte (1GB) = 1024 MB.

– One Terabyte (1TB) = 1024 GB.

C. The Arithmetic and Logic Unit (ALU):

The UAL support arithmetic operations (additions, subtractions, multiplications and divisions) and logical operations: NOT (negation) - OR (disjunction) - AND (conjunction) - Comparisons - etc. The UAL includes registers, that is to say temporary memory spaces where it stores information and where it accumulates results.

D. The Control or Command Unit (UC):

The role of the CPU is to direct the computer's organs according to the program's instructions. It manages the operation of the UAL as well as the exchange of data and instructions with the memory. It ensures the decoding of data entered into the computer and stored in its internal memory and ensures that the operations commanded by the program instructions are executed in order and automatically.

E. The input/output units:

These are the organs ensuring communication between the machine and the external environment, for example:

- Input units: keyboard, mouse, sensor, etc.

- Output units: screens, printer, data show, plotter, etc.

F. Secondary memories (M.S):

Also called mass memories or auxiliary memories or peripheral memories, they are used to store information for a long period of time (HDD: Hard Disk Drive, FDD: Floppy Diskettes, CD-Rom, etc.). Secondary memories have larger capacities than those of central memory. For this, access to information in these memories is much longer compared to that of the M.C.

I.5. Computer System

A computer system is made up of two parts:

- Hardware part: Consisting of the central unit and the input/output devices.

- Software part: These are programs (sets of instructions that the machine must execute).

I.5. 1. Hardware part:

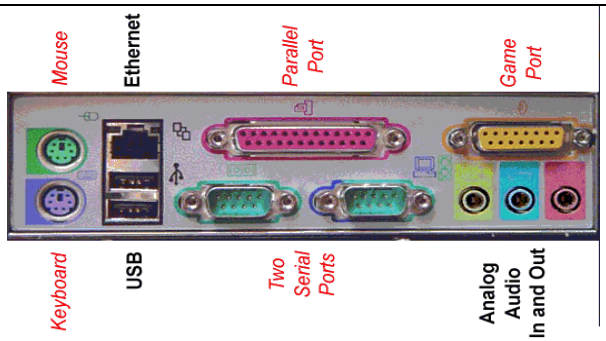
Components	Image
• **Motherboard :** The motherboard is a master card of the Personal Computer PC to which various essential components are connected, such as the processor, RAM, chipset, hard drive, graphics card, bios, sound card, graphics card, the expansion card, etc. etc. .	
• **Processor:** This is the brain of the computer. It executes program instructions using a set of instructions.	
• **Processor heat sink: (Cooler):** Its role is to dissipate heat from the processor to prevent its circuits from melting.	
• **RAM:** Allows information to be stored and exchanged during the entire operating time of the computer.	

- **ROM (Basic Input Output System):**

The basic program serving as an interface between the operating system and the motherboard. The BIOS is stored in ROM (read only memory).

- **Input/output Port:**

This port Allows connect different peripherals (screens, mouse, keyboard, microphone...etc).



- **Power Supply:**

Provides electricity to different parts of the computer.

- **HDD: Hard Disk Drive:**

It is the part of the computer used to store data permanently.

A hard drive is made up not of a single disk, but of several rigid disks made of metal, glass or ceramic, stacked at a very short distance from each other called platters.

• **Screen :** Computer video output device. It displays images generated by the computer's graphics card.	
• **Keyboard:** A keyboard is a man-machine interface equipped with keys allowing the user to enter a sequence of data, particularly textual data, into the computer.	
• **Mouse:** It allows you to point to elements displayed on the screen and select them by clicking.	

I.5. 2. Software part:

I.5. 2. 1. Software:

Is a set of programs allowing you to use the hardware part of the computer, there are two families of a software:

A. Operating systems OS:

They allow the management of hardware (the computer and its peripherals). They are generally provided by the manufacturers, among the basic software, we find the operating systems (Dos, Windows, Linux, etc.), the language compilers (Pascal, C, C++, etc.)

B. Application software:

Tool software (word processing, spreadsheet, etc.) or any user programs.

I.5. 2. 2. The functions of an operating system:

The major functions of an operating system are resource management (memory, CPU, shared files, etc.), data management, task management and providing a communication interface between

the user and the computer.

I.5. 2. 3. Programming languages

As with natural languages (English, Chinese, Arabic, Russian, French, etc.), there are a large number of programming languages. Some are tailored to particular domains, others depend on certain types of computers ...etc.

A. Definition:

A programming language is a conventional symbolism ensuring communication between the machine (computer) and the user. He has a well-defined vocabulary. There are three types of programming language: **I**nternal **M**achine **L**anguage ILM (The operations in this machine language are represented by hexadecimal or binary codes (Addition:= 0010; subtraction: = 0001; Comparison:=1110.), **E**xternal **M**achine **L**anguage ELM (The operations in this machine language are represented by significant mnemonics (Addition:= ADD; subtraction: = SUB; Comparison:=CMP) and advanced language (AL)

A.1. Assembler language:

It is a program that converts or translates a sequence of instructions (ELM program into an ILM program (directly executable by the computer)

A.2. Advanced Languages (AL):

They are also called high-level languages. Their degree of evolution is maximum (very close to natural languages. The symbols involved are more explicit than those of machine languages previously seen. (Example: Fortran, Pascal, Vhdl, C, C++,.....)

B. Translation systems:

B.1. Compiler:

The compiler is an important program responsible for translating a program written in AL (source program) into another expressed in machine language (object program). The computer, subsequently, would only have to execute the object program. The compiler translates the entire program before execution.

B.2. Interpreter:

The interpreter has a function almost analogous to that of the compiler except that it translates and executes the program instruction by instruction.

Algorithms

An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

We can also view an algorithm as a tool for solving a well-specified computational problem. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving that input/output relationship.

عبد الله بن محمد بن موسى **الخوارزمي**، أصله من **خوارزم** بأوزبكستان، لا يعرف الكثير عن حياته سوى أنه كان يعيش في بغداد في عهد الخليفة المأمون حوالي 813 إلى 833م وكان مشرفا على مكتبة المأمون وقد خلف العديد من الكتب منها: كتابا الزيج الأول والثاني، كتاب الرخامة، كتاب عمل الإسطرلاب، كتاب الجمع والتفريق، كتاب الجبر والمقابلة. أما أشهر كتبه وأهمها فهو كتاب: الجبر والمقابلة وقد كان الخوارزمي مهتما بعلم الفلك أيضا وشغل عضوية في اللجنة التي كلفت بقياس درجة من درجات محيط الأرض غير أن الخوارزمي حاز على شهرته الأساسية في مجال الرياضيات وفي الجبر بصفة خاصة.

الخوارزمي أول من اخترع مفهوم اللوغاريتمات، أو ما يعرف بالخوارزميات؛ وهو العلم الذي يعمل على حل المسائل المعقدة المختلفة وما زال يستخدم للآن،

II.1. Definition:

A. Program:

A series of instructions that can be executed in sequence, or in parallel (hardware parallelism) that performs a task and implements an algorithm.

B. Algorithm:

Word derived from the name of the mathematician Al_Khwarizmi who lived in the 9th century, was a member of an academy of sciences in Baghdad.

• An algorithm takes data as input, expresses a particular processing and provides data as output. An algorithm does not depend on the language in which it is implemented. An algorithm can be represented in lexical form or in flowchart form.

II.2. Fundamental problems in algorithms:

• **Modelable**: Are there tasks for which there is no algorithm? Given a task, can we say if there is an algorithm that solves it?

• **Complexity**: How long will an algorithm achieve the expected result? , How much space does it need?

• **Test:** Can we be sure that an algorithm answers the problem for which it was designed?

II.2. 1. Algorithmic language:

A. Textual Representation:

This form can be written as follows:

```
Algorithme Algorithme_Name
{This part is reserved for data declaration}
Constant MAX : integer - 10
Variables val, Number: integer
Name, familly_name : string
Begin
  { this part is reserved for data processing }
  val =     max + nombre
  Display (name,val)
End
```

B. Chart representation.

The different structures of an algorithm can be represented in the form of a flow chart.

Example 1: Suppose we have to make a flowchart for adding 2 numbers A and B.

Solution:

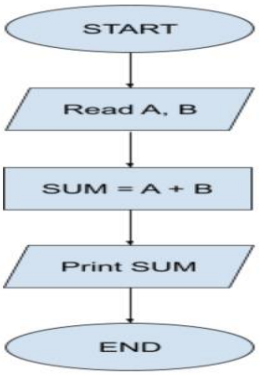


Figure II.1. Flowchart for adding 2 numbers A and B.

Example 2: Suppose we have to check if a number is even or odd.

Solution:

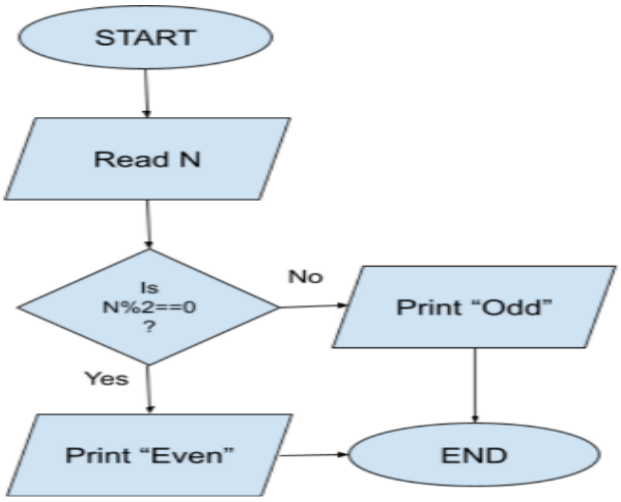


Figure II.2. Flowchart for check if a number is even or odd.

Example 3: To print the numbers less than or equal to n where n is given by the user.

Solution:

.

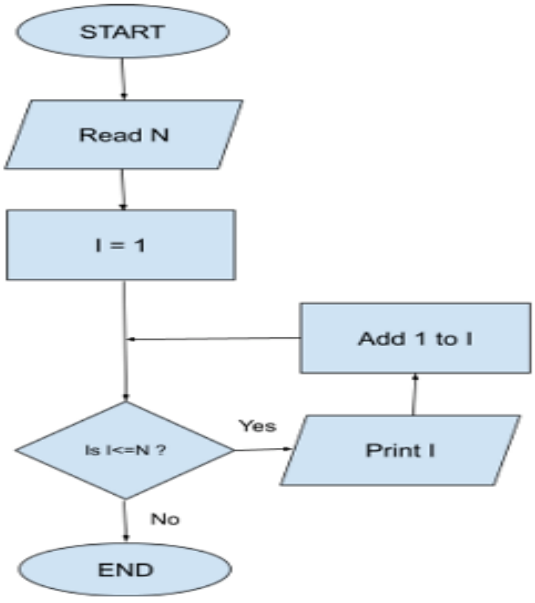


Figure II.3. Flowchart for check if a number is lesser than n.

II.3. Pascal program structure:

A Pascal program consists of three parts: the program header part, the declarations part, and the program body

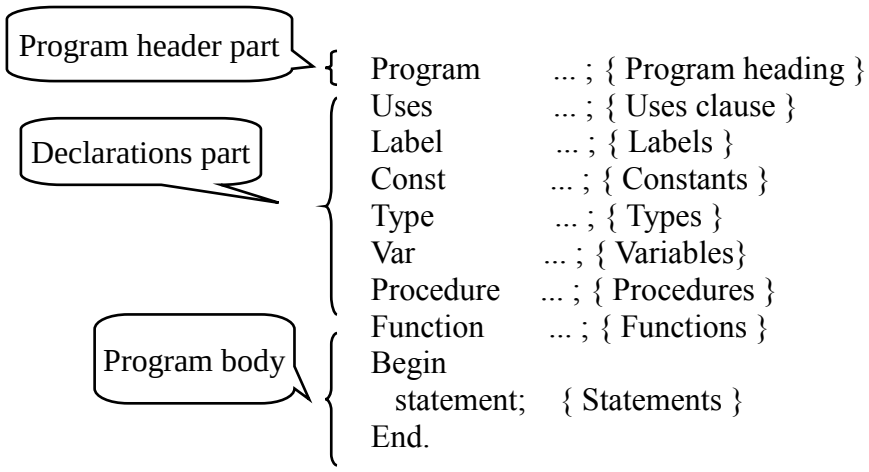


Figure II.4. General structure of a Pascal program

A. Header Part:

This part begins with the reserved word PROGRAM followed by an identifier of the program name, followed by the semicolon ';'.

Examples:

PROGRAM Operations;

Program calculates;

B. B. Declaration part:

This part includes:

B.1. The declaration of the used libraries, preceded by the reserved word 'USES'. The identifiers of the libraries are separated by commas and the list is terminated by a semicolon ';'.

Example: USES Crt, Graph, Dos;

B.2. The declaration of user-defined data type identifiers, preceded by the reserved word TYPE.

Example: TYPE Number = 0..99 ;

B.3. The declaration of constant identifiers, preceded by the reserved word CONST.

Example: CONST Pi = 3.14 ;

B.4. The declaration of variable identifiers, preceded by the reserved word VAR.

Example: VAR a : integer ;

B.5. Definition of the different subroutines (see the subroutines chapter).

C. Program body part (main block of the program):

This part contains the instructions to be executed, it begins with the reserved word BEGIN followed by a sequence of instructions (separated by semicolons ';') and ends with the reserved word END followed by a period '.'.

II.4 Problem analysis steps:

We generally distinguish four steps:

A. **Analyzing the problem statement** and making the objective of the program clear in our minds like what is the input and what is the required output.
B. **Break-down the complex problem into smaller parts** to make them easier to understand.
C. **Pseudo-code the solution** with basic steps that would help us get a clear intuition of what we are going to do with the problem statement.
D. **Run through test cases** to verify the solution.
E. **Code** the solution.

II.4.1 Pascal compiler:

- A program name respects the rules related to identifiers and **cannot contain the dot character "."**
- A main program always begins with **BEGIN** and ends with **END**. (With a dot).
- A subprogram (or function, procedure, conditional block...) also begins with **BEGIN but ends with END; (a semicolon).**
- Each command must end with a semicolon ';'. **There is no exception to the rule except BEGIN and the instruction preceding END or ELSE.**
- It is tolerated to put several instructions one after the other on the same line of the file but it is recommended to write only one per line.

II.5 Data structures:

II.5.1. Variables and constants:

A variable is an area in the computer's RAM, with a name and a type. The name of the variable allows access to the contents of the memory area; the type specifies the nature of what can be stored in the memory area (integer, real, character, etc.).

When declaring a variable, the processor reserves a memory location in RAM and allocates it a name (same name of the variable) with the specific size defined by the type of this variable;

Syntax:

Var <identifier> **: <type>;**

Example:

```
var   x       : integer ;
      temp    : real;
      Var1    : char
```

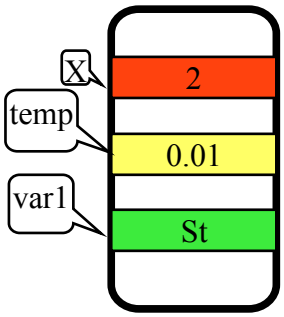


Figure II.4. RAM

II.5.1.2 Global variables and local variables

- Variables declared in the declaration part of the program are called global.
- Variables declared in the declaration part of a subprogram are called local.
- Global variables are known by all subprograms, while local variables are known only by the subprogram in which they are declared.

II.5.2 Constants

Unlike a variable, a constant cannot be modified (theoretically).

Syntax:

Const <identifier> = value;

Example:

```
CONST  faux   = false;
       integ  = 14;    { NAMED constants }
       reel   = 0.0;
       carac  = 'z';
       Chaine = 'hello';
```

II.5.3 Data Types

A data type defines the set of values that a variable can take. Each variable in a program must be associated with one and only one data type. Data types in TURBO Pascal are all constructed from simple (unstructured) types.

A simple type can either be defined by the programmer (it is then called a declared scalar type), or it can be one of the standard scalar types: integer, real, boolean, character, or byte.

The following is a description of these five standard scalar types.

A. The Integer Type

Integers are whole numbers. In TURBO Pascal, they are limited from -32768 to 32767. Integers occupy two bytes of memory.

Turbo Pascal also supports four other integer data types, each with a different range of values. Table 3.1 shows the five integer types.

Table 1: The five types of integers.

Type	Intervalle	Format
Shortint	-128..127	Signé 8-bit
Integer	-32768..32767	Signé 16-bit
Longint	- 2147483648..2147483647	Signé 32-bit
Byte	0..255	Non-signé 8-bit
Word	0..65535	Non-signé 16-bit

- Example: Integers declaration:

```
Var variable1 : integer;
    variable2 : longint;
    variable3 : word;
```

- The operations and functions that can be applied to integers are, for example:

1) + , -, *, DIV, MOD, INC, DEC, SUCC, PRED.

2) Relational operators: >=, <=, >, <, <>,… etc

B. The Real Type (REAL):

Standard Pascal defines the real data type, which can be represented in floating-point notation with a fixed number of digits. Consisting of a significand (fractional part) multiplied by an exponent (power of 10).

Turbo Pascal provides five predefined real types.

Table 2: The five predefined real types.

Type	Intervalle	Digits	Bytes
Real	2.9e-39.....1.7e38	11-12	6
Single	1.5e-45.....3.4e38	7-8	4
Double	5.0e-324.....1.7e308	15-16	8
Extended	3.4e-4932.....1.1e4932	19-20	10
Comp	-9.2e18.....9.2e18	19-20	8

Example:

5.00234E-5 = 0.0000500234.

5.00234E5 = 500234.

The operations and functions that can be applied to real numbers are, for example:

+ , -, *, / , ABS, SQR, SQRT, SIN, COS, ARCTAN, LN, EXT, ROUND, TRUNC, <=, >=, >, <.

Example:

Declaring real numbers in a program:

Var variable1 **: real;**

variable2 **: single;**

variable3 **: double;**

C. Character and String Data Types:

A Char value is a character in the American Standard Code for Information Interchange (ASCII) character set. Character constants are represented by surrounding the character with single quotes (e.g., 'A', 'e', '?', '2').

Note that '2' means the character 2, while 2 means the integer 2 (and 2.0 means the real number 2). They are categorized by their ASCII value, for example: 'A' < 'B'.

Ordinal (ASCII) values for characters range from 0 to 255. A character variable occupies one byte of memory.

Some functions that can be applied to characters include: ORD; PRED; SUCC; CHR;

Example:

Declaration of characters and character strings in a program:

Var variable1: char;

variable2: string;

D. Boolean data type

A Boolean value can take one of the logical truth values denoted by the standard identifiers True and False. These are defined so that False < True. A Boolean variable occupies one byte in memory.

The type of the variable is chosen according to the type of data it will receive. However, it is possible for the programmer to create his own types. Types must be declared with the Type keyword before declaring the variables.

Among the functions that can be applied to Booleans, we can cite:

- Logical operators AND, OR, NOT.
- Relational operators (<, >, >=, <=, <>).

E. Enumerated type

Syntax:

Type type_name = (identifier_1, identifier_2,.., identifier_n)

In this case, we must enumerate all the values of the enumerated type.

Example: Type month = (January, February, March, April);

F. Interval type

Syntax: type type_name = start_of_interval...end_of_interval

Example: Type average = 0...20;

Here we declare a type named average whose values are integers between 0 and 20.

II.6 Operators and expressions:

A. Assignment operator:

This instruction (operator) is used to transcribe a value into a variable.

The assignment symbol: (:=)

Syntax:

<Var> := <Val> ;

With:

- <Var>: variable of any type.

- <Val>: the value of a constant or variable data, the value of a function, or the result of an expression.

Example:

x :=2 ;

Y := X+1 ;

B. Relational and Logical Operators:

Operators fall into five categories, denoted by their order of precedence:

1) Unary minus (minus with only one operand).

2) Not.

3) Multiplication operators: *, /, div, mod, and, shl, and shr.

4) Addition operators: +, -, or, and xor.

5) Relational operators: <>, <, >, <, >.

- Sequences of operators of the same precedence are evaluated from left to right.

- Expressions in parentheses are evaluated first and independently of the preceding or following operators.

- If both operands of the multiplier and adder operators are of type Integer, the result is of type Integer. If one or both operands are of type Real, then the result is also of type Real.

- Expressions are composed of operators and operands. Most operators are binary, that is, they implement two operands (example: A+B). Operators with one operand are called unary (example: –A, +A).

- In more complex expressions, the existence of precedence rules makes it possible to eliminate any ambiguity in the order in which operations are performed.

- **The three basic rules of operator precedence are:**

1- An operand placed between two operators of different precedence will be bound to the one with the higher precedence.

2- An operand placed between two operators of the same precedence will be bound to the one on the left.

3- Expressions contained in parentheses are evaluated first in order to treat their results as a single operand.

In an arithmetic expression, the operators are the arithmetic operators (+, -, *, /, DIV, MOD).

An operand can be:

- A variable or numeric constant name.
- A numeric constant.
- A function name of a numeric type such as COS, SIN, etc.

A simple logical expression is a comparison between two arithmetic expressions.

The comparison operators are: =, <>, <, >, <=, >=.

A logical expression is the composition of simple logical expressions by the logical operators:

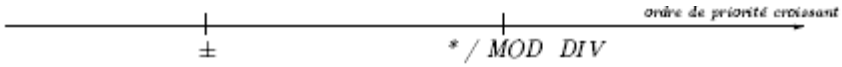
OR and AND binary logical operators, OR for disjunction and AND for conjunction. NOT unary operator, NOT negation operator.

C. String Operator:

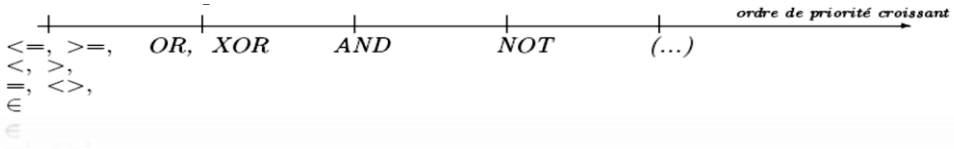
The only string operation is the + operator, which is used to concatenate two characters. Example 'A'+ 'B' = AB.

E. Priorities in Operators:

E. 1. Order of precedence of arithmetic operators:



F. 1. Order of precedence of Boolean operators:



Generally speaking, all these operators can be classified in descending order as sequences:

Table3: Operator precedence order:

Operators precedence order	**Operators**
1 primary	( ), function ( )
2 unary operators	+, -, not
3 multiplicative	*, /, div, mod, and
4 additive	+, -, or
5 relations	=, <>, <, <=, >=, >
6 Logical operators	Xor, Not, And.....

Example:

Calculate: 25/2 + 4*3 Div 2-5 MOD 4 Div 5

Progress of the evaluation:

step1: 25/2 + 4*3 Div 2-5 MOD 4 Div 5

step2: 12.5 + 4*3 Div 2-5 MOD 4 Div 5

step3: 12.5 + 12 Div 2-5 MOD 4 Div5

step4: 12.5 + 6-5 MOD 4 Div 5

step5: 12.5 + 6-1 Div5

step6: 12.5 + 6-0

step7: 18.5 - 0

step8: 18.5

II.7 Simple Instructions

An instruction specifies an operation or a sequence of operations to be performed on objects. Instructions are separated by semicolon';' and are executed sequentially, that is, one after the other, from the key word BEGIN to the key word END.

A. Input instructions READ ( ) and READLN ( ):

An input instruction allows you to read data from the keyboard.

The **READLN ()** command allows the user to enter a value that will be used by the program. This command causes a carriage return, i.e. the cursor moves to the next line.

When no variable is assigned to the command, simply press <ENTER>.

Syntax: Read (V1, V2,…., Vn)

ReadLn (V1,V2,...Vn)

B. Output instructions WRITE ( )

An output instruction allows the values corresponding to the arguments considered to be displayed on the screen. The WRITE () command allows text to be displayed and the cursor to be left at the end of the displayed text. This command allows character strings to be displayed, as well as values of variables, constants, types, etc.

- The text must be between apostrophes.
- If the text to be displayed contains an apostrophe, it must be doubled.
- The different variable names must be separated by commas.
- Adding LN at the end of Write () allows the cursor to return to the line.

Example:

We give: name is a variable that takes the value 5.

Execute the following instructions:

1)Write (nom);

2) Write('Nom') ;

3) Write ('Nom=', nom);

4) Write**LN** ('Nom= ', nom);

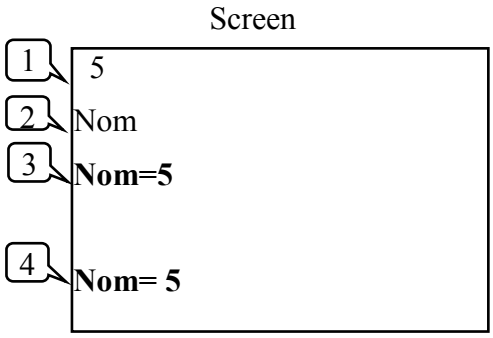


Figure.II.5: Execution of the WRITE () statement.

C. Identifiers:

The names of constants, variables, procedures, functions, tables, etc. (called identifiers) must be simple names, an identifier is used to give a name to an object.

Syntax

A letter is a character from 'a'..'z' or 'A'..'Z' or the character (_).

A digit is a character from '0'..'9'.

A Pascal identifier is a sequence of letters or digits joined together, starting with a letter.

Writing rules:

– All identifiers must start with a letter (a...z or A...Z). The rest of the identifier can be made up of letters, underscores and/or numbers (0...9).

– There is no difference between lowercase and uppercase letters.

– Accents or punctuation characters are not allowed.

– An identifier must be different from keywords (Begin, Write, Real, Case,...).

– An identifier must not contain accented characters, spaces, periods and the following characters: @, $, &, #, +, -, *, /.

– Numbers are accepted except in the first place like X123, G310.

– Identifiers can have any length, but only the first 63 characters are important.

II.8 Control structures:

II.8 .1 Conditional control structures (branching instructions):

In turbo pascal we find two test instructions which are the instruction: ***If...Then...Else*** ...and the instruction: ***Case (.....) of...***

A. The *If ... Then ... Else...* statement

This is the translation of "*If ... then ... otherwise ....*"

Syntax:

If (condition) **is true**, (Block1) will be executed

Otherwise (block2) will be.

In the case where (block2) does not exist;

The "If ...Then ... Else ..." statement is reduced to:

"If (condition) Then (statement 1);"

```
If (condition is True ) then
    begin
        instruction 1;
        instruction 2;          } Bloc1
        .....;
        instruction n;
    end
else
    begin
        instruction 3;
        instruction 4;          } Bloc2
        .....;
        instruction n;
end;
```

Rule:

- There is never a Semicolon (;) before the ELSE .
- The ELSE always refers to the last THEN Encountered

Example 4:

Write a program named absolute that calculates the absolute value of an integer.

```
Program absolue ;
Var  a  :integer ;
Begin
writeln ('entrez la valeur de A');
readln (A);
if  a > 0 then
a :=A;
else
A:= (-A);
writeln ('la valeur absolue est' , A);
Readln
end.
```

B. The *Case ... ..of …..END* instruction:

This instruction allows you to test and execute one of several instructions depending on the case of test_var. Its instructions are given below:

```
CASE (var_test) OF
        cas 1 : instruction 1 ; cas 1:instruction 1 ;
        cas 2 : instruction 2 ; cas 2:instruction 2 ;
        ... ...                 .....
        cas n : instruction n; cas n : instruction n
ELSE    instruction (n+1)
END;
```

With:

var_test : is a test variable (whose type is an integer, a character, a Boolean, or an enumerated, but not a real or a string).

case 1,...,case n: Are the possible values of test_var of the same type test_var.

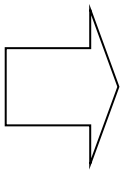
Example:

Write a calculator program that reads two integers and reads the type of operation to perform between the two integers.

```
program calculator ;
var ch :  char;
      a,b : integer:
begin
writeln (‘enter the type of operation performed ');
readln (ch);
writeln ('enter two positive values');
readln (a,b);
case ch of
          '+' : writeln ('sum': a+b)
else
          '-' : writeln (difference: a-b)
else
           '*' : writeln ('product': a*b);
end ;
readln
end .
```

The ***Case....of*** statement can be replaced by the ***if.....then*** statement as follows:

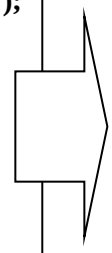
```
CASE (var_test) of
          cas 1 : instruction 1 ;
          cas 2 : instruction 2 ;
          ... ...
          cas n : instruction n;
else      instruction (n+1)
END;
```



```
if (var_test = cas 1) then
      instruction 1 ;
if (var_test = cas 2) then
instruction 2 ;
....
if (var_test = cas n) then
 instruction n ;
```

The calculator example can be written with the ***if ...then*** statement as shown in the figure:

```
program calculate ;
var ch :  char;
      a,b : integer:
begin
writeln ('enter the type of operation performed ');
readln (ch);
writeln ('enter two positive values');
readln (a,b);
case ch of
'+' : writeln ('sum': a+b);
'-' : writeln ('diff': a-b);
else
 '*' : writeln ('product': a*b);
end ;
readln
end .
```



```
program calculate ;
var ch :  char;
      a,b : integer:
begin
writeln ('enter the type of operation performed ');
readln (ch);
writeln ('enter two positive values');
readln (a,b);
if  ch = '+'  then
 writeln ('sum': a+b);
if  ch ='-'  then
 writeln ('diff': a-b);
if  ch = '*'  then
  writeln ('product': a*b);
readln
end .
```

II.8.2 Repetitive control structures:

These instructions allow the repetition of execution of one or more instructions, following a validation test.

In turbo pascal we distinguish three repetition instructions:

The ***FOR .....DO ....*** instruction, the ***WHILE .....DO....*** instruction and the ***REPEAT .....UNTIL*** instruction..

A. The While .....Do.... statement

This statement means (***while (condition is true) do .....***). This statement is controlled by a variable called the loop control variable, which allows us to exit the loop.

Its syntax is given by:

```
while  (condition Vrai ) do
begin
   instruction 1;
   instruction 2;
   .....................
   instruction n;
end.
```

Example:

Write a program that counts from 0 to 10.

```
program count ;
var i : integer ;
begin
i:=0
while  (i<=10 ) do
begin
          writeln ('i=',i);
          i:=i+1;
end ;
end.
```

B. The *Repeat... Until...* instruction:

This instruction means ***Repeat .... Until (condition is validated).*** It allows you to repeat the execution of the instructions between ***Repeat*** and ***Until***. This instruction is controlled by a variable called the loop control variable.

Syntaxe:

```
Repeat
          instruction 1 ;
          Instruction 2 ;
          .....................
          instruction n ;
Until (condition is true) ;
```

Example:

Write a program that counts from 0 to 10

```
program compte ;
var i : integer ;
begin
i:=0
repeat
        writeln ('i=',i);
        i:=i+1;
until  ( i <=10 ) do
end.
```

Note:

The difference between While and Repeat can be summarized by:

1- The 'While' instruction tests the truth of its condition before any execution,

2- The 'Repeat' instruction tests the truth of its condition only at the end of each repetition.

3- During the execution of 'while', its condition is True, while for 'Repeat' it is the opposite (the condition is False).

4- No need to surround a group of instructions with a Begin.....end, the Repeat... Until already plays this role.

5- The Repeat instruction is executed at least once (at least one iteration)

C. *For ... Do ...* statement:

The control variable in this statement is automatically incremented or decremented. When we know in advance the number of times to execute one or more statements, we call this statement.

Syntax:

```
For <integer_var> : = <min_value> to <max_value> Do
Begin
        instruction 1 ;
        instruction 1 ;
        .....
        instruction n ;
End;
```

With:

integer_var: Control variable of the loop of integer type.

min_value, **max_value**: Min and max values respectively of the start and end of the loop of the same type as integer_var.

Example:

Write a program that counts from 0 to 10.

```
program count ;
var  i : integer ;
begin
for i:=0 to 10 do
begin
   writeln ('i=', i);
   end;
end.
```

Note:

The FOR statement can be written as follows:

```
For <integer_var>  : = <max_value> Downto <min_value> Do
Begin
        instruction 1 ;
        instruction 1 ;
        .....
        instruction n ;
End;
```

D. Loop choice:

If the number of iterations is known a priori, then we use **for** statement.

Otherwise: we use the ***Repeat*** instruction (when there is always at least one iteration), or the ***while*** instruction (when the number of iterations can be zero).

Chapter III

Indexed Variables

Summary

III. Indexed Variables (Table):

An array is a structured type composed of a fixed number of components of the same type, called the component type. Or the base type. Each component can be explicitly accessed by indexes into the array. Indices are expressions of any scalar type enclosed in square brackets with the suffix of the array identifier. Their type is called the index type.

Generally speaking, the definition of an array requires three pieces of information:

- The type of the elements in the array (remember: an array is a sequence of data of the same type);
- The name of the array (in other words, its identifier);
- The length of the array (the number of elements that compose it). The latter must be an integer constant.

III.1 One-dimensional arrays (Vector):

The definition of an array includes the reserved word "ARRAY" followed by the index type, placed in square brackets, followed by the reserved word OF, followed by the component type such as: integer, real, Char, enumerated type or range type.

Syntax:

Type Vector: **Array** [range or enumerated type] **OF** type;

Example1:

TYPE interval = 1..12 ;
Vect : **Array**[interval] **of** real;

Example :
Representation of a one-dimensional array with 5 elements:

TYPE vect : **array** [0..5] **of** integer;
VAR v : vect;
V : Is an array of 6 integer values, indexed from 0 to 5.

Index	**0**	**1**	**2**	**3**	**4**	**5**
Memory box	**13**	**4**	**6**	**-7**	**13**	**0**

III.1.1 representation in Memory:

A table is represented in the computer's central memory (RAM) as a succession of memory cells (memory boxes). The physical address of a table element is obtained by the system. The vector V can be represented in memory as follows:

The vector V →

Physical Address	value
00000000	
00000001	
00000002	13
	4
	6
	-7
	13
	0
.....	
11111110	
11111111	

III.1.2 Accessing the elements of an array

Access to the elements of an array is done using an index, an integer corresponding to the position of each element in the array (first, second, third, etc.).

Example:

Considering the previous example:

V[0] = 13;

V[1] = 4;

V[2] = 6;

V[5] = 0;

III.1.3 Reading and writing a one-dimensional array

***) Reading a Table:**

```
Writeln ('Introduce V: ');
For i:= start_index to end_index do
Readln (V[i]);
```

*) Writing A Table (display):

```
Writeln ('Display of V: ');
For i:= start_index to end_index do
 writeln ( v[i]);
```

With:

start_index, end_index: the start and the end of the table index respectively with start_index < end_index

Example:

Write a program that reads and writes the elements of an array composed of 10 integer elements.

```
Program vector ;
Type  Vect : Array[1..10] of  Integer;
Var  i : integer ;
begin
Writeln ('Introduce V : ');
For  i:= 1  to  10  do
   Readln ( vect[i]);
Writeln ('display V : ') ;
For  i:= 1  to  10  do
   Writeln ( vect[i]);
readln
end.
```

III.1.4 Boundary control:

To make the program portable and easily modifiable, it is generally advisable to identify the interval bounds with named constants:

```
CONST  vec_min = 1;
           vec_max = 10;
TYPE vec_t : array [vec_min..vec_max] of integer;
```

Note

- It is totally forbidden to use an index outside the declaration interval, otherwise we will have an error at runtime (error when executing a program).
- It is therefore necessary to be very rigorous in the program, and not hesitate to test if an index i is correct before using v[i].
- At the declaration, the content of the table is undetermined, like any variable.
- We access the box index i by v[i] (and not v(i)).

Example:

Write a Pascal program that permutes between 2 values of an array chosen by the user.

```
program permut_tab;
Const min = 1;
max = 10;
var temp, i, box1, box2 :integer;
tab: Array[min..max] of integer;
begin
writeln ('***this program permutes between 2 boxes of a table***');
writeln (' ************** ************* ') ;
writeln (' enter the values of the table');
for i:= min to max do
begin
readln (tab[i]);
end;
writeln (' the table is ');
writeln (' enter the number of the first box to permute');
readln (box1);
writeln (' enter the number of the second box to permute');
readln (box2);
temp:=tab[box1];
tab[box1]:= tab[box2];
tab[box2]:= temp;
for i := min to max do begin writeln(tab[i]);
end;
readln end.
```

III.2 Two-dimensional arrays (Matrices):

The definition of a multidimensional array is done in the same way as that of a one-dimensional array except that you must provide the size of the different dimensions. The elements (components) in a two-dimensional array (or two-dimensional), are identified by a pair of indices (i, j) where i is the number of the row and j is the number of the column.

Syntax:

Declaration:

type Vector: **Array** [interval or enumerated type, interval or enumerated type] **OF** type;

v1: **Array** [1..10] **of array** [1..20] **of real**;

Example 1:

TYPE interval_x = 1..100 ;

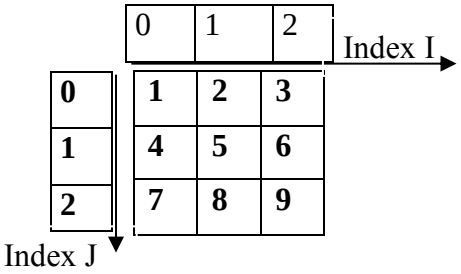
interval_y = 1..50 ;

Vect : **Array [**interval_X, interval_Y**] of char**;

III.2.1 Representation in Memory:

Technically, the data of a multidimensional array are stored next to each other in memory: they are gathered in a single-dimensional array

Representation of a Two-dimensional Vect array with (3X3) elements:



III.2.2 Accessing the elements of a two-dimensional array

Access to the elements of an array is done using the index of each dimension.

Example:

vect[2,2] =9;
vect[0,0]= 1;
vect [1,2]=8.

III.2.3 Reading and writing a two-dimensional array

Reading An Array:

```
Writeln('Introduce Vect:');
For i:= indexi_start  to indexi_stop do
For J:= indexj_start to indexj_stop do
   Readln ( vect[i,j]);
```

Writing a Table (display):

```
Writeln ('Display Vect : ');
For i:= indexi_start  to indexi_stop do
For J:= indexj_start to indexj_stop do
Writeln (vect[i,j]);
```

Avec :

indexi_start, indexi_stop: The start and end of the index "i" of the array respectively.

indexj_start, indexj_stop: The start and end of the index "j" of the array respectively.

Example:

Write a program that reads and writes the elements of a two-dimensional array composed of 9 integer elements.

```
Program  tab ;
Type  Vect : Array[1..3, 1..3] of  Integer;
Var  i, j : integer ;
begin
{lecture}
For  i:= 1  to  3  do
    For  J:= 1  to  3  do
         Readln ( vect[i]);
{ecriture}
For  i:= 1  to  3  do
    For  J:= 1  to  3  do
          Writeln ( vect[i]);
end.
```

Example:

Write a program that calculates the product of a matrix and a vector.

```
Program ProdMatVect;
var   mat :array [1..10, 1..10] of real;
      vect , p:array[1..10] of real;
      N,M, i,j : integer;
Begin
writeln (' the number of columns of the matrix must be
          equal to the number of components of the vector ');
Read(N, M);
For i:=1 To N Do
   For j:=1 To M Do
         Read(mat[i, j]);

For i:=1 To M Do
         Read(vect[i]);
For i:=1 To N Do
 begin
         p[i]=0;
         For j:=1 To M Do
         p[i]:= p[i]+ mat[i,j]*vect[j];
 end;
For i:=1 To N Do
         write(p[i]);
End.
```

Conclusion:

In this course we presented an introduction to computing, as well as the basic notions necessary to write programs, in Turbo Pascal language.

References :

- Jacques Courtin, Irène Kowarski, "Introduction to Algorithms and Data Structures," Volume 1, Dunod.
- Bruno Baynat, "Algorithm Exercises and Problems with Detailed Solutions," 2nd Edition, 2007.
- Claude Delannoy, "Programming in Turbo Pascal 7.0," Eyrolles.