

**République Algérienne Démocratique et Populaire**  
**Ministère de l'Enseignement Supérieure et de la Recherche Scientifique**  
**Université Abderrahmane Mira - Bejaia**  
**Faculté de la Technologie**  
**Département de Technologie**



**Polycopié de cours**  
Première Année Technologie

# Informatique 1

**Enseignant :**

Dr. OUARET Ahmed

**Année Universitaire : 2024/2025**

## **A propos de ce cours**

Ce support de cours concerne le module Informatique 1 destiné aux étudiants de la première année (L1) Technologie. Il est dédié à décrire de manière résumée les éléments de base de la science de l'informatique : notion de traitement automatique de l'information, évolution de l'informatique et des ordinateurs, architecture simplifiée d'ordinateur, composition d'un ordinateur, codage de l'information, les expressions arithmétiques, relationnelles et logiques, ...etc. La seconde partie de ce cours sert surtout à soigneusement introduire l'étudiant dans le domaine de la programmation par le biais de l'algorithmique : l'étape essentielle pour n'importe quel futur programmeur. Les points étudiés sont surtout : la notion d'algorithme, les étapes de conception algorithmique, la notion de donnée en entrée et résultat en sortie, constantes, variables et types de données scalaires, instructions de base, instructions conditionnelles et instructions itératives. En fin de semestre l'étudiant doit être capable d'écrire ses propres algorithmes dédiés à la résolution de problématiques de niveau moyen.

## **Pré-requis**

Pour bien suivre ce cours, l'étudiant doit avoir acquis certaines connaissances sur :

- Culture générale en informatique
- Compétences en résolution de problèmes
- Les langages de programmation

## **Les compétences visées**

Ce cours vise à :

- Comprendre les concepts fondamentaux de l'informatique
- Comprendre les concepts d'algorithmique
- Acquérir des compétences en programmation de base

## **Contenu de l'enseignement**

Cours : 1h30 par semaine

TP : 1h30 par semaine

Méthode d'évaluation : 60% examen et 40% contrôle continu

# Table des matières

## Chapitre I : Introduction à l'informatique

I.1. Objectif de ce chapitre .....	2
I.2. Définition de l'informatique .....	2
I.2.1. Role de l'informatique .....	2
I.3. Evolution de l'informatique et des ordinateurs .....	2
I.3.1. Evolution de l'informatique .....	2
I.3.2. Evolution des ordinateurs .....	3
I.3.2.1. Définition de l'ordinateur .....	3
I.3.2.2. Générations des ordinateurs .....	3
I.4. Architecture et fonctionnement d'un ordinateur .....	4
I.5. Composition d'un ordinateur (Matériel & Logiciel) .....	6
I.5.1. Partie matériel (Hardware) .....	6
I.5.2. Partie logiciel (Software) .....	9
I.6. Les systèmes de codage des informations .....	10
I.6.1. Les systèmes de numérotation .....	10
I.6.2. Conversion d'un nombre d'un système à un autre .....	12
I.6.3. Le code D.C.B. (Decimal coded Binary – Décimal codé binaire) .....	16
I.6.4. La codification Alphanumériques .....	16
I.7. Les expressions arithmétiques, relationnelles et logiques .....	17
I.7.1. Expressions arithmétiques .....	17
I.7.2. Expressions relationnelles .....	17
I.7.3. Expressions logiques (booléennes) .....	18
I.7.4. Les fonctions .....	18
I.7.5. La priorité dans les expressions .....	19
I.7.6. Evaluation des expressions .....	19
I.8. Exercices corrigés .....	21
I.9. Exercices supplémentaires .....	24

## Chapitre II : Notion d'algorithme et de programme

II.1. Objectif de ce chapitre .....	26
II.2. Concept d'un algorithme .....	26
II.3. La démarche et analyse d'un problème .....	27

II.4. Structures d'un algorithme .....	28
II.4.1. L'Entête .....	29
II.4.2. La partie déclarative .....	29
II.4.3. Le corps de programme .....	31
II.5. Types d'instructions .....	31
II.5.1. Instructions d'Entrées/Sorties (Lecteur/Ecriture) .....	31
II.5.1.1. Entrées (Lecteur) .....	31
II.5.1.2. Sorties (Ecriture) .....	32
II.5.2. Instruction d'affectation .....	33
II.5.3. Structures de contrôles .....	34
II.5.3.1. Structures de contrôle conditionnelle .....	34
II.5.3.2. Structure de contrôle répétitives .....	36
II.5.3.3. Structure de contrôle de branchements/sauts (l'instruction Goto) .....	38
II.6. Correspondance Algorithme-Pascal .....	40
II.7. Représentation en organigramme .....	42
II.7.1. Les symboles d'organigramme .....	42
II.8. Représentation des primitives algorithmiques .....	42
II.8.1. L'enchaînement .....	42
II.8.2. La structure alternative simple .....	43
II.8.3. La structure alternative double .....	44
II.8.4. La structure itérative POUR (Boucle POUR) .....	44
II.8.5. La structure itérative Tant-Que (Boucle Tant-Que) .....	45
II.8.6. La structure itérative Répéter (Boucle répéter) .....	45
II.9. Exercices corrigés .....	47
II.10. Exercices supplémentaires .....	56
<b>Bibliographie</b> .....	<b>60</b>

## Introduction

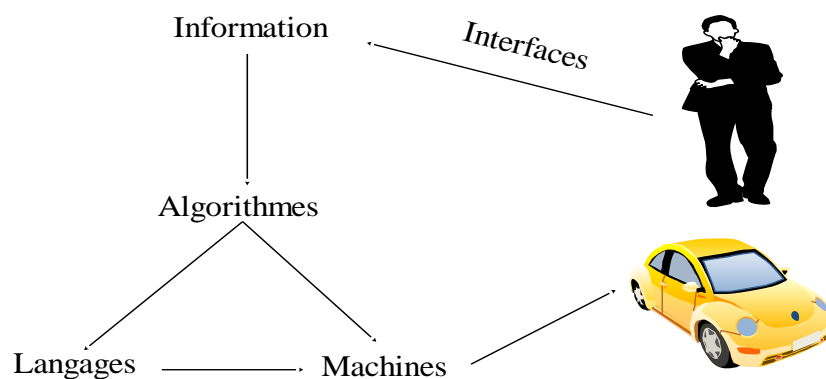
L'informatique, contraction d'information et automatique, est la science du traitement de l'information. Apparue au milieu du 20<sup>ème</sup> siècle, elle a connu une évolution extrêmement rapide. A sa motivation initiale qui était de faciliter et d'accélérer les calculs, de nombreuses fonctionnalités, comme l'automatisation, le contrôle et la commande des processus, la communication ou le partage de l'information. La mise en œuvre de ces systèmes s'appuie sur deux modes de réalisation distinct, le matériel et le logiciel. Le matériel (hardware) correspond à l'aspect concret du système : unité centrale, mémoire, organes d'entrées-sorties, etc.... Le logiciel (software) correspond à un ensemble d'instructions, appelé programme, qui sont contenues dans les différentes mémoires du système et qui définissent les actions effectuées par le matériel.

Ce cours intitulé « Informatique 1 » permet d'introduire les concepts généraux des ordinateurs et de la programmation.

Le cours est divisé en un ensemble d'unités d'apprentissage qui vous permet d'acquérir des compétences sur les différents composants d'un ordinateur, le système de codage, le concept d'un algorithme, l'approche et l'analyse d'un problème.

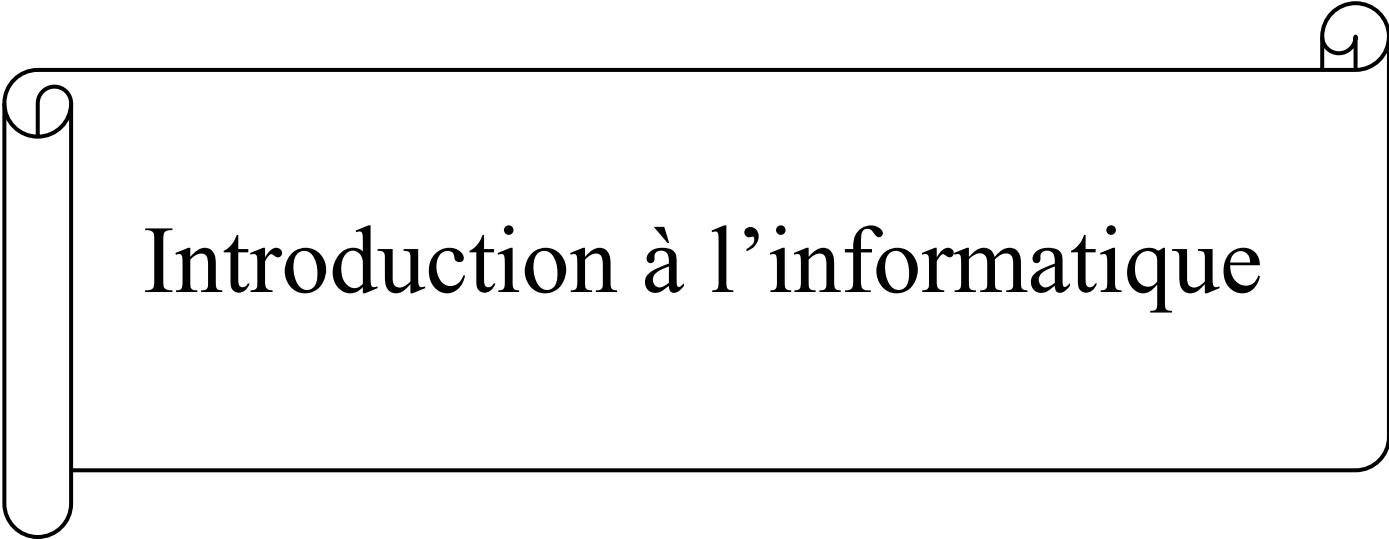
Il vous permet, également de découvrir le langage de programmation Pascal.

L'informatique peut être résumée par le schéma de la figure 1. Au centre se trouvent les algorithmes. En haut, les données, provenant d'hommes, de capteurs ou d'autres calculs informatiques. En bas, les machines qui effectuent les calculs, ainsi que les langages qui servent à transcrire la pensée humaine et à la traduire en instructions « machine ». À droite, les interfaces avec le monde extérieur des humains ou des autres systèmes informatiques : capteurs de données physiques, écrans de visualisation, effecteurs mécaniques, etc.



**Figure 1** : Les composantes de l'informatique

# CHAPITRE I



## Introduction à l'informatique

## Chapitre I : Introduction à l'informatique

### I.1. Objectif de ce chapitre

A l'issu de ce chapitre, l'apprenant sera capable de :

- Se familiariser avec l'environnement informatique
- Identifier les principaux composants d'un ordinateur
- Connaître les systèmes de codage des informations
- Apprendre comment évaluer les expressions arithmétiques et logiques

### I.2. Définition de l'Informatique

L'informatique est la **science** qui s'occupe du **traitement automatique** de **l'information**. Le terme "*informatique*" vient de la contraction des mots "*information*" et "*automatique*".

#### I.2.1. Rôle de l'informatique

L'informatique a pour rôle :

- La conception et la construction des ordinateurs;
- Le fonctionnement et la maintenance des ordinateurs;
- Leur exploitation (utilisation des ordinateurs dans les différents domaines d'activités).

### I.3. Evolution de l'informatique et des ordinateurs

#### I.3.1. Evolution de l'informatique

L'histoire de l'informatique est justement marquée par la volonté des hommes d'automatiser certaines tâches longtemps réalisées à la main, en particulier le calcul.

Le Tableau I.1 représente les différentes évolutions

**Tableau I.1** : Evolution de l'informatique

Date	Inventeur	Machine	Fonction
1642	Blaise Pascal	Pascaline	Effectue des additions et soustractions (des nombres de six chiffres)
1822-1832	Charles Babbage	Machine à Différences ( <i>Difference Engine</i> )	Effectue des calculs numériques compliqués (Tables numériques).
1887	Hermann Hollerith	Machine mécanographique	Accélérer le traitement des données lors du recensement

		(Calculateur de statistiques à cartes perforées)	américain de 1890 (invention de la carte perforée).
<b>1945 (inauguré en 1946)</b>	John Presper Eckert & John William Mauchly	“ENIAC” Electronic Numerical Integrator and Computer	Premier calculateur numérique électronique programmable. 18.000 tubes, 30 tonnes. Multiplication de deux nombre de 10 chiffres signés en moins de millisecondes.
	John Von Neumann	“EDVAC” Electronic Discrete Variable Automatic Computer	Enregistrer le programme mémoire.

### I.3.2. Evolution des ordinateurs

#### I.3.2.1. Définition de l'ordinateur

Un ordinateur est une **machine automatique programmable**, utilisée pour le **traitement de l'information**.

#### I.3.2.2. Générations des ordinateurs

- **1ère Génération (1945 - 1957)** : C'est les ordinateurs construits sur la base des “*tubes électroniques*” (aussi appelés “*tubes à vide*”. **Exemple** : ENIAC (1945) et IBM 701 (1952).
- **2ème Génération (1958 - 1963)** : Elle est définie par l'invention du “*Transistor*” en 1947 (le transistor est un composant électronique capable de réguler le courant). Ces ordinateurs sont 100 fois plus rapides que ceux de la 1ère génération et consomment moins d'énergie électrique et sont moins volumineux (occupent moins d'espace). **Exemples**: PDP I (1er ordinateur de 2ème génération (1960)), IBM 7030 (1961).
- **3ème Génération (1964 - 1971)** : Elle est définie après l'apparition du “*Circuit Intégré*” en 1958, les ordinateurs de cette génération utilisent les transistors et les circuits intégrés. **Exemple**: IBM 360 (1964), DEC PDP8 (1964).
- **4ème Génération (1971 - 1980)** : Elle survient après l'invention du “*microprocesseur*”. Le premier microprocesseur est fabriqué par la société INTEL en 1971. On parle, désormais, de microordinateurs. **Exemple**: MICRAL 8008 (1973), ALTAIR 8800 (1975).
- **5ème Génération (1980 - 2000)** : Les ordinateurs entrent dans les foyers, on parle alors d'informatique familiale (c'est la naissance des **ordinateurs personnels** (PC)). **Exemple** : IBM PC (1981), Intel Pentium (1993).

Aussi, les systèmes d'exploitation MS-DOS et Mac OS font leur apparition.



- **6ème Génération (2000 – aujourd'hui) :** On assiste, dans cette génération, à l'apparition des ordinateurs portables et des réseaux sans fil.

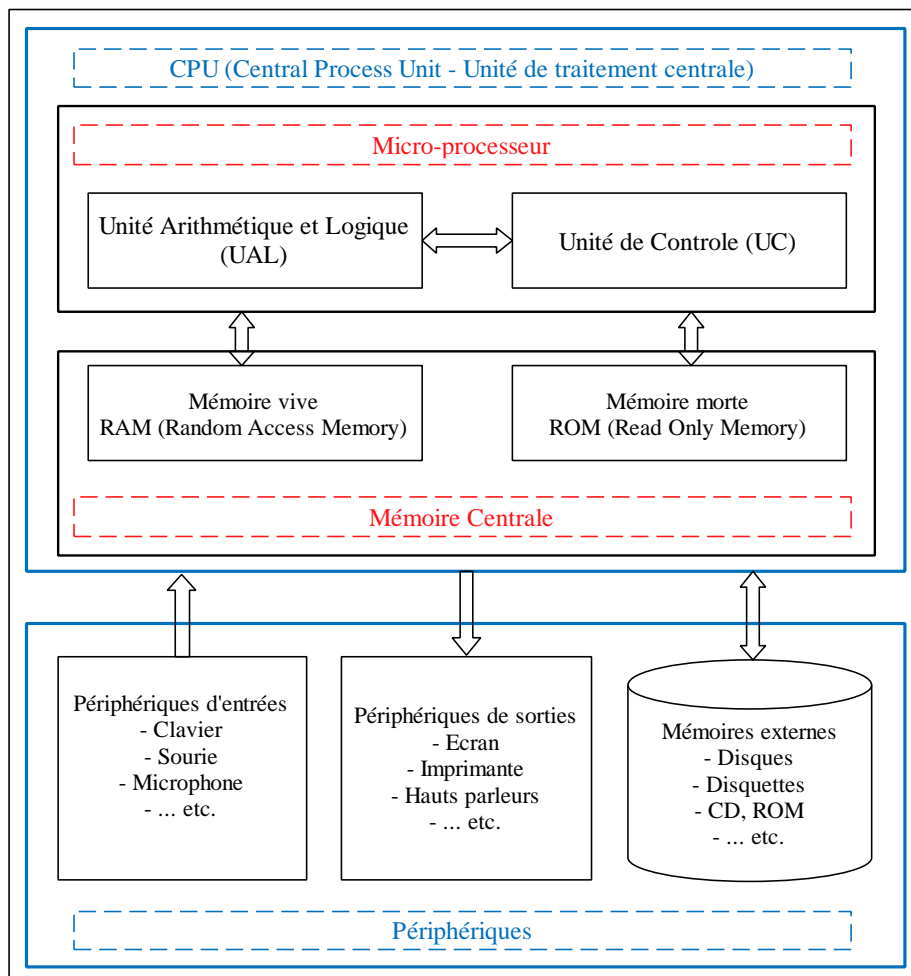
Actuellement l'évolution des ordinateurs tend à exploiter le laser, les fibres optiques, la biochimie et même l'informatique moléculaire.

#### I.4. Architecture et fonctionnement d'un ordinateur

L'architecture et le fonctionnement d'un ordinateur sont des aspects essentiels pour comprendre le système informatique.

L'architecture d'un ordinateur se réfère à la conception globale et à la structure interne de ses composants matériels, tels que le processeur, la mémoire, la mémoire centrale, les périphériques d'entrée/sortie, etc. Cette organisation matérielle permet à l'ordinateur d'exécuter des tâches spécifiques.

La Figure I.1 représente l'architecture générale d'un ordinateur



**Figure I.1 :** Architecture générale d'un ordinateur

**CPU** : Central Process Unit (Unité de traitement centrale)

**UAL** : Unité Arithmétique et Logique

**UC** : Unité de Contrôle ou de commande

**MC** : Mémoire Centrale

Un ordinateur est constitué, d'une manière générale, d'une Unité Centrale de Traitement, d'une mémoire centrale et d'un ensemble de périphériques.

**a) L'Unité Centrale de Traitement (UCT/CPU) ou bien Processeur (Microprocesseur) :**

Le processeur est le cœur de l'ordinateur, chargé de traiter et d'exécuter les programmes (les instructions) de l'ordinateur. Il comporte, essentiellement, deux unités : l'Unité de Contrôle (UC) et l'Unité Arithmétique et Logique (UAL).

✚ **L'unité de Contrôle (UC) :** Elle est responsable de la lecture en mémoire centrale et du décodage des instructions ;

✚ **L'unité Arithmétique et Logique (UAL) :** Aussi appelée unité de traitement, cette dernière exécute les instructions. Elle se charge d'effectuer toutes les opérations arithmétiques, logiques et relationnelles contenues dans l'instruction et effectue aussi des échanges de données (réception/transmission) avec la mémoire vive.

Ces deux unités communiquent, non seulement, avec la mémoire centrale, mais également avec les différents périphériques.

**b) La Mémoire Centrale (ou Principale) :** Elle est constituée d'une mémoire vive (RAM : Random Access Memory) et d'une mémoire morte (ROM : Read Only Memory).

✚ **La RAM :** Une RAM ou mémoire vive sert au stockage temporaire des données et du programme à exécuter. Les mémoires vives sont en général volatiles : elles perdent leurs informations en cas de coupure du courant électrique.

✚ **La ROM :** Une ROM ou mémoire morte, par opposition à la RAM, ne s'efface pas à la coupure du courant. Elle peut être lue mais pas (ou peu de fois) écrite. Elle conserve des programmes nécessaires au fonctionnement du matériel, surtout lors du démarrage (avant le chargement du système d'exploitation dans la RAM).

La mémoire centrale se mesure actuellement par milliers de mégaoctets.

On a :

□ bit : est l'abréviation de "binary digit" (chiffre binaire). C'est l'unité de base de l'information dans les systèmes informatiques et numériques. Un bit peut avoir deux valeurs possibles : 0 ou 1.

- ✓ 1 octet = 8 bits
- ✓ 1 Kilooctets (Ko) =  $2^{10}$  octets = 1024 octets
- ✓ 1 Mégaoctets (Mo) =  $2^{20}$  octets
- ✓ 1 Gigaoctets (Go) =  $2^{30}$  octets

c) **Les périphériques** : Les périphériques sont composés de périphériques d'entrée, de sortie et des périphériques d'entrée/sortie.

- + **Périphériques d'entrée** : C'est l'ensemble de périphériques qui permet de transmettre des données à l'ordinateur : Le clavier, la souris, le microphone, la webcam, etc.).
- + **Périphériques de sortie** : C'est l'ensemble de périphériques qui permet de recevoir des données de l'ordinateur et de les renvoyer vers l'extérieur : L'écran, l'imprimante, les haut-parleurs, etc.).
- + **Périphériques d'entrée/sortie** : C'est l'ensemble de périphériques qui permet, à la fois, de transmettre et de recevoir des données. Ils sont, également, appelés périphériques de stockage ou mémoires externes (mémoires auxiliaires) : La disquette, le CD-ROM, la clé USB, le disque dur externe, etc.).

## I.5. Composition d'un ordinateur (Matériel & Logiciel)

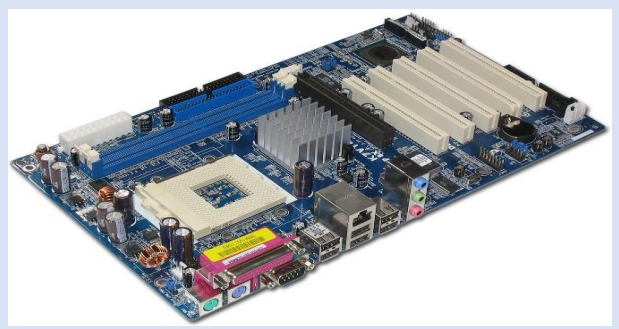
L'ordinateur est composé de deux parties essentielles : le matériel (hardware) et le logiciel (software).

### I.5.1. Partie matériel (Hardware)

Le matériel d'un ordinateur est la composante physique qui constitue la machine. Il inclut les composants électroniques, mécaniques et électriques qui permettent le fonctionnement de l'ordinateur. Voici quelques-unes des principales parties du matériel d'un ordinateur :

**- Carte Mère :**

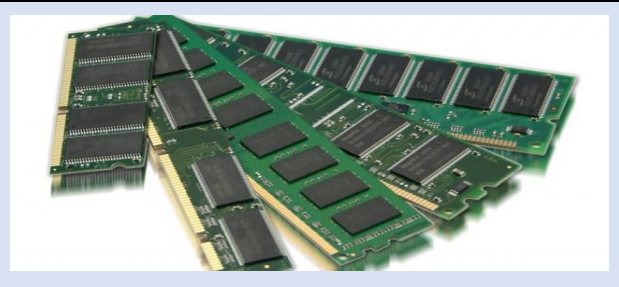
Carte principale sur laquelle on retrouve tous les composants nécessaires au fonctionnement d'un ordinateur, dont le processeur, la mémoire centrale, les bus et les connecteurs d'extension destinés à recevoir des cartes d'extension (carte graphique externe, carte son, carte réseau, etc.).

**- Processeur :**

Processeur dont tous les éléments sont miniaturisés et rassemblés sur une puce en un seul ou en plusieurs circuits intégrés. Le processeur remplit les fonctions d'unité centrale dans un micro-ordinateur. Un ventilateur est utilisé pour réduire sa température souvent élevée et réduire les risques de surchauffe.

**- Mémoire Vive :**

Mémoire du travail du processeur, tout programme exécuté doit être chargé sur la mémoire vive, dans laquelle les instructions peuvent lire ou écrire des données.

**- Disque Dur :**

Support de mémoire de masse d'accès rapide qui est composé d'un ou de plusieurs disques magnétiques généralement solidaires d'une mécanique, et ayant une très grande capacité de stockage.



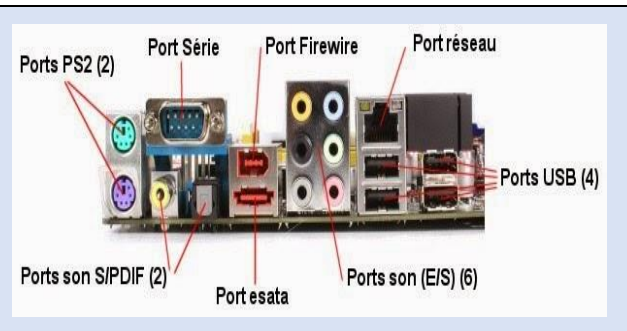
### - Carte graphique :

Carte d'extension servant à stocker et à interpréter les images reçues par l'ordinateur et à mettre à jour continuellement l'écran du moniteur. Elle n'est pas obligatoire sur certains ordinateurs qui en possèdent une incorporée dans la carte mère. Mais pour de meilleurs performances il est conseillé d'en installer une.



### - Connecteurs et Ports :

Ces ports permettent de relier des périphériques externes (clavier, souris, etc.) au boîtier de l'unité centrale. Il y a plusieurs type de ports : Port USB, port parallèle, port série, etc.



### - Écran :

Périphérique de sortie, il permet d'afficher les toute sorte de données. Il y a deux mode d'affichage : mode texte et mode graphique. Il y a des écran tactile permettant d'émettre des commandes via de simples touches de doigts, dans ce cas, l'écran représente aussi un périphérique d'entrée.



### - Clavier :

Périphérique permettant d'entrer des données sous forme textuelle vers le CPU. Donc il représente un périphérique de sortie.



- **Souris :**

Permet de lancer des commandes à travers de simples cliques : clique gauche pour la sélection, double clique gauche pour exécuter un programme ou ouvrir un fichier, clique droit affiche le menu contextuel, etc.

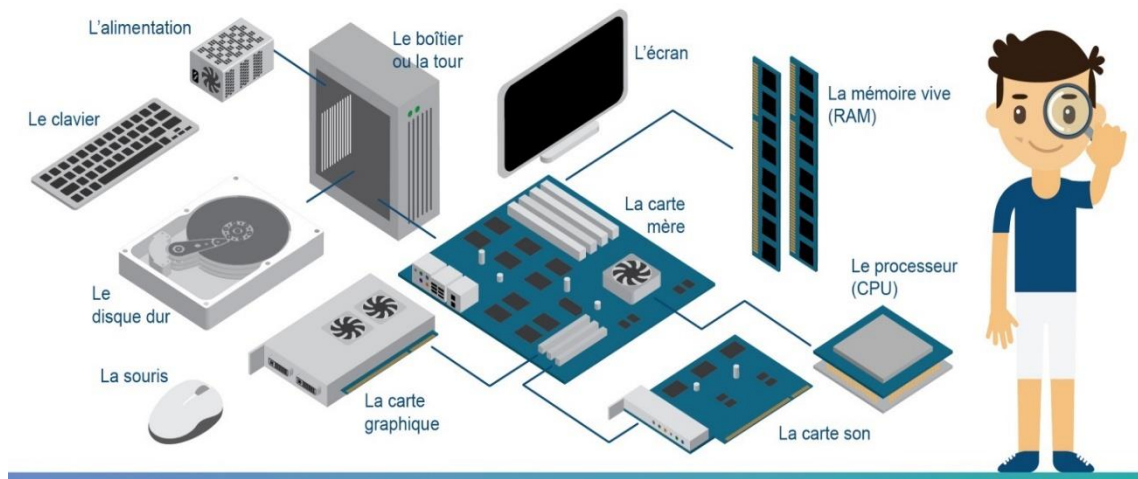


- **Imprimante :**

Périphérique de sortie permettant d'afficher des données (textuelles, images ou graphes) sur du papier.



La figure I.2 résume la partie hardware de l'ordinateur



**Figure I.2 :** Partie hardware de l'ordinateur

### I.5.2. Partie logiciel (Software)

Le logiciel d'un ordinateur est l'ensemble des programmes et des données qui permettent à l'ordinateur de fonctionner et d'effectuer des tâches spécifiques. On peut diviser cette partie en trois grandes catégories :

- **Système de base :** ce sont les systèmes d'exploitation comme Windows, Linux, Mac OS, etc.

- **Langages de Programmation** : Ce sont de langages utilisés par les développeurs d'applications et de logiciels afin d'écrire des programmes. On trouve plusieurs langage de programmation, comme C/C++, PASCAL, Java, Fortran, Matlab, etc.
- **Logiciels d'application** : Ce sont des programmes exécutables spécialement écrits pour un système d'exploitation, et qui permettent de réaliser tout type de fonctions: traitement de texte, jeux et loisirs, retouche d'image, navigation internet, lecture de médias son, image, vidéo... (comme suite de MSOffice, PhotoShop, Auto CAD, Jeux, etc.).

## I.6. Les systèmes de codage des informations

Toute information (numérique, textuelle, image, vidéo, son,.. etc.) manipulée par un ordinateur, est représentée par des séquences de deux chiffres : **0** et **1**, ces derniers sont désignés par **BIT** (**B**Inary digi**T**).

En informatique, il y a deux états possibles : il y a du courant électrique ou il n'y a pas de courant électrique, c'est la seule information qu'un ordinateur puisse comprendre. Cet état est déterminé par le **bit** (C'est la plus petite unité qui existe en informatique). Par conséquence :



### I.6.1. Les systèmes de numérotation

- **Définition** : C'est la représentation d'un entier naturel  $N$  en une base  $b$ . Ainsi, un système de numérotation se définit par deux éléments :
  1. La base du système ( $b$ );
  2. Les symboles du système (l'ensemble des chiffres et des lettres qui représentent la base).

En informatique, les systèmes les plus utilisés sont :

Système	Base	Symboles	Nombre de symboles du système
Décimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10
Binaire	2	0, 1	2
Octal	8	0, 1, 2, 3, 4, 5, 6, 7	8
Hexadécimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	16

avec :  $(A)_{16}=(10)_{10}$ ,  $(B)_{16}=(11)_{10}$ ,  $(C)_{16}=(12)_{10}$ ,  $(D)_{16}=(13)_{10}$ ,  $(E)_{16}=(14)_{10}$ ,  $(F)_{16}=(15)_{10}$ .

❖ **Remarque :** le nombre de symboles d'un système =  $b$ .

### • Notation

Un nombre quelconque  $N_b$  exprimé dans une base  $b$  sera noté comme suit :

$$N_b = (a_{n-1} a_{n-2} \dots a_1 a_0)_b$$

avec :

$b$  : la base du système de numérotation

$a_i$  : symbole du système de numérotation,  $i=0, \dots, n-1$  et  $a_i < b$

### Exemples :

$$N_{b_1} = (1995)_{10} \text{ avec } a_3=1, a_2 = 9, a_1 = 9, a_0 = 5$$

$$N_{b_2} = (243)_8 \text{ avec } a_2 = 2, a_1 = 4, a_0 = 3$$

$$N_{b_3} = (1011010)_2 \text{ avec } a_6 = 1, a_5 = 0, a_4 = 1, a_3=1, a_2 = 0, a_1 = 1, a_0 = 0$$

$$N_{b_4} = (BAC)_8 \text{ avec } a_2 = B, a_1 = A, a_0 = C$$

$N_{b_5} = (248)_8$  Cette notation est erronée, car le nombre contient **C** un symbole supérieur ou égale à la base

### ❖ Remarques :

- Quand la base n'est pas mentionnée, on considère qu'on est en base 10.
- Dans une base  $b$ , nous avons  $b$  chiffres : 0, 1, 2, ...,  $(b-1)$ .



## I.6.2. Conversion d'un nombre d'un système à un autre

### a) Conversion de la base b (base 2, 8, 16,...etc) → base 10

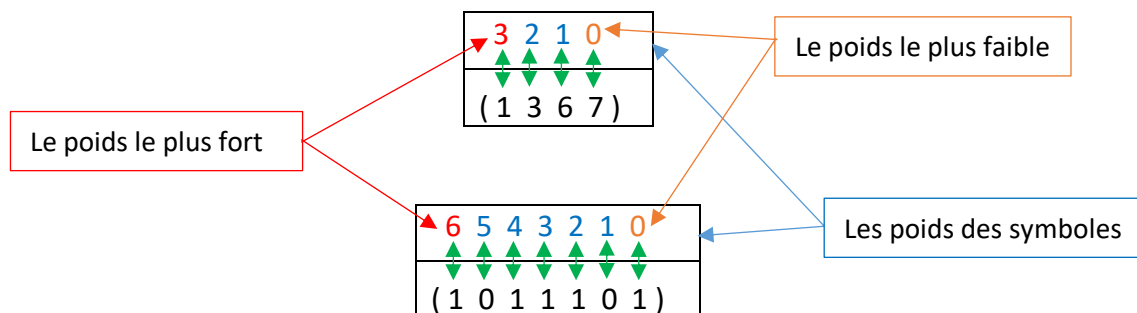
Pour convertir un nombre  $Nb = (a_{n-1} a_{n-2} \dots a_1 a_0)_b$  de la base b vers la base 10, on effectue le calcul suivant :

$$(Nb)_b = (a_{n-1} * b^{n-1} + a_{n-2} * b^{n-2} \dots + a_1 * b^1 + a_0 * b^0)_{10}$$

$$(Nb)_b = \sum_{i=0}^{n-1} a_i * b^i$$

Elle correspond à la somme des produits de chaque symbole du nombre par le poids correspondant.

**Le poids des symboles** : il s'obtient en numérotant les symboles à partir de la droite vers la gauche, en commençant du 0.



#### Exemple 1 :

$$(1367)_8 = (?)_{10}$$

$$(1367)_8 = (1 * 8^3 + 3 * 8^2 + 6 * 8^1 + 7 * 8^0)_{10}$$

$$(1367)_8 = (759)_{10}$$

#### Exemple 2 :

$$(1011101)_2 = (?)_{10}$$

$$(1011101)_2 = (1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0)_{10}$$

$$(1011101)_2 = (93)_{10}$$

#### Exemple 3 :

$$(2B3)_{16} = (?)_{10}$$

$$(2B3)_{16} = (2 * 16^2 + B * 16^1 + 3 * 16^0)_{10}$$

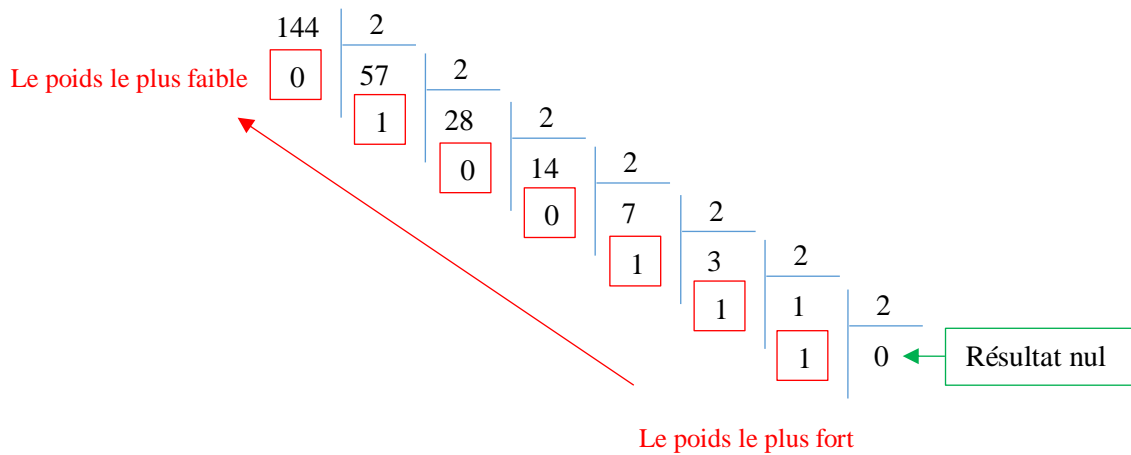
$$(2B3)_{16} = (691)_{10}$$

**b) Conversion de la base 10 → base b (base 2, 8, 16,...etc)**

Soit Nb un nombre exprimé dans la base 10, pour trouver son équivalent en base b, on applique la méthode des divisions successives sur b, jusqu'à l'obtention d'un résultat nul. Puis, on récupère les restes des divisions dans le sens inverse, i.e. le dernier reste trouvé représentera le poids le plus fort et le premier reste trouvé sera le poids le plus faible.

**Exemple 1 :**

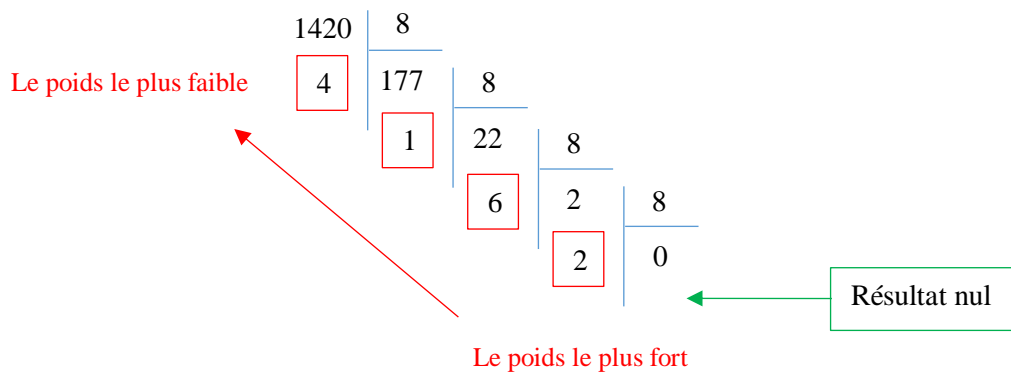
○  $(144)_{10} = (?)_2$



○  $(144)_{10} = (1110010)_2$

**Exemple 2 :**

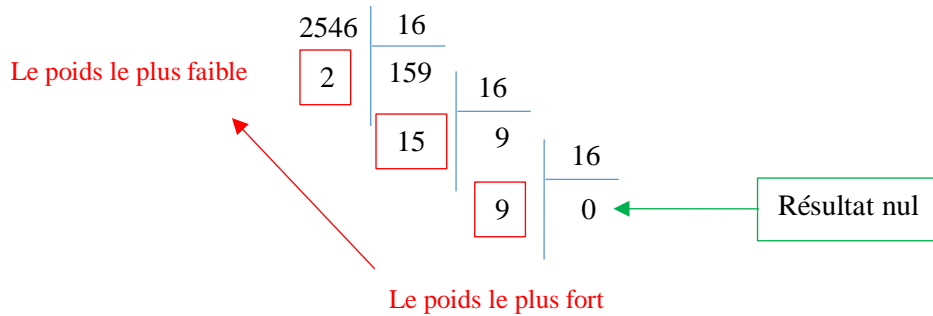
○  $(1420)_{10} = (?)_8$



○  $(1420)_{10} = (2614)_8$

**Exemple 3 :**

○  $(2546)_{10} = (?)_{16}$



○  $(2546)_{10} = (9F2)_{16}$

**c) Conversion de la base 8 → base 2**

Pour convertir un nombre Nb exprimé en base 8 vers la base 2, nous procédons comme suit:

$$8 = 2^3$$

Il faut donc utiliser **3 bits** pour exprimer un seul chiffre octal en binaire.

La représentation des chiffres de la base 8 vers le binaire est comme suit :

○  $(7)_8 = (1 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (111)_2$

○  $(4)_8 = (1 * 2^2 + 0 * 2^1 + 0 * 2^0)_2 = (100)_2$

○  $(3)_8 = (0 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (011)_2$

Chiffre en octal	Chiffre équivalent en binaire ( $2^2$ $2^1$ $2^0$ )
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

**Exemple :**

○  $(743)_8 = (111 100 011)_2$

**d) Conversion de la base 16 → base 2**

Pour convertir un nombre Nb exprimé en base 16 vers la base 2, nous procédons comme suit:

$$16 = 2^4$$

Il faut donc utiliser **4 bits** pour exprimer un seul chiffre hexadécimal en binaire.

La représentation des chiffres de la base 16 vers le binaire est comme suit :

Chiffre en hexadécimal	Chiffre équivalent en binaire ( $2^3$ $2^2$ $2^1$ $2^0$ )
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1

Chiffre en hexadécimal	Chiffre équivalent en binaire ( $2^3$ $2^2$ $2^1$ $2^0$ )
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

- $(7)_{16} = (0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (0111)_2$
- $(B)_{16} = (1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (1011)_2$
- $(3)_{16} = (0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0)_2 = (0011)_2$
- $(A)_{16} = (1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0)_2 = (1010)_2$

**Exemple :**

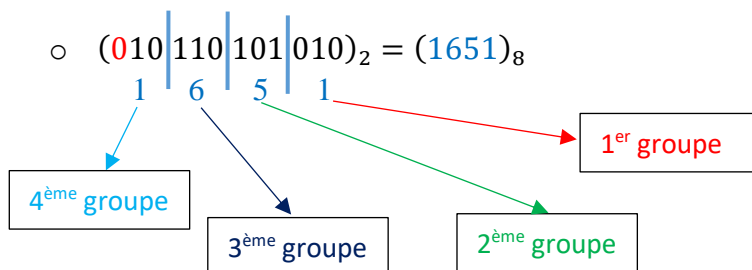
- $(7B3A)_{16} = (0111\ 1011\ 0011\ 1010)_2$

**e) Conversion de la base 2 → base 8**

Pour trouver l'équivalent d'un nombre binaire en octal, il suffit de former des **groupes de 3 bits** chacun ( Puisque  $8 = 2^3$ ), en commençant du poids le plus faible ( à partir de la droite), si le dernier groupe formé possède moins de 3 bits, il suffit de rajouter des 0, puis calculer l'équivalent en octal de chaque groupe.

**Exemple :**

- $(10110101010)_2 = (?)_8$



**f) Conversion de la base 2 → base 16**

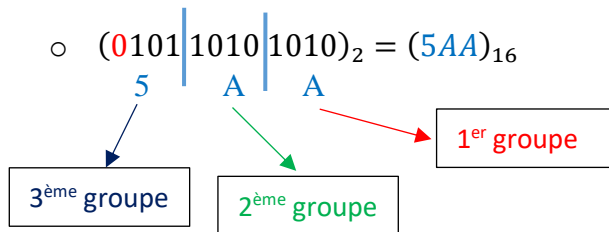
Pour trouver l'équivalent d'un nombre binaire en Hexadécimal, il suffit de former des **groupes de 4 bits** chacun ( Puisque  $16 = 2^4$ ), en commençant du poids le plus faible ( à partir

de la droite), si le dernier groupe formé possède moins de 4 bits, il suffit de rajouter des 0, puis calculer l'équivalent en Hexadécimal de chaque groupe.

**Exemple :**

○  $(10110101010)_2 = (?)_{16}$

○  $(0101|1010|1010)_2 = (5AA)_{16}$



**g) Conversion de la base 16 → base 8**

Pour convertir un nombre Nb exprimé en base 16 vers la base 8 ou vice versa, nous devons **passer par une base intermédiaire** tel que le **décimal ou le binaire**, mais le passage par le binaire est beaucoup plus simple.

**Exemple :**

○  $(C9F)_{16} = (1100\ 1001\ 1111)_2$

○  $(110|010|011|111)_2 = (6237)_8$

**I.6.3. Le code D.C.B. (Decimal Coded Binary – Décimal codé binaire)**

Le code DCB consiste à convertir chaque chiffre décimal en un nombre binaire sur 4 positions.

**Exemple :**

Le nombre décimal 378 est codé en DCB comme suit : 0101 0111 1000

Dans ce code, chaque chiffre décimal est remplacé par 4 chiffre binaires (comme dans le cas de hexadécimal).

**I.6.4. La codification Alphanumériques**

Les chiffres, lettres, signes de ponctuation, les symboles mathématiques, etc., sont représentées généralement en utilisant le code normalisé à 8 positions binaires. On utilise pour cela le code EBDIC (Extended Binary Coded Decimal International Code) ou le code ASCII (American Standard Code Information Interchange).

**Exemples :****Tableau I.2 :** Exemples de codification Alphanumériques

Lettre / Signe / Symbole	En EDCDIC	En ASCII
A (Majuscule)	11000001	10100001
D (miniscule)	10000010	11100100
=	01111110	01011101

**I.7. Les expressions arithmétiques, relationnelles et logiques****I.7.1. Expressions arithmétiques**

C'est une combinaison d'opérande(s) (valeur, constante, variable) et d'opération(s) arithmétiques. Le Tableau I.3 résume ces opérations.

**Tableau I.3 :** Les opérations arithmétiques

Opération	Symbole
Addition	+
Soustraction	-
Multiplication	*
Division réelle	/
Division entière	Div
Modulo (reste de la division entière)	Mod

Applicables sur des entiers

❖ **Remarque :** La division entière donne comme résultat un quotient entier tandis que la division réelle donne un quotient réel. Le Mod, quant à lui, est le modulo, c.à.d. le reste de la division entière.

10 Div 4 = 2 tandis que  $10 / 4 = 2.5 \rightarrow 10 \text{ Mod } 4 = 2$

20 Div 6 = 3 tandis que  $20 / 6 = 3.33 \rightarrow 20 \text{ Mod } 6 = 2$

6 Div 8 = 0 tandis que  $6 / 8 = 0.75 \rightarrow 6 \text{ Mod } 8 = 6$

**I.7.2. Expressions relationnelles**

C'est les expressions qui utilisent des opérateurs relationnels (ou de comparaison). Le Tableau I.4 résume ces opérateurs.

**Tableau I.4 :** Les opérateurs relationnels

Relation	Symbole
Egalité	=
Inférieur	<
Inférieur ou égal	<=

Supérieur	>
Supérieur ou égal	>=
Différent	<>

### I.7.3. Expressions logiques (booléennes)

C'est une combinaison de variables de type booléen (**True** : Vrai ou **False** : Faux) et d'opérateurs booléens (**Not** : Non, **And** : Et, **Or** : Ou). Le Tableau I.5 représente la table de vérité.

**Tableau I.5** : L'opérateur booléens AND

Opérande 1	Opérande 2	Opérande 1 <b>AND</b> Opérande 2
True	True	True
True	False	False
False	True	False
False	False	False

**Tableau I.6** : L'opérateur booléens OR

Opérande 1	Opérande 2	Opérande 1 <b>OR</b> Opérande 2
True	True	True
True	False	True
False	True	True
False	False	False

**Tableau I.7** : L'opérateur booléens NOT

Opérande	<b>NOT</b> Opérande
True	False
False	True

### I.7.4. Les fonctions

Le Tableau I.8 présente une liste non-exhaustive de fonctions standards (ou prédéfinies) applicables sur des entiers ou des réels.

**Tableau I.8** : Les fonctions

Fonction	Appel	Résultat retourné
Abs	Abs(x)	La valeur absolue d'un nombre x
Exp	Exp(x)	L'exponentiel d'un nombre x
Ln	Ln(x)	Le logarithme népérien d'un nombre x
Log10	Log10(x)	Le logarithme à base 10 d'un nombre x
Sqrt	Sqrt(x)	La racine carrée d'un nombre x
Sqr	Sqr(x)	Le carré d'un nombre x

Arctan	Arctan(x)	L'arc tangente d'un nombre x
Cos	Cos(x)	Le cosinus d'un nombre x
Sin	Sin(x)	Le sinus d'un nombre x
Round	Round(x)	La valeur arrondie d'un nombre x
Trunc	Trunc(x)	La partie entière d'un nombre x
Etc.	..	..

### I.7.5. La priorité dans les expressions

La priorité des opérateurs dans les expressions arithmétiques, logiques et relationnelles est comme suit :

1. Les parenthèses ;
2. Les fonctions ;
3. Le moins unaire, le Not ;
4. \*, /, Div, Mod, And
5. +, -, Or
6. =, <>, <, >, <=, >=

### I.7.6. Évaluation des expressions

L'évaluation d'une expression consiste à calculer, au fur et à mesure, les résultats des calculs jusqu'à obtenir un résultat finale. Cela se fait en plusieurs étapes :

- Écrire l'expression sous forme linéaire (Il faut noter qu'en algorithmique, les expressions s'écrivent sous forme linéaire:  $\frac{(x+z)}{(y*2)} \rightarrow (x + z)/(y * 2)$  ;
- Remplacer les identifiants (c'est à dire les noms) des variables et des constantes par leurs valeurs ;
- Évaluer (Calculer) étape par étape chacune des sous-expressions en commençant par les sous-expressions qui sont dans les parenthèses les plus internes.
- Indiquer à chaque calcul, le rang d'évaluation.

❖ **Remarque :** Si les opérateurs ont le même rang de priorité, l'évaluation se fait de gauche à droite.

#### Exemple :

Évaluer l'expression suivante en indiquant l'ordre d'évaluation :

$$E = (\text{sqr}(b) \text{ mod } a > c) \text{ or } (d / (a + 3) <> b) , \text{ avec } a = 2, b = 3, c = 1, d = 10$$



**1ère méthode :**

$$E = (\text{sqr}(3) \bmod 2 > 1) \text{or} (\underline{10 / (2 + 3)} <> 3)$$

(1)

$$E = (\underline{\text{sqr}(3) \bmod 2 > 1}) \text{or} (10 / \underline{5} <> 3)$$

(2)

$$E = (\underline{9 \bmod 2 > 1}) \text{or} (10 / 5 <> 3)$$

(3)

$$E = (\underline{1 > 1}) \text{or} (10 / 5 <> 3)$$

(4)

$$E = \text{False} \text{or} (\underline{10 / 5} <> 3)$$

(5)

$$E = \text{False} \text{or} (\underline{2} <> 3)$$

(6)

$$E = \text{False} \text{or} \underline{\text{True}} = \text{True}$$

(7)

**2ème méthode :**  $E = (\text{sqr}(3) \bmod 2 > 1) \text{or} (10 / (2 + 3) <> 3) = \text{True}$

(2) (3) (4) (7) (5) (1) (6)

## I.8. Exercices corrigés

### Exercice N°01 : (Systèmes de numérotation)

Effectuer les conversions suivantes :

$$2022 = (?)_2 \quad ; \quad (1011001101)_2 = (?)_{10} \quad ; \quad (1011001101)_2 = (?)_8 = (?)_{16} \quad ;$$

$$(32103)_4 = (?)_2 \quad ; \quad (37163)_8 = (?)_2 \quad ; \quad (379)_{10} = (?)_{16} \quad ; \quad (3A2D)_{16} = (?)_{10} \quad ;$$

$$(4D5B)_{16} = (?)_8$$

### Corrigé de l'exercice N°01 : (Systèmes de numérotation)

Effectuer les conversions suivantes :

$$2022 = (11111100110)_2 \quad ;$$

$$(1011001101)_2 = (717)_{10} \quad ;$$

$$(1011001101)_2 = (1315)_8 = (2CD)_{16} \quad ;$$

$$(32103)_4 = (1110010011)_2 \quad ;$$

$$(37163)_8 = (011111001110011)_2 \text{ ou bien } (11111001110011)_2 \quad ;$$

$$(379)_{10} = (17B)_{16} \quad ;$$

$$(3A2D)_{16} = (14893)_{10} \quad ;$$

$$(4D5B)_{16} = (46533)_8$$

### Exercice N°02 : (Expressions arithmétiques en Algèbre/Pascal)

Réécrire les expressions mathématiques en Algèbre/Pascal

Expressions mathématiques	PASCAL
$b^2 - 4ac$	
$\frac{-b - \sqrt{d}}{2a}$	
$e^{3a} +  b $	
$4a < \frac{b}{c}$ ET $(5c \leq 7)$ OU $(a \neq b)$	

Utiliser le tableau suivant :

Expression	PASCAL
$2a$	$2 * a$
$\frac{a}{b}$	$a/b$
$a^2$	$sqr(a)$
$\sqrt{a}$	$sqr(a)$
$ a $	$abs(a)$
$\ln(a)$	$\ln(a)$
$\log(a)$	$\ln(a) / \ln(10)$
$e^a$	$\exp(a)$
$x^n$	$\exp(n * \ln(x))$

**Corrigé de l'exercice N°02 :** (*Expressions arithmétiques en Algorithme/Pascal*)

Réécrire les expressions mathématiques en Algorithme/Pascal

Expressions mathématiques	PASCAL
$b^2 - 4ac$	$sqr(b) - 4 * a * c$ ou bien $b * b - 4 * a * c$
$\frac{-b - \sqrt{d}}{2a}$	$(-b - sqrt(d))/(2 * a)$
$e^{3a} +  b $	$exp(3 * a) + abs(b)$
$4a < \frac{b}{c}$ ET $(5c \leq 7)$ OU $(a \neq b)$	$4 * a < (b/c)$ AND $(5 * c \leq 7)$ OR $(a \neq b)$

**Exercice N°03 :** (*Evaluation des expressions*)

Evaluer les expressions suivantes en respectant l'ordre de priorité des opérateurs :

Expression 1 :  $50 + 3 \text{ MOD } 2 - 4 \text{ DIV } 3 + 40$

Expression 2 :  $a/b + ((d * c + 3)/5 * a) + 2 * c$  ; avec  $a = 4$ ;  $b = 2$ ;  $c = 4$ ;  $d = 3$

Expression 3 :  $(a < b) \text{ OR NOT } (c \leq d) \text{ AND } (b > a)$  ; avec  $a = 1$ ;  $b = 2$ ;  $c = 4$ ;  $d = 6$

**Corrigé de l'exercice N°03 :** (*Evaluation des expressions*)

**Expression 1 :**  $50 + 3 \text{ MOD } 2 - 4 \text{ DIV } 3 + 40$   
(1)

$$50 + 1 - 4 \text{ DIV } 3 + 40$$

(2)

$$50 + 1 - 1 + 40$$

(3)

$$51 - 1 + 40$$

(4)

$$50 + 40 = 90$$

(5)

**N.B :** Lorsque les opérateurs ont la même priorité, on commence par le plus à gauche

**Expression 2 :**  $a/b + ((d * c + 3)/5 * a) + 2 * c$

$$4/2 + \underbrace{((3 * 4 + 3)/5 * 4)}_{(1)} + 2 * 4$$

$$4/2 + \underbrace{((12 + 3)/5 * 4)}_{(2)} + 2 * 4$$

$$4/2 + \underbrace{(15/5 * 4)}_{(3)} + 2 * 4$$

$$4/2 + \underbrace{(3 * 4)}_{(4)} + 2 * 4$$

$$\underbrace{4/2 + 12}_{(5)} + 2 * 4$$

$$2 + 12 + \underbrace{2 * 4}_{(6)}$$

$$2 + 12 + 8$$

$$\underbrace{14 + 8}_{(8)} = 22$$

On commence par remplacer les variables par leurs valeurs

**Expression 3 :**  $(a < b) \text{ OR NOT } (c \leq d) \text{ AND } (b > a)$

$$\underbrace{(1 < 2)}_{(1)} \text{ OR NOT } (4 \leq 6) \text{ AND } (2 > 1)$$

$$\text{True OR NOT } \underbrace{(4 \leq 6)}_{(2)} \text{ AND } (2 > 1)$$

$$\text{True OR NOT True AND } \underbrace{(2 > 1)}_{(3)}$$

$$\text{True OR NOT True AND True}$$

$$\text{True OR } \underbrace{\text{False AND True}}_{(5)}$$

$$\underbrace{\text{True OR False}}_{(6)} = \text{True}$$

## I.9. Exercices supplémentaires

### Exercice 01 :

1. Donner l'ordre de priorité des opérateurs arithmétiques et logiques dans les expressions suivantes :

$$E1 = (2 * x + 3 * y) / x + 6 * y - (5 \text{ MOD } y * 2 * x)$$

$$E2 = (3 \text{ MOD } a * 2 <= b) \text{ OR } \text{NOT}(b >= 8) \text{ NOT}(a = b)$$

2. Traduire les expressions suivantes en langage Pascal (on rappelle que  $a^b = e^{b \ln(a)}$ )

$$E1 = \frac{-\sqrt{a} + e^{3b} + \sqrt{a^2 + ab}}{2a + |b|}, \quad E2 = \frac{5^2 + a^{3b} + \sqrt{e^a + c}}{3\sqrt{b}}$$

3. Effectuer les conversions suivantes (les réponses doivent être justifiées):

$$(127)_{10} = (?)_2 = (?)_8, \quad (B7C)_{16} = (?)_{10}$$

4. a) Classer ce qui suit dans les trois catégories suivantes : 1. Systèmes d'exploitation 2.

Langages de programmation 3. Logiciels spécialisés.

Pascal, Microsoft office, Lecteur Media, Windows vista, Google chrome, Mac OS, Adobe, C++, Matlab, Unix, WinRAR.

- b) Citer 3 noms de systèmes d'exploitation pour PC

### Exercice 02 :

1. a) Classer ce qui suit dans les trois catégories suivantes : 1. Systèmes d'exploitation 2.

Langages de programmation 3. Logiciels spécialisés.

Pascal, Microsoft word, Lecteur Media, Windows vista, Google chrome, Mac OS, C++, Matlab, Linux.

- b) Quelle est la signification des acronymes suivants : CPU, UAL, RAM, ROM

2. Evaluer les expressions suivantes en respectant l'ordre de priorité des opérateurs :

$$E1 = (a/c) - ((d/2 * a + 4)/4 - c)/2b ; \text{ avec } a = 6, b = 1, c = 1, d = 4$$

$$E2 = (1 > c) \text{ AND } \text{NOT}(7 \text{ MOD } a * 2 <= b) \text{ OR } (b >= 8) \text{ AND } (a = b) ; \text{ avec}$$

$$a = 6, b = 3, c = 1$$

3. Traduire les expressions suivantes en langage Pascal

$$E1 = \frac{-\sqrt{a} + e^{3b} + \sqrt{a^2 + ab}}{2a + |b|}, \quad E2 = \frac{a^2 + \sqrt{e^a + c}}{3\sqrt{b}}$$

4. Effectuer les conversions suivantes :

$$(120)_{10} = (?)_2 = (?)_8, \quad (10110101111)_2 = (?)_{16}$$

**Exercice 03 :**

1. Citer deux périphériques de chacune des trois catégories suivantes :

Entrée	Sortie	Entrée/Sortie

2. Effectuer les conversions suivantes :

$$(90)_{10} = (?)_2 = (?)_{16}, (10110101011)_2 = (?)_8$$

3. Traduire les expressions suivantes en langage Pascal (on rappelle que  $a^b = e^{b \ln(a)}$ )

$$E1 = \frac{-\sqrt{2a} + e^{3b} + \sqrt{|2a| + b}}{8 + |b|}, \quad E2 = \frac{5^2 + a^{3b} + \sqrt{e^a + c}}{\sqrt{b}}$$

4. Evaluer l'expression suivante en respectant l'ordre de priorité des opérateurs :

$$E1 = (a/c) - ((d/2 * a + 4)/4 - c)/2b ; \text{ avec } a = 6, b = 1, c = 1, d = 4$$

5. Donner l'expression arithmétique correspondante à l'expression suivante écrite en Pascal :

$$\exp(\text{sqrt}(x))/(2 * y - 1) + \text{abs}(x) - 1/(\text{sqr}(x) + 3)$$

**Exercice 04 :**

1. Quelles sont les trois structures itératives de base ?

2. Citer deux périphériques d'entrée et deux périphériques de sortie.

3. Quels sont les identificateurs valides et invalides parmi les exemples suivants (les identificateurs non valides doivent être justifiés) : `_EX01` ; `EX_01` ; `EX-01` ; `EX01_` .

4. Evaluer l'expression suivante tout en montrant l'ordre des opérations.

$$(a > 9 \text{ DIV } 4) \text{ AND } (a < > b) \text{ OR NOT } (c = b). \text{ avec } a = 4, b = 8, c = 8.$$

5. Donner l'expression arithmétique correspondante à l'expression suivante écrite en Pascal :

$$\text{sqrt}(\text{abs}(2*x + 1 + x*y)) / (\text{sqr}(x) - 2*x*y) + \text{sqrt}(4*x + 5*x).$$

6. Effectuer les conversions suivantes :  $(1EA)_{16} = (?)_8$  ,  $(240)_{10} = (?)_2$  .

## CHAPITRE II



# Notion d'algorithme et de programme

## Chapitre II : Notion d'algorithme et de programme

### II.1. Objectif de ce chapitre

A l'issu de ce chapitre, l'apprenant sera capable de :

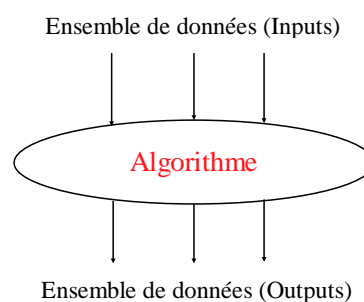
- Comprendre la définition et le rôle des algorithmes dans la résolution de problèmes informatiques.
- Identifier les différentes étapes d'un algorithme, de la spécification à l'exécution.
- Appréhender les concepts de variables, de types de données et d'opérations élémentaires dans la programmation.
- Analyser des exemples concrets d'algorithmes simples et comprendre comment ils sont traduits en programmes exécutables.

### II.2. Concept d'un algorithme

Un Algorithme est une **séquence d'instructions ordonnées**, qui permet de **résoudre un problème**. Le terme "Algorithm" vient de l'arabe الخوارزمي, nom du mathématicien perse Al-Khwarizmi.

Un algorithme prend, en entrée, un ensemble de données (Inputs) et délivre (produit, renvoie) un ensemble de données en sortie (Outputs), afin de résoudre un problème.

Un algorithme peut être schématisé comme suit :



**Figure II.1** : Structure d'un algorithme

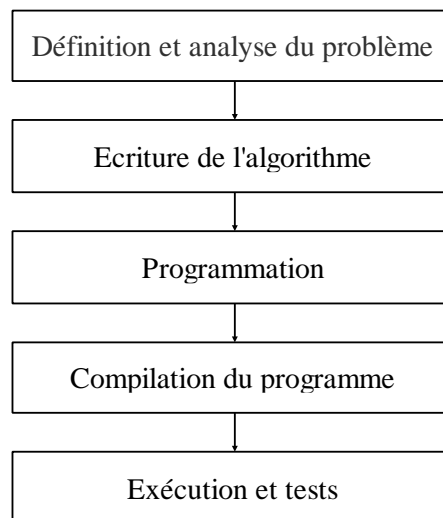
Donc, un algorithme représente une solution pour un problème donné. Cette solution est spécifiée à travers un ensemble d'instructions (séquentielles avec un ordre logique) qui manipulent des données. Une fois l'algorithme est écrit (avec n'importe quelle langue : français, anglais, arabe, etc.), il sera transformé, après avoir choisi un langage de programmation, en un programme code source qui sera compilé (traduit) et exécuté par l'ordinateur.



Pour le langage de programmation qui sera utilisé, ça sera le langage **PASCAL**.

### II.3. Les étapes de résolution d'un problème

La résolution d'un problème, en informatique, passe par différentes étapes. Ces dernières sont schématisées dans la figure II.2.



**Figure II.2 :** Etapes de résolution d'un problème

- **Définition et analyse du problème :** L'objectif de cette étape est de bien comprendre l'énoncé du problème et de déterminer :
  - Les résultats attendus (sorties),
  - Le traitement à effectuer (méthodes et formules de calculs pour atteindre le résultat),
  - Les données nécessaires au traitement (entrées).
- **Écriture de l'algorithme :** Une fois l'analyse terminée, il faut écrire les instructions dans leur ordre logique d'exécution et obtenir ainsi un algorithme. Ce dernier peut être écrit dans n'importe quelle langue.
- **Programmation :** C'est la traduction de l'algorithme en programme en utilisant un langage de programmation.
- **Langage de programmation :** C'est l'ensemble de la syntaxe (commandes et instructions) utilisé par les développeurs afin d'écrire un **programme**. Il existe plusieurs langages de programmation, comme C/C++, Pascal, Java, Fortran, Matlab, etc.
- **Compilation du programme :** Désigne le procédé de traduction d'un programme, écrit et lisible par un humain (code source), en un programme exécutable par un ordinateur (code binaire).

- **Exécution et tests** : Une fois compilé, un programme doit être testé pour s'assurer de son fonctionnement et qu'il répond aux besoins exprimés par l'utilisateur. Un programme est testé par des valeurs différentes de données (jeu de test).

## II.4. Structure d'un algorithme

Un algorithme se compose de trois parties principales : L'entête, la partie déclarative et le corps de l'algorithme.

- *L'Entête* : Dans cette partie on déclare le nom de l'algorithme à travers un identificateur.
- *La partie déclarative* : Dans cette partie on déclare toutes les données utilisées par l'algorithme.
- *Le corps de programme* : Représente la séquence d'actions (instructions) Pour écrire un algorithme.

Ces parties doivent respecter une syntaxe bien déterminée, définie comme suit :

**Tableau II.1** : Structure d'un algorithme

Algorithme	
L'Entête	<b>Algorithme</b> <identificateur_algo> ;
La partie déclarative	<b>Constantes</b> < identificateur_constante> = valeur ; <b>Variables</b> < identificateur_variable> : Type;
Le corps de programme	<b>Début</b> <Instruction 1> ; <Instruction 2> ; .. <Instruction n> ; <b>Fin.</b>

**Tableau II.2** : Structure d'un algorithme en Pascal

Pascal	
L'Entête	<b>Program</b> <identificateur_algo> ;
La partie déclarative	<b>Const</b> < identificateur_constante> = valeur ; <b>Var</b> < identificateur_variable> : Type;
Le corps de programme	<b>Begin</b> <Instruction 1> ; <Instruction 2> ; .. <Instruction n> ; <b>End.</b>

### II.4.1. L'Entête

L'entête sert à donner un nom à l'algorithme en utilisant un identificateur. Ce dernier est précédé par le mot clé "**Algorithme**". Alors qu'est-ce qu'un identificateur ?

- **Identificateur** : Un identificateur est une chaîne de caractère qui permet de donner un nom unique à un programme (algorithme), une constante, une variable, une procédure ou une fonction. Cette chaîne doit commencer soit par un caractère alphabétique ou par un tiret du 8 ( \_ ) et ne peut contenir que des caractères alphanumériques. Aussi, les mots réservés (mots-clés) d'un langage de programmation ne peuvent être utilisés comme identificateurs. Voici quelques mots réservés au langage Pascal: *Begin, end, program, var, const, real, integer, char, if, then, else, while, for, do, repeat*.

#### Exemples :

**Tableau II.3** : Exemples d'identificateurs valides et non valides

Identificateur valide	Identificateur non valide
a1	x1 y (à cause du blanc ou l'espace)
a_1	x1-y (à cause du signe )
A_1	1xy (commence par un caractère numérique)
x12y	
x1_y	

### II.4.2. La partie déclarative

La partie déclarative sert à déclarer les différentes données que l'algorithme utilise (Constantes, variables,.. etc.). Une donnée non déclarée et utilisée par l'algorithme engendre une erreur lors de la compilation. Alors qu'est-ce qu'une variable et qu'est-ce qu'une constante ?

- **Constantes** : Une constante est un objet contenant une valeur qui ne peut jamais être modifiée. Son objectif est d'éviter d'utiliser une valeur d'une manière direct. Imaginons qu'un algorithme utilise la valeur *3.14* une dizaines de fois (le nombre d'occurrences de la valeur *3.14* est par exemple *15*) et qu'on veut modifier cette valeur par une autre valeur plus précise : *3.14159*. Dans ce cas on est amené à modifier toutes les occurrences de *3.14*. Par contre, si on utilise une constante  $PI = 3.14$  on modifier une seule fois cette constante.

**Exemples :**

Ci-dessous quelques exemples de déclarations de constantes :

**Const** PI = 3.14;      Constante réelle.  
**Const** MAX = 10;      Constante entière.  
**Const** cc = 'a';      Constante caractère.  
**Const** ss = 'algo';      Constante chaîne de caractère.  
**Const** b1 = true;      Constante booléenne.  
**Const** b2 = false;      Constante booléenne.

- **Variables** : Une variable est un objet contenant une valeur pouvant être modifiée.

Le tableau II.4 résume les 5 types de variables.

**Tableau II.4** : Les 5 types de base

Algorithme	Pascal	Valeurs
Entier	Integer	Représente l'ensemble {..., 4, 3, 2, 1, 0, 1, 2, 3, 4, ...}
Réel	Real	Représente les valeurs numériques fractionnels et avec des virgule fixes (ou flottante)
Booléen	Boolean	Représente les deux valeurs <i>TRUE</i> et <i>FALSE</i> .
Caractère	Char	Représente tous les caractères imprimable.
Chaîne de caractères	String	Une séquence d'un ou plusieurs caractères

**Exemples :**

**Tableau II.5** : Exemples de variables

Algorithme	Pascal	Signification
x : réel	x : real;	variable réelle
n, m : entier	n, m : integer;	deux variables entières
s : chaîne de caractères	s : String;	variables chaîne de caractères
b1, b2, b3 : booleen	b1, b2, b3 : boolean;	3 variables booléennes
c1 : caractère	c1 : char;	variable caractère

❖ **Remarques**

- Pour commenter un programme Pascal, on écrit les commentaires entre les accolades { }. Par exemple : {Ceci est un commentaire}.
- Dans un programme Pascal, on déclare les constantes dans une section qui commence par le mot clé **const**.
- Dans un programme Pascal, on déclare les variables dans une section qui commence par le mot

clé **var**.

- En plus des constantes et des variables, il est possible de déclarer de nouveaux types, des étiquettes, et (dans un programme Pascal) des fonctions et des procédures.

### II.4.3. Le corps de programme

Le corps d'un algorithme est constitué d'un ensemble d'actions / instructions ordonnées de manière séquentielle et logique. Ces instructions se divisent en cinq types distincts:

- **Lecture** : Cette opération consiste à introduire des données dans l'algorithme. Une lecture consiste à donner une valeur arbitraire à une variable.
- **Écriture** : Cette opération implique l'affichage de données. Elle permet d'afficher des résultats ou des messages.
- **Affectation** : Elle permet de modifier les valeurs des variables en leur assignant de nouvelles valeurs.
- **Structures de contrôle** : Ces structures permettent de modifier la séquentialité de l'algorithme. Elles sont utilisées pour sélectionner différents chemins d'exécution ou pour répéter un traitement.
  - Structure de **Test alternatif simple / double**
  - Structure répétitives (itérative)
    - ✚ La boucle **Pour**
    - ✚ La boucle **Tant-que**
    - ✚ La boucle **Répéter**

Dans le langage Pascal, chaque instruction se termine par un **point-virgule**. Sauf à la fin du programme, on met un **point**.

## II.5. Types d'instructions

Une instruction spécifie une opération ou un enchaînement d'opérations à exécuter sur des objets (constante, variable, .. etc.).

Les instructions se situent entre les mots-clés Début (Begin) et Fin (End.).

Les instructions sont séparées par des ; et sont exécutées séquentiellement, c'est-à-dire l'une après l'autre, depuis le Begin jusqu'au End. final.

### II.5.1. Instructions d'Entrées/Sorties (Lecture / Écriture)

#### II.5.1.1. Entrées (Lecture)

Une instruction d'entrée est une instruction qui permet d'introduire (saisir) une valeur à

l'Algorithme à travers le clavier.

La syntaxe d'une lecture est comme suit :

**Tableau II.6 :** La syntaxe d'une lecture

Algorithme	Pascal
<b>Lire</b> (id_var) ;	<b>read</b> (id_var) ; <b>Readln</b> (id_var) ; {Lecture avec retour à la ligne.}
<b>Lire</b> (id_var1, id_var2, ..., id_varN) ; {Lecture de plusieurs variables en même temps.}	<b>Read</b> (id_var1, id_var2, id_var3, ..., id_varN) ;

❖ **Remarque :** Il est important de noter que l'instruction de lecture concerne uniquement les variables, on peut pas lire des constantes ou des valeurs. Lors de la lecture d'une variable dans un programme Pascal, le programme se bloque en attendant la saisie d'une valeur via le clavier. Une fois la valeur saisie, on valide par la touche *entrée*, et le programme reprend l'exécution avec l'instruction suivante.

### Exemples :

**Tableau II.7 :** Exemples d'entrées

Algorithme	Pascal
Lire (a, b, c) lire (hauteur)	read(a, b, c); read(hauteur);

### II.5.1.2. Sorties (Écriture)

L'écriture est une instruction qui permet d'afficher, à l'écran, des données. Ces dernières peuvent être un message, une valeur, la valeur d'une variable, une constante et même le résultat d'un calcul.

La syntaxe d'une écriture est comme suit :

**Tableau II.8 :** La syntaxe d'une écriture

Algorithme	Pascal
<b>Écrire</b> (id_var) ;	<b>Write</b> (id_var) ; <b>Writeln</b> (id_var) ; {Écriture avec retour à la ligne.}
<b>Écrire</b> (' Ceci est un message ! '); {Affichage du message : Ceci est un message ! à l'écran.}	<b>Write</b> (' Ceci est un message ! ');

❖ **Remarque :** Il est à noter que l'instruction d'écriture ne concerne pas uniquement les variables,

on peut écrire des constantes, valeurs ou des expressions (arithmétiques ou logiques). On peut afficher une valeur et sauter la ligne juste après à travers l'instruction : **writeln**.

### Exemples :

**Tableau II.9 :** Exemples de sorties

Algorithme	Pascal	Signification
écrire('Bonjour')	write ('Bonjour');	Afficher le message Bonjour
écrire(a, b, c)	write(a, b, c);	Afficher les valeurs des variables a, b et c
écrire(5+2)	write(5+2);	Afficher le résultat du calcul c-à-d 7
écrire(a+b-c)	write(a+b-c);	Afficher le résultat de l'évaluation de l'expression : a+b-c
écrire(5<2)	write(5<2);	Afficher le résultat de l'évaluation de l'expression 5 < 2 c-à-d false.

❖ **Remarque :** En Pascal, il est possible de commenter un programme, il suffit d'écrire les commentaires entre accolades { } ou de mettre deux slashes // avant le commentaire.

Par exemple : {Ceci est un commentaire} ou // Ceci est un commentaire.

Le commentaire n'est pas pris en compte à la compilation. Il sert à rendre le programme plus clair à la lecture, à noter des remarques, etc

### II.5.2. Instruction d'affectation

L'affectation est une instruction qui permet de modifier la valeur d'une variable.

La syntaxe d'une affectation est :

**Tableau II.10 :** La syntaxe d'une affectation

Algorithme	Pascal
$a \leftarrow b$ ; {a : <b>DOIT</b> être une variable. b peut être une valeur, une constante, une variable ou une expression.}	a :=b ;

❖ **Remarque :** Les deux côtés d'une affectation doivent être du même type sauf pour le type entier qui peut être stocké dans un réel car l'ensemble des réels inclut l'ensemble des entiers.

**Exemples :****Tableau II.11 :** Exemples d'affectation

Algorithme	Pascal	Signification
$a \leftarrow 5$	$a:=5;$	Mettre la valeur 5 dans la variable a
$b \leftarrow a+5$	$b:=a+5;$	Mettre la valeur de l'expression a+5 dans la variable B
$sup \leftarrow a>b$	$sup:=a>b;$	a>b donne un résultat booléen, donc sup est une variable booléenne

**II.5.3. Structures de contrôles**

En générale, les instructions d'un programme sont exécutés d'une manière séquentielle : la première instruction, ensuite la deuxième, après la troisième et ainsi de suite. Cependant, dans plusieurs cas, on est amené soit à choisir entre deux ou plusieurs chemins d'exécution (un choix entre deux ou plusieurs options), ou bien à répéter l'exécution d'un ensemble d'instructions, pour cela nous avons besoins de structures de contrôle pour contrôler et choisir les chemins d'exécutions ou refaire un traitement plusieurs fois. Les structures de contrôle sont de deux types : Structures de contrôles conditionnelles et structures de contrôle répétitives (itératives).

**II.5.3.1. Structures de contrôle conditionnelle**

Ces structures sont utilisées pour décider ou pas de l'exécution d'un ou de plusieurs instructions en testant (vérifiant) une ou plusieurs condition. On en distingue deux types de tests : le test alternatif simple et double.

**a) Test alternatif simple**

Ce test contient un seul bloc d'instructions. Selon une condition (expression logique), on décide est ce que le bloc d'instructions est exécuté ou non. Si la condition est vraie, on exécute le bloc, sinon on l'exécute pas.

La syntaxe d'un test alternatif simple est comme suit :

**Tableau II.12 :** La syntaxe d'un test alternatif simple

Algorithme	Pascal
<b>si</b> <condition> <b>alors</b> <instruction(s)> <b>fin</b> si;	<b>if</b> <condition> <b>then</b> <b>begin</b> <instruction(s)>; <b>end</b> ;



**Exemples :****Tableau II.13 :** Exemple d'un test alternatif simple

Algorithme	Pascal
<pre>lire(x) <b>si</b> x &gt; 2 <b>alors</b>   x ← x + 3 ; <b>finsi</b> ; écrire (x)</pre>	<pre>read(x); <b>if</b> x &gt; 2 <b>then</b>   <b>begin</b>     x:= x + 3;   <b>end</b>; write(x);</pre>

❖ **Remarque :** En Pascal, le bloc d'instructions à exécuter après if (juste après then) **DOIT** être délimité par un **begin** et **end**.

Si le bloc contient une seule instruction, **begin** et **end** sont facultatifs (on peut les enlever).

**b) Test alternatif double**

Ce test contient deux blocs d'instructions. Le premier bloc est exécuté lorsque la condition est vérifiée (vraie) et le second lorsque la condition n'est pas vérifiée (fausse).

La syntaxe d'un test alternatif double est :

**Tableau II.14 :** La syntaxe d'un test alternatif double

Algorithme	Pascal
<pre><b>si</b> &lt;condition&gt; <b>alors</b>   &lt;instruction(s)1&gt; <b>sinon</b>   &lt;instruciton(s)2&gt; ; <b>finsi</b> ;</pre>	<pre><b>if</b> &lt;condition&gt; <b>then</b>   <b>begin</b>     &lt;instruction(s)1&gt; ;   <b>end</b> <b>else</b>   <b>begin</b>     &lt;instruction(s)2&gt; ;   <b>end</b> ;</pre>

**Exemple :****Tableau II.15 :** Exemples d'un test alternatif double

Algorithme	Pascal
<pre>lire(x) <b>si</b> x &gt; 2 <b>alors</b>   x ← x + 3 <b>sinon</b>   x ← x - 2 <b>finsi</b> écrire (x)</pre>	<pre>read(x); <b>if</b> x &gt; 2 <b>then</b>   <b>begin</b>     x:= x + 3;   <b>end</b> <b>else</b>   <b>begin</b>     x:= x - 2;   <b>end</b>; write(x);</pre>

❖ **Remarques :**

- En Pascal, l'instruction qui précède **else** ne doit pas contenir un pointvirgule ( ; ).
- Dans l'exemple précédent, on peut enlever **begin** et **end** du **if** et ceux du **else** puisqu'il y a une seule instruction dans les deux blocs.

**II.5.3.2. Structures de contrôle répétitives**

Les structures répétitives nous permettent de répéter un traitement un nombre fini de fois. Par exemple, on veut afficher tous les nombre premier entre 1 et N (N nombre entier positif donné).

Nous avons trois types de structures itératives (boucles) :

**a) Boucle Pour (For)**

La structure de contrôle répétitive pour (for en langage Pascal) utilise un indice entier qui varie (avec un incrément = 1) d'une valeur initiale jusqu'à une valeur finale. À la fin de chaque itération, l'indice est incrémenté de 1 d'une manière automatique (implicite).

La syntaxe de la boucle **pour** est comme suit :

**Tableau II.16 :** La syntaxe de la boucle **pour**

Algorithme	Pascal
<p><b>pour</b> &lt;indice&gt; ← &lt;vi&gt; à &lt;vf&gt; faire            &lt;instruction(s)&gt; ;  <b>finPour</b>;</p>	<p><b>for</b> &lt;indice&gt; := &lt;vi&gt; to &lt;vf&gt; do            <b>begin</b>            &lt;instruction(s)&gt; ;            <b>end</b>;</p>

<indice> : variable entière

<vi> : valeur initiale                      <vf> : valeur finale

La boucle pour contient un bloc d'instructions (les instructions à répéter). Si le bloc contient une seule instruction, le **begin** et **end** sont facultatifs.

Le bloc sera répété un nombre de fois = (<vf> - <vi> + 1) si la valeur finale est supérieure ou égale à la valeur initiale. Le bloc sera exécuté pour <indice> = <vi>, pour <indice> = <vi>+1, pour <indice> = <vi>+2, ..., pour <indice> = <vf>.

❖ **Remarque :** Il ne faut jamais mettre de point-virgule après le mot clé **do**. (erreur logique)

**b) Boucle Tant-que (While)**

La structure de contrôle répétitive **tantque** (**while** en langage Pascal) utilise une expression

logique ou booléenne comme condition d'accès à la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) donc on entre à la boucle, sinon on la quitte.

La syntaxe de la boucle **tantque** est comme suit :

**Tableau II.17** : La syntaxe de la boucle **tantque**

Algorithme	Pascal
<b>tant-que</b> <condition> faire <instruction(s)> ; <b>fin tant-que</b> ;	<b>while</b> <condition> <b>do</b> <b>begin</b> <instruction(s)>; <b>end</b> ;

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions Tant que la condition est vraie. Une fois la condition est fausse, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après fin Tant que (après **end**).

Comme la boucle **for**, il faut jamais mettre de point-virgule après **do**.

❖ **Remarque** : Il est possible de remplacer toute boucle "**pour**" par une boucle "**tantque**", cependant, l'inverse n'est pas toujours réalisable.

### c) Boucle Répéter (Repeat)

La structure de contrôle répétitive **répéter** (**repeat** en langage Pascal) utilise une expression logique ou booléenne comme condition de sortie de la boucle : si la condition est vérifiée (elle donne un résultat vrai : TRUE) on sort de la boucle, sinon on y accède (on répète l'exécution du bloc).

La syntaxe de la boucle **répéter** est comme suit :

**Tableau II.18** : La syntaxe de la boucle **répéter**

Algorithme	Pascal
<b>répéter</b> <instruction(s)> ; <b>Jusqu'à</b> <condition> ;	<b>repeat</b> <instruction(s)>; <b>until</b> <condition> ;

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions jusqu'à avoir la condition correcte. Une fois la condition est vérifiée, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après **jusqu'à** (après **until**). Dans la boucle **repeat** on utilise pas **begin** et **end** pour délimiter le bloc d'instructions (le bloc est déjà délimité par **repeat** et **until**).

La différence entre la boucle **répéter** et la boucle **tantque** est :

- La condition de **répéter** est toujours l'inverse de la condition **tantque** : pour **répéter** c'est la condition de sortie de la boucle, et pour **tantque** c'est la condition d'entrer.
- Le teste de la condition est à la fin de la boucle (la fin de l'itération) pour **répéter**. Par contre, il est au début de l'itération pour la boucle **tantque**. C'est-à-dire, dans **tantque** on teste la condition avant d'entrer à l'itération, et dans **répéter** on fait l'itération après on teste la condition.

### II.5.3.3. Structure de contrôle de branchements / sauts (l'instruction Goto)

Une instruction de branchement nous permet de sauter à un endroit du programme et continuer l'exécution à partir de cet endroit. Pour réaliser un branchement, il faut tout d'abord indiquer la cible du branchement via une étiquette `<num_etiq> : .` . Après on saute à cette endroit par l'instruction aller à `<num_etiq>` (en pascal : `goto <num_etiq>`).

La syntaxe d'un branchement est comme suit :

**Tableau II.19** : La syntaxe d'un branchement

Algorithme	Pascal
<pre> <b>aller à</b> &lt;num_etiq&gt; . . . &lt;num_etiq&gt; : . . . </pre>	<pre> <b>goto</b> &lt;num_etiq&gt;; . . . &lt;num_etiq&gt; : . . . </pre>

#### N.B :

- Une étiquette représente un numéro (nombre entier), exemple : 1, 2, 3, etc.
- Dans un programme Pascal, il faut déclarer les étiquettes dans la partie déclaration avec le mot clé **label**. (on a vu **const** pour les constantes **var** pour les variables)
- Une étiquette désigne un seule endroit dans le programme, on peut jamais indiquer deux endroits avec une même étiquette.
- Par contre, on peut réaliser plusieurs branchement vers une même étiquette.
- Un saut ou un branchement peut être vers une instruction antérieure ou postérieure (avant ou après le saut).

**Exemple :****Tableau II.20 :** Exemple de branchement

Algorithme	Pascal
<b>algorithme</b> branchement <b>variables</b> a, b, c : entier ; <b>début</b> lire (a, b); 2: c ← a; <b>si</b> (a > b) <b>alors</b> <b>aller à 1;</b> <b>finsi</b> a ← a + 5; <b>aller à 2;</b>  1: écrire (c); <b>fin</b>	<b>program</b> branchement; <b>uses</b> wincrt; <b>var</b> a, b, c:integer; <b>label</b> 1, 2; <b>begin</b> read(a,b); 2: c:=a;  <b>if</b> (a>b) <b>then</b> <b>goto</b> 1; a := a + 5; <b>goto</b> 2;  1: write(c); <b>end.</b>

Dans l'exemple ci-dessus, il y a deux étiquettes : **1** et **2**. L'étiquette **1** fait référence la dernière instruction de l'algorithme/programme (écrire(c) /write(c)), et l'étiquette **2** fait référence la troisième instruction de l'algorithme / programme ( $c \leftarrow a$  /  $c := a$ ). Pour le déroulement de l'algorithme, on utilise le tableau suivant ( $a = 2$  et  $b = 5$ ) :

**Tableau II.21 :** Déroulement de l'algorithme

Variables / Instructions	a	b	c
Lire (a, b) Donner deux valeur quelconque à a et b	2	5	?
$c \leftarrow a$ ;	2	5	2
$a > b \rightarrow$ false puisque $a = 2$ et $b = 5$ on entre pas au bloc du si $a \leftarrow a + 5$ ;	7	5	2
aller à 2 $c \leftarrow a$ ;	7	5	7
$a > b \rightarrow$ true puisque $a = 7$ et $b = 5$ on entre au bloc du si aller à 1 => écrire (c)	7	5	7 (résultat affiché)

Il y a deux types de branchement :

- a. **branchement inconditionnel** : c'est un branchement sans condition, il n'appartient pas à un bloc de *si* ou un bloc *sinon*. Dans l'exemple précédent, l'instruction **aller à 2 (goto 2)** est un saut inconditionnel.
- b. **branchement conditionnel** : Par contre, un branchement conditionnel est un saut qui appartient à un bloc *si* ou un bloc *sinon*. L'instruction **aller à 1 (goto 1)**, dans l'exemple précédent est un saut conditionnel puisque il appartient un bloc *si*.

## II.6. Correspondance Algorithme-Pascal

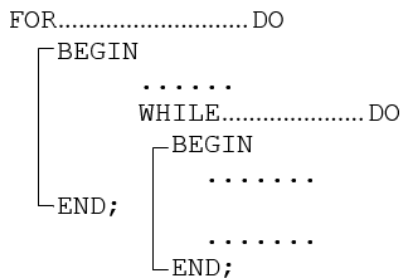
Pour traduire un algorithme en programme Pascal, on utilise le tableau récapitulatif suivant pour traduire chaque structure syntaxique d'un algorithme en structure syntaxique du Pascal.

**Tableau II.22** : Correspondance Algorithme-Pascal

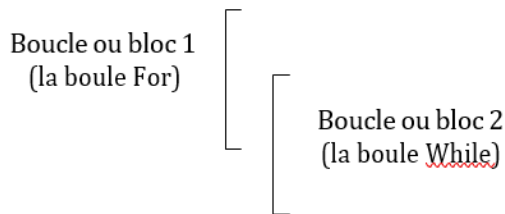
Vocabulaire / Syntaxe Algorithmique	Vocabulaire / Syntaxe du PASCAL
Algorithme	Program
Constantes	Const
Type	Type
Variables	Var
Etiquette	Label
Entier	Integer
Réel	Real
Caractère	Char
Booléen	Boolean
Chaîne de Caractères	String
Fonction	Function
Procédure	Procedure
Début	Begin
Fin	End
Si ... Alors ... Sinon ...	If ... Then ... Else ...
Tant-que ... Faire ...	While ... Do ...
Pour i ← 1 à N Faire ...	For i:= 1 To N Do ...
Pour i ← N à Pas- 1 Faire ...	For i:= 1 DownTo 1 Do ...
Répéter ... Jusqu'à ...	Repeat .... Until ...

❖ **Remarques importantes**

1. Langage Pascal est insensible à la casse, c'est-à-dire, si on écrit begin, Begin ou BEGIN c'est la même chose.
2. Lorsque l'action après THEN, ELSE ou un DO comporte plusieurs instructions, on doit obligatoirement encadrer ces instructions entre BEGIN et END. Autrement dit, on les définit sous forme d'un bloc. Pour une seule instruction, il n'est pas nécessaire (ou obligatoire) de l'encadrer entre BEGIN et END (voir en travaux pratiques). Un ensemble d'instructions encadrées entre BEGIN et END, s'appelle un BLOC ou action composée. On dit qu'un programme Pascal est structurée en blocs.
3. Il est interdit de chevaucher deux structures de boucles ou de blocs. Par exemple :



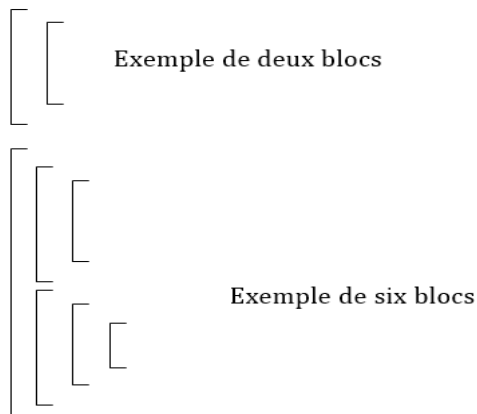
On a eu la forme suivante :



Ce qui est interdit.

Les boucles et blocs doivent en aucun cas chevaucher, ils doivent être imbriqués.

**Exemples de structures autorisées :**



## II.7. Représentation en organigramme

Un organigramme est la représentation graphique de la résolution d'un problème. Il est similaire à un algorithme. Chaque type d'action dans l'algorithme possède une représentation dans l'organigramme.

Il est préférable d'utiliser la représentation algorithmique que la représentation par organigramme notamment lorsque le problème est complexe.

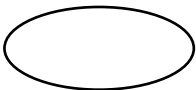
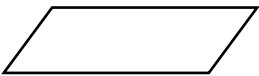
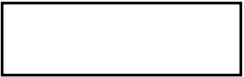
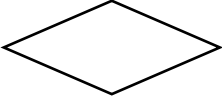


Les inconvénients qu'on peut rencontrer lors de l'utilisation des organigrammes sont :

- Quand l'organigramme est long et tient sur plus d'une page,
- Problème de chevauchement des flèches,
- Plus difficile à lire et à comprendre qu'un algorithme.

### II.7. 1. Les symboles d'organigramme

Les symboles utilisés dans les organigrammes sont illustrés dans le tableau II.23.

**Tableau II.23** : Les symboles d'organigramme

	Représente le début et la Fin de l'organigramme
	Entrées / Sorties : Lecture des données et écriture des résultats.
	Calculs, traitements
	Tests et décision : on écrit le test à l'intérieur du losange
	Ordre d'exécution des opérations (Enchaînement)
	Connecteur

## II.8. Représentation des primitives algorithmiques

### II.8.1. L'enchaînement

L'enchaînement permet d'exécuter une série d'actions dans l'ordre de leur apparition. Soit  $A_1, A_2, \dots, A_n$  une série d'actions, leur enchaînement est représenté comme suit :





$A1, A2, \dots, An$  : peuvent être des actions élémentaires ou complexes.

### II.8.2. La structure alternative simple

La syntaxe et l'organigramme de la structure alternative simple sont présentés dans le tableau II.24.

**Tableau II.24** : La syntaxe et l'organigramme de la structure alternative simple

Représentation algorithmique	Représentation sous forme d'organigramme
<p><b>si</b> &lt;condition&gt; <b>alors</b></p> <p style="padding-left: 20px;">&lt;Action(s)&gt;;</p> <p><b>finsi</b>;</p> <p>Si la condition est vérifiée, le bloc &lt;action(s)&gt; sera exécuté, sinon rien, et on continue l'exécution de l'instruction après fin si.</p>	

Les conditions utilisées pour les tests (simple ou double) sont des expressions logiques ou booléennes, ça veut dire des expressions dont leur évaluation donne soit TRUE (Vrai) ou FALSE (faux). Toute comparaison entre deux nombres représente une expression logique. On peut former des expressions logiques à partir d'autres expressions logiques en utilisant les opérateurs suivants : Not, Or et And.

#### Exemple :

$(x \geq 5)$  : est une expression logique, elle est vraie si la valeur de  $x$  est supérieure ou égale à 5. elle est fautive dans le cas contraire.

$\text{Not}(x \geq 5)$  : est une expression logique qui est vraie uniquement si la valeur de  $x$  est inférieure à 5.

$(x \geq 5) \text{ And } (y \leq 0)$  : est une expression logique qui est vraie si  $x$  est supérieure ou égale à 5 et  $y$  inférieure ou égale à 0.

### II.8.3. La structure alternative double

La syntaxe et l'organigramme de la structure alternative double sont présentés dans le tableau II.25.

**Tableau II.25 :** La syntaxe et l'organigramme de la structure alternative double

Représentation algorithmique	Représentation sous forme d'organigramme
<p><b>si</b> &lt;condition&gt; <b>alors</b>            &lt;Action1(s)&gt;  <b>sinon</b>            &lt;Action2(s)&gt;;  <b>finsi</b>;</p> <p>Si la condition est vérifiée, le bloc &lt;action1(s)&gt; sera exécuté, sinon (si elle est fausse) on exécute &lt;action2(s)&gt;.</p>	<pre> graph TD     C{Condition ?} -- Faux --&gt; A2[Action2(s)]     C -- Vrai --&gt; A1[Action1(s)]     A2 --&gt; J(( ))     A1 --&gt; J     J --&gt; Exit[ ]   </pre>

### II.8.4. La structure itérative POUR (Boucle POUR)

La syntaxe et l'organigramme de la structure itérative POUR sont présentés dans le tableau II.26.

**Tableau II.26 :** La syntaxe et l'organigramme de la structure itérative POUR

Représentation algorithmique	Représentation sous forme d'organigramme
<p><b>pour</b> &lt;cpt&gt; ← &lt;vi&gt; <b>à</b> &lt;vf&gt; <b>faire</b>            &lt;Action(s)&gt;;  <b>finpour</b>;</p>	<pre> graph TD     Init[&lt;cpt&gt; ← &lt;vi&gt;] --&gt; Act[Action(s)]     Act --&gt; Inc[&lt;cpt&gt; ← &lt;cpt&gt; + 1]     Inc --&gt; Test{&lt;cpt&gt; ≤ &lt;vf&gt;}     Test -- Vrai --&gt; Init     Test -- Faux --&gt; Exit[ ]   </pre>

Dans la boucle **POUR**, on exécute le bloc <Action(s)> ( $<vf> - <vi> + 1$ ) fois. Ceci dans le cas où <vf> est supérieur ou égal à <vi>. Dans le cas contraire, le bloc d'actions ne sera jamais exécuté. Le déroulement de la boucle POUR est exprimé comme suit :

1. La variable entière <cpt> (le compteur) prends la valeur initiale <vi> ;
2. On compare la valeur de <cpt> à celle de <vf> ; si <cpt> est supérieur à <vf> on sort de la boucle ;
3. Si <cpt> est inférieur ou égal à <vf> on exécute le bloc <Action(s)> ;

4. La boucle POUR incrémente automatiquement le compteur  $\langle cpt \rangle$ , c'est-à-dire elle lui ajoute un ( $\langle cpt \rangle \leftarrow \langle cpt \rangle + 1$ );
5. On revient à 2 (pour refaire le teste  $\langle cpt \rangle \leftarrow \langle vi \rangle$  C'est pour cela qu'on dit la boucle);

❖ **Remarque :** La boucle **POUR** est souvent utilisée pour les structures de données itératives (les tableaux et les matrices – variables indicées).

### II.8.5. La structure itérative Tant-que (Boucle Tant-que)

La syntaxe et l'organigramme de la structure itérative Tant-que sont présentés dans le tableau II.27.

**Tableau II.27 :** La syntaxe et l'organigramme de la structure itérative Tant-que

Représentation algorithmique	Représentation sous forme d'organigramme
<p><b>Tant-que</b> <math>\langle condition \rangle</math> <b>faire</b></p> <p style="padding-left: 40px;"><math>\langle Action(s) \rangle</math>;</p> <p><b>finpour</b>;</p>	

On exécute le bloc d'instructions  $\langle Action(s) \rangle$  tant que la  $\langle condition \rangle$  est vérifiée (c'est-à-dire elle est vraie). Le déroulement de la boucle est comme suit :

1. On évalue la condition : si la condition est fausse on sort de la boucle ; Si la condition est vraie, on exécute le bloc  $\langle Action(s) \rangle$  ;
2. On revient à 1 ;
3. On continue la suite de l'algorithme

### II.8.6. La structure itérative Répéter (Boucle Répéter)

La syntaxe et l'organigramme de la structure itérative répéter sont présentés dans le tableau II.28.

**Tableau II.28 :** La syntaxe et l'organigramme de la structure itérative répéter

Représentation algorithmique	Représentation sous forme d'organigramme
<p><b>Répéter</b></p> <p style="padding-left: 40px;"><math>\langle Action(s) \rangle</math>;</p> <p><b>Jusqu'à</b> <math>\langle condition \rangle</math>;</p>	

On répète l'exécution du bloc  $\langle \text{Action(s)} \rangle$  jusqu'à avoir la condition correcte. Le déroulement est comment suit :

1. On exécute le bloc  $\langle \text{Action(s)} \rangle$  ;
2. On évalue la condition : si la condition est vérifiée (elle est vraie) on sort de la boucle (on continue la suite de l'algorithme);
3. Si la condition n'est pas vérifiée (elle est fausse) on revient à 1.

❖ **Remarques :**

- ✓ N'importe quelle boucle **POUR** peut être remplacée par une boucle **Tant-Que**, cependant l'inverse n'est pas toujours correcte, c'est-à-dire, il y a des cas où la boucle **Tant-Que** ne peut pas être remplacée par une boucle **POUR**.
- ✓ On transforme une boucle **POUR** à une boucle **Tant-Que** comme suit :

**Tableau II.29 :** Transformation de la boucle POUR à la boucle Tant-que

Boucle <b>POUR</b>	Boucle <b>Tant-Que</b>
<p><b>pour</b> <math>\langle \text{cpt} \rangle \leftarrow \langle \text{vi} \rangle</math> <b>à</b> <math>\langle \text{vf} \rangle</math> <b>faire</b></p> <p style="padding-left: 40px;"><math>\langle \text{action(s)} \rangle</math>;</p> <p><b>Finpour;</b></p>	<p><math>\langle \text{cpt} \rangle \leftarrow \langle \text{vi} \rangle</math>;</p> <p><b>Tant-que</b> <math>\langle \text{cpt} \rangle \leftarrow \langle \text{vf} \rangle</math> <b>faire</b></p> <p style="padding-left: 40px;"><math>\langle \text{Action(s)} \rangle</math>;</p> <p style="padding-left: 40px;"><math>\langle \text{cpt} \rangle \leftarrow \langle \text{cpt} \rangle + 1</math>;</p> <p><b>Fin Tant-Que;</b></p>

- ✓ La boucle Répéter possède une condition de sortie (c'est-à-dire si elle est vraie on sort de la boucle), alors que la boucle Tant-que possède une condition d'entrée (c'est-à-dire si elle est vraie on entre dans la boucle).
- ✓ La boucle Répéter exécute le bloc  $\langle \text{Action(s)} \rangle$  au moins une fois, le teste vient après l'exécution du bloc.
- ✓ La boucle Tant-Que peut ne pas exécuter le bloc  $\langle \text{Action(s)} \rangle$  (dans le cas où la condition est fausse dès le début), puisque le teste est avant l'exécution du bloc.

## II.9. Exercices corrigés

### Exercice N°01 : (Type de variables)

Donner le type des variables suivantes : 2010 ; 124.5 ; 667.0E-8 ; 'A' ; TRUE ; False ; 'division par zéro'

### Corrigé de l'exercice N°01 :

Type des variables

Variable	Type
2010	Entier / Integer
124.5	Réel / Real
667.0E-8	Réel / Real
'A'	Caractère / Char
TRUE	Booléen / Boolean
False	Booléen / Boolean
'division par zéro'	Chaîne de caractère / String

### Exercice N°02 : (Identificateurs)

Identifier les identificateurs valides et non valides : 1A ; R? ; K2 ; T280 ; 12R ; Hauteur ; Prix-HT ; Prix\_HT ; Exo 04 ; Exo\_04 ; Exo-04 ; Program ; read.

### Corrigé de l'exercice N°02 :

Les variables valides et non valides :

Variable valide	Variable non valide
K2	1A
T280	R?
Hauteur	12R
Prix_HT	Prix-HT
Exo_04	Exo 04
	Exo-04
	Program
	read

### Exercice N°03 : (Enoncé du problème → Algorithme → Programme)

Écrire un algorithme, puis traduit le en programme PASCAL, pour chacun des problèmes suivants :

- 1) Permuter entre les deux variables X et Y ?
- 2) Permuter entre les trois variables X, Y et Z de telle sorte que la valeur de X soit dans Y, celle de Y dans Z et la valeur de Z dans X ?

- 3) Calculer la division entre deux nombres réels a et b ? (**NB** : sans faire de condition, c-à-d :  $b \neq 0$ )
- 4) Calculer le quotient et le reste de la division euclidienne de a par b ?
- 5) Convertir en octets un nombre donné en bits ?

### Corrigé de l'exercice N°03 :

- 1) Permuter entre les deux variables X et Y ?

Algorithme	Programme PASCAL
<b>Algorithme</b> Exo2_1; <b>Variables</b> x, y, t : entier;  <b>Début</b> {--*-- Entrées --*--} <b>Lire</b> (x, y) ;  {--*-- Traitement --*--} t ← x; x ← y; y ← t;  {--*-- Sorties --*--} Écrire('x=', x, 'y=', y) ; <b>Fin.</b>	<b>Program</b> Exo2_1; <b>Var</b> x, y, t : integer;  <b>Begin</b> {--*-- Entrées --*--} <b>Read</b> (x,y) ;  {--*-- Traitement --*--} t := x; {on conserve la valeur de X dans t} x := y; {pas de risque de perte de valeur} y := t; {on récupère l'ancienne valeur de X}  {--*-- Sorties --*--} Write('x=', x, 'y=', y); <b>End.</b>

- 2) Permuter entre les trois variables X, Y et Z de telle sorte que la valeur de X soit dans Y, celle de Y dans Z et la valeur de Z dans X ?

Algorithme	Programme PASCAL
<b>Algorithme</b> Exo2_2; <b>Variables</b> x, y, z, t : entier;  <b>Début</b> {--*-- Entrées --*--} <b>Lire</b> (x, y, z) ;  {--*-- Traitements --*--} t ← y; y ← x; x ← z; z ← t;  {--*-- Sorties --*--} Écrire('x=', x, 'y=', y, 'z=', z); <b>Fin.</b>	<b>Program</b> Exo2_2; <b>Var</b> x, y, z, t : integer;  <b>Begin</b> {--*-- Entrées --*--} <b>Read</b> (x, y, z) ;  {--*-- Traitements --*--} t := y; {on conserve la valeur de y dans t} y := x; {x dans y} x := z; {z dans x} z := t; {y dans z}  {--*-- Sorties --*--} Write('x=', x, 'y=', y, 'z=', z); <b>End.</b>

3) Calculer la division entre deux nombres réels a et b ?

Algorithme	Programme PASCAL
<b>Algorithme</b> Exo2_3; <b>Variables</b> a, b, c : réel;  <b>Début</b> {-*-*- Entrées -*-*-} <b>Lire</b> (a, b) ;  {-*-*- Traitements -*-*-} $c \leftarrow a / b$ ;  {-*-*- Sorties -*-*-} <b>Écrire</b> (c) ; <b>Fin.</b>	<b>Program</b> Exo2_3; <b>Var</b> a, b, c : <b>real</b> ;  <b>Begin</b> {-*-*- Entrées -*-*-} <b>Read</b> (a, b) ;  {-*-*- Traitements -*-*-} $c := a / b$ ;  {-*-*- Sorties -*-*-} <b>Write</b> (c) ; <b>End.</b>

4) Calculer le quotient et le reste de la division euclidienne de a par b ?

Algorithme	Programme PASCAL
<b>Algorithme</b> Exo2_4; <b>Variables</b> a, b, Q, R : entier;  <b>Début</b> {-*-*- Entrées -*-*-} <b>Lire</b> (a, b) ;  {-*-*- Traitements -*-*-} $Q \leftarrow a \text{ div } b$ ; $R \leftarrow a \text{ mod } b$ ;  {-*-*- Sorties -*-*-} <b>Écrire</b> ('Le quotient est : ', Q, 'et le reste est : ', R) ; <b>Fin.</b>	<b>Program</b> Exo2_4; <b>Var</b> a, b, Q, R : <b>integer</b> ;  <b>Begin</b> {-*-*- Entrées -*-*-} <b>Read</b> (a, b) ;  {-*-*- Traitements -*-*-} $Q := a \text{ div } b$ ; $R := a \text{ mod } b$ ;  {-*-*- Sorties -*-*-} <b>Write</b> ('Le quotient est : ', Q, 'et le reste est : ', R) ; <b>End.</b>

5) Convertir en octets un nombre donné en bits ?

Algorithme	Programme PASCAL
<b>Algorithme</b> Exo2_7; <b>Variables</b> bit : <b>integer</b> ; octet : <b>réel</b> ;  <b>Début</b> { <i>-*-*- Entrées -*-*-</i> } <b>Écrire</b> ('Nombres de bits =') ; <b>Lire</b> (bit) ;  { <i>-*-*- Traitements -*-*-</i> } octet ← bit/8;  { <i>-*-*- Sorties -*-*-</i> } <b>Écrire</b> (bit, ' bits =', octet:8:3,' octet'); <b>Fin.</b>	<b>Program</b> Exo2_7; <b>Var</b> bit : <b>integer</b> ; octet : <b>real</b> ;  <b>Begin</b> { <i>-*-*- Entrées -*-*-</i> } <b>Write</b> ('Nombres de bits =') ; <b>Read</b> (bit) ;  { <i>-*-*- Traitements -*-*-</i> } octet := bit/8;  { <i>-*-*- Sorties -*-*-</i> } <b>Write</b> (bit, ' bits =', octet:8:3,' octet'); <b>End.</b>

#### **Exercice N°04 :**

Ecrire un programme Pascal intitulé **ordre\_croissant**, qui permet d'afficher trois valeurs numérique A, B et C avec ordre croissant ?



**Corrigé de l'exercice N°04 :**

Algorithme	Programme PASCAL
<b>Algorithme</b> ordre_croissant; <b>Variables</b> A, B, C : entier; <b>Début</b> {--*--*-- Entrées --*--*--} Écrire( 'Donner trois valeurs entière A, B et C : '); Read(A, B, C) ;  {--*--*-- Traitement & Sorties--*--*--} <b>Si</b> (A <= B) <b>ET</b> (B <= C) <b>alors</b> Écrire(A, B, C); <b>Sinon</b> <b>Si</b> (A <= C) <b>ET</b> (C <= B) <b>alors</b> Écrire (A, C, B) ; <b>Sinon</b> <b>Si</b> (B <= A) <b>ET</b> (A <= C) <b>alors</b> Écrire(B, A, C) <b>Sinon</b> <b>Si</b> (B <= C) <b>ET</b> (C<=A) <b>alors</b> Écrire(B, C, A) <b>Sinon</b> <b>Si</b> (C <= A) <b>ET</b> (A<=B) <b>alors</b> Écrire(C, A, B); <b>Sinon</b> Écrire(C, B, A); <b>Fin-Si</b> ; <b>Fin-Si</b> ; <b>Fin-Si</b> ; <b>Fin-Si</b> ; <b>Fin.</b>	<b>Program</b> ordre_croissant; <b>Var</b> A, B, C : integer; <b>Begin</b> {--*--*-- Entrées --*--*--} <b>Write</b> ( 'Donner trois valeurs entière A, B et C : '); <b>Read</b> (A, B, C) ;  {--*--*-- Traitement & Sorties--*--*--} <b>Writeln</b> ( 'Les 3 valeurs avec un ordre croissant : ' ) ; <b>if</b> (A <= B) <b>And</b> (B <= C) <b>then</b> <b>Write</b> (A, B:5, C:5) <b>else</b> <b>if</b> (A <= C) <b>And</b> (C <= B) <b>then</b> <b>Write</b> (A, C:5, B:5) <b>else</b> <b>if</b> (B <= A) <b>And</b> (A <= C) <b>then</b> <b>Write</b> (B, A:5, C:5) <b>else</b> <b>if</b> (B <= C) <b>And</b> (C <= A) <b>then</b> <b>Write</b> (B, C:5, A:5) <b>else</b> <b>if</b> (C <= A) <b>And</b> (A <= B) <b>then</b> <b>Write</b> (C, A:5, B:5) <b>else</b> <b>Write</b> (C, B:5, A:5); <b>End.</b>

**NB :** Il existe d'autres solutions qui permettent d'afficher trois valeurs numériques A, B et C avec ordre croissant.

**Exercice N°05 :**

Ecrire un programme Pascal intitulé **PARITE** qui saisit un nombre entier et détecte si ce nombre est pair ou impair.

**Corrigé de l'exercice N°05 :**

Algorithme	Programme PASCAL
<b>Algorithme</b> Partie ; <b>Variables</b> N : entier;  <b>Début</b> {--*--*-- Entrées --*--*--} Écrire ('Donner un entier : '); Lire (N);  {--*--*-- Traitement & Sorties--*--*--} <b>Si</b> N mod 2 = 0 <b>alors</b> Écrire (N, ' est pair') <b>Sinon</b> Écrire (N, ' est impair') ; <b>Fin-Si</b> ; <b>Fin.</b>	<b>Program</b> Parite ; <b>Var</b> N : Integer ;  <b>Begin</b> {--*--*-- Entrées --*--*--} <b>Writeln</b> ('Donner un entier : '); <b>Readln</b> (N);  {--*--*-- Traitement & Sorties--*--*--} <b>IF</b> N mod 2 = 0 <b>Then</b> <b>Writeln</b> (N, ' est pair') <b>Else</b> <b>Writeln</b> (N, ' est impair') ; <b>End.</b>

**Exercice N°06 :**

Ecrire un algorithme/programme Pascal pour chaque cas suivant :

- 1) Calculer la somme  $S = 1^2 + 3^2 + 5^2 + \dots + (2N + 1)^2$
- 2) Calculer le produit  $P = 1 * 2 * 3 * \dots * N$
- 3) Calculer la somme  $S = X + X^2 + X^3 + \dots + X^N$
- 4) Calculer la somme  $S = x + \frac{x^3}{2} + \frac{x^5}{4!} + \frac{x^7}{6!} + \dots + (N\text{ème terme})$

**Corrigé de l'exercice N°06 :**

- 1) Calculer la somme  $S = 1^2 + 3^2 + 5^2 + \dots + (2 * N + 1)^2$

Algorithme	Programme PASCAL
<b>Algorithme</b> exo2_1;  <b>variables</b> S, N, i: entier;  <b>Début</b>	<b>Program</b> exo2_1;  <b>Var</b> S, N, i: integer;  <b>Begin</b>

<pre> {--*-*- Entrées --*-*-} Ecrire('Donner la valeur de N : '); Lire(N); {--*-*- Traitement --*-*-} S ← 0;    Pour i ← 0 à N faire     S ← S + sqr(2*i+1);   Fin-pour {--*-*- Sortie --*-*-} Ecrire ('S =', S); <b>Fin.</b> </pre>	<pre> {--*-*- Entrées --*-*-} Writeln('Donner la valeur de N : '); Readln(N); {--*-*- Traitement --*-*-} S:=0;    For i:=0 to N do     S:= S + sqr(2*i+1);   {--*-*- Sortie --*-*-}   Write('S =', S); <b>End.</b> </pre>
--	---

2) Calculer le produit  $P = 1 * 2 * 3 * \dots * N$

Algorithme	Programme PASCAL
<pre> <b>Algorithme</b> exo2_2; <b>Variables</b>   N, i, P: entier ; <b>Début</b>   {--*-*- Entrées --*-*-}   Ecrire('Donner la valeur de N : ');   Lire(N);   {--*-*- Traitement --*-*-}   P ← 1;   Pour i ← 1 à N faire     P ← P* i;   Fin-pour;   {--*-*- Sortie --*-*-}   Ecrire('P =', P); <b>Fin.</b> </pre>	<pre> <b>Program</b> exo2_2; <b>Var</b>   N, i, P: integer ; <b>Begin</b>   {--*-*- Entrées --*-*-}   Writeln('Donner la valeur de N : ');   Readln(N);   {--*-*- Traitement --*-*-}   P:=1;   for i:=1 to N do     P:= P* i;   {--*-*- Sortie --*-*-}   Write('P =', P); <b>End.</b> </pre>

3) Calculer la somme  $S = X + X^2 + X^3 + \dots + X^N$

Algorithme	Programme PASCAL
<p><b>Algorithme</b> exo2_3;</p> <p><b>Variables</b></p> <p>N, i: entier;</p> <p>X, S, P: réel;</p> <p><b>Début</b></p> <p>{-*-*- Entrées -*-*-}</p> <p><b>Ecrire</b>('Donner la valeur de N : ');</p> <p><b>Lire</b>(N);</p> <p><b>Ecrire</b>('Donner la valeur de X : ');</p> <p><b>Lire</b>(X);</p> <p>{-*-*- Traitement -*-*-}</p> <p>S ← 0;</p> <p>P ← X;</p> <p><b>Pour</b> i ← 1 à N <b>faire</b></p> <p>    S ← S+P;</p> <p>    P ← P*X;</p> <p><b>Fin-pour</b>;</p> <p>{-*-*- Sortie -*-*-}</p> <p><b>Ecrire</b>(' S = ', S:0:3);</p> <p><b>Fin.</b></p>	<p><b>Program</b> exo2_3;</p> <p><b>Var</b></p> <p>N, i: integer;</p> <p>X, S, P: real;</p> <p><b>Begin</b></p> <p>{-*-*- Entrées -*-*-}</p> <p><b>Write</b>('Donner la valeur de N : ');</p> <p><b>Read</b>(N);</p> <p><b>Write</b>('Donner la valeur de X : ');</p> <p><b>Read</b>(X);</p> <p>{-*-*- Traitement -*-*-}</p> <p>S:=0;</p> <p>P:=X;</p> <p><b>for</b> i:=1 <b>to</b> N <b>do</b></p> <p>    <b>Begin</b></p> <p>        S:= S+P;</p> <p>        P:= P*X;</p> <p>    <b>End</b>;</p> <p>{-*-*- Sortie -*-*-}</p> <p><b>Write</b>(' S = ', S:0:3);</p> <p><b>End.</b></p>

4) Calculer la somme  $S = X + \frac{X^3}{2!} + \frac{X^5}{4!} + \frac{X^7}{6!} + \dots$  ( $N^{\text{ème}}$  terme)

Algorithme	Programme PASCAL
<p><b>Algorithme</b> exo2_4;</p> <p><b>Variables</b></p> <p>N, i, F : entier;</p> <p>X, P, S : réel;</p> <p><b>Début</b></p> <p>{-*-*- Entrées -*-*-}</p> <p>Ecrire('Donner la valeur de N : ');</p> <p>Lire(N);</p> <p>Ecrire('Donner la valeur de X : ');</p> <p>Lire(X);</p> <p>{-*-*- Traitement -*-*-}</p> <p>S ← 0;</p> <p>P ← X;</p> <p>F ← 1;</p> <p><b>Pour</b> i:=0 à (N-1) <b>faire</b></p> <p style="padding-left: 20px;">S ← S + P/F;</p> <p style="padding-left: 20px;">P ← P*X*X;</p> <p style="padding-left: 20px;">F ← F*(2*i+1)* (2*i+2);</p> <p><b>Fin-Pour;</b></p> <p>{-*-*- Sortie -*-*-}</p> <p>Ecrire(' S = ', S:0:3);</p> <p><b>Fin.</b></p>	<p><b>Program</b> exo2_4;</p> <p><b>Var</b></p> <p>N, i, F : integer;</p> <p>X, P, S : real;</p> <p><b>Begin</b></p> <p>{-*-*- Entrées -*-*-}</p> <p>Write('Donner la valeur de N : ');</p> <p>Read(N);</p> <p>Write('Donner la valeur de X : ');</p> <p>Read(X);</p> <p>{-*-*- Traitement -*-*-}</p> <p>S:=0;</p> <p>P:=X;</p> <p>F:=1;</p> <p><b>For</b> i:=0 to (N-1) <b>do</b></p> <p style="padding-left: 20px;"><b>Begin</b></p> <p style="padding-left: 40px;">S:= S + P/F;</p> <p style="padding-left: 40px;">P:= P * sqr(X);</p> <p style="padding-left: 40px;">F:=F*(2*i+1)* (2*i+2);</p> <p style="padding-left: 20px;"><b>End;</b></p> <p>{-*-*- Sortie -*-*-}</p> <p>Write(' S = ', S:0:3);</p> <p><b>End.</b></p>

## II.10. Exercices supplémentaires

### Exercice 01 :

Réaliser les conversions suivantes :

$$2021 = (?)_2$$

$$(753)_8 = (?)_2$$

$$(10110110001)_2 = (?)_{10}$$

$$(101110011100011)_2 = (?)_8 = (?)_{16}$$

$$(753)_8 = (?)_{10}$$

$$(AB0793)_{16} = (?)_8$$

### Exercice 02 :

a) Traduire les expressions suivantes en langage Pascal :  $y = x^2 + \sqrt{\frac{|2x| + \sqrt{x}}{2e^x}}$  ;  $z = e^{\sqrt{5x+|-3x|}}$

b) Définir les opérateurs **DIV** et **MOD** en donnant deux exemples numériques pour chacun.

### Exercice 03 :

Soit a, b, c, d, x, y des variables réelles, tel-que : a=1, b=2, c=3, d=6

Évaluer les expressions suivantes en indiquant l'ordre d'évaluation :

$$(a + b) + (c + a * (d / 3)) + 6 / c + 2 * a$$

$$(a + b) < (c + a * (d / 3)) + 6 / c + 2 * a$$

$$(a > b) \text{ And Not } (c + a > d / 3) \text{ OR } (6 \text{ Mod } c = 2 \text{ Div } c)$$

### Exercice 04 :

En PASCAL, indiquer, parmi cette liste de mots, les identificateurs valides et non-valides :

12K, a, x1, k12, prix unitaire, qte-stock, sinon, while, begin, hateur, largeur

### Exercice 05 :

Ecrire un algorithme puis la traduction en Pascal d'un programme **Surface\_Rectangle**, qui calcule la surface d'un rectangle de dimensions données et affiche le résultat sous la forme suivante : "La surface du rectangle dont la longueur mesure .... m et la largeur mesure .... m, a une surface égale à .... mètres carrés".

### Exercice 06 :

Ecrire un algorithme puis la traduction en Pascal d'un programme **Trapeze**, qui lit les dimensions d'un trapèze et affiche sa surface.

**Exercice 07 :**

Ecrire un algorithme puis la traduction en Pascal d'un programme qui lit une **température** en degrés Celsius et affiche son équivalent en Fahrenheit.

**Exercice 08 :**

Exécuter les séquences d'instructions suivantes manuellement et donner les valeurs finales des variables A, B, C et celles de X, Y, Z.

a)  $A \leftarrow 5$ ;  $B \leftarrow 3$ ;  $C \leftarrow B+A$ ;  $A \leftarrow 2$ ;  $B \leftarrow B+4$ ;  $C \leftarrow B-2$

b)  $X \leftarrow -5$ ;  $Y \leftarrow 2*X$ ;  $X \leftarrow X+1$ ;  $Y \leftarrow \text{sqr}(-X-Y)$ ;  $Z \leftarrow \text{sqr}(-X+Y)$ ;  $X \leftarrow -(X+3*Y)+2$

Ecrire les algorithmes correspondants puis les programmes en Pascal correspondants et les exécuter.

**Exercice 09 :**

Ecrire un algorithme permettant d'effectuer une permutation circulaire de trois nombre entiers a, b et c.

Exemple : a=10, b=20 et c=30

Après permutation : a=30, b=10 et c=20

**Exercice 10 :**

Ecrire un programme permettant de lire la valeur de la température de l'eau et d'afficher son état :

« Glace » Si la température  $\leq 0$ ,

« Liquide » Si  $0 < \text{la température} < 100$ ,

« Vapeur » Si la température  $\geq 100$ .

**Exercice 11 :**

Soit un service d'impression qui établit le prix d'impression d'une page selon le nombre de pages (nb\_pages) :

a- Si nb\_pages est inférieure ou égale à 10 : 5 D.A.

b- Si nb\_pages est entre 11 et 20 : 4.5 D.A.

c- Si nb\_pages est entre 21 et 60 : 3 D.A.

d- Si nb\_pages est supérieure à 60 : 2.5 D.A.

**Exercice 12 :**

Ecrire un programme qui permet de résoudre l'équation du second degré  $ax^2 + bx + c = 0$

**Exercice 13 :**

On demande d'écrire l'algorithme d'une fiche de paie journalière d'un ouvrier rémunéré à la tâche. Pour cela, on donne :

- La valeur de cette rémunération par pièces réalisées VP,
- Le salaire brut (SB) est calculé selon le nombre de pièces correctes réalisées pendant la journée (NPC) comme suit :

Si  $NPC \leq 100$ , l'ouvrier touche  $NPC * VP$

Si  $NPC > 100$ , l'ouvrier touche  $150 * VP$

- On enlève à la fin 10% du salaire pour les charges sociales (CS).

Calculer et afficher le salaire journalier brut (SB), les charges sociales (CS) et salaire journalier net (SN).

NB : Salaire brut=salaire totale ; Salaire net=salaire sans les charges sociales.

**Exercice 14 :**

Ecrire un programme Pascal qui calcule et affiche la **somme** et le **produit**, des 20 premiers entiers (de 1 à 20).

**Exercice 15 :**

Ecrire un programme Pascal faisant calculer et afficher le **factoriel** d'un entier naturel N donné. Sachant que (pour  $N > 0$ ) :  $N! = N \times (N-1) \times (N-2) \times \dots \times 3 \times 2 \times 1$ .

**Exercice 16 :**

Ecrire un algorithme permettant de calculer la somme de tous les nombres impairs entre deux valeurs N et M.

**Exercice 17 :**

Ecrire un algorithme permettant de tester si un nombre N est premier en utilisant une boucle **Tant que** puis une boucle **Répéter**.

**Exercice 18 :**

Ecrire un programme Pascal qui permet de calculer la somme S suivante : (avec X un nombre réel donné)

$$S = \frac{X^2}{2} - \frac{X^4}{4} + \frac{X^6}{6} - \frac{X^8}{8} + \frac{X^{10}}{10} - \frac{X^{12}}{12} + \dots$$



**Exercice 19 :**

Ecrire un programme Pascal qui permet d'afficher tous les multiples de 7 positifs strictement et inférieur à 100. Le programme doit aussi calculer et afficher la somme, le produit et la moyenne de ces multiples de 7.

On veut aussi avoir sur l'écran à l'exécution du programme, les affichages sous la forme suivante :

Les multiples de 7 inférieurs à 100 sont : 7 14 21 .....
La somme de ces nombres est : .....
Le produit de ces nombres est : .....
La moyenne de ces nombres est : .....

A decorative scroll-like frame with a black outline. The frame is horizontal and has a rounded right side. On the left side, there is a vertical strip that looks like a scroll's edge, with a small loop at the top. On the top right corner, there is a small loop. The word "Bibliographie" is centered within the frame in a black serif font.

# Bibliographie

## **Bibliographie**

- [1] John Paul Mueller et Luca Massaron, Les algorithmes pour les Nuls grand format, 2017.
- [2] Charles E. Leiserson, Clifford Stein et Thomas H. Cormen, Algorithmique: cours avec 957 exercices et 158 problèmes, 2017.
- [3] Thomas H. Cormen, Algorithmes: Notions de base, 2013.
- [4] Ouzeggane R. Cours informatique 1, Université de Béjaia.