الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Faculté des Sciences Exactes
Département d'Informatique

كلية العلوم الدقيقة
قسم الإعلام الآلي

Support de cours du module **Algorithmique et Structures de données** I

1ère Année MI

Dr Ammar Boucherit

Année Universitaire : 2020/2021

Sommaire

Chapitre 00 : Introduction générale 1
Note préliminaire 1
1. Contexte scientifique 1
1.1. Qu'est ce que l'Informatique ? 1
1.2. Qu'est ce qu'une Information ? 2
1.3. Les aspects de l'Informatique 2
1.4. Architecture Matérielle d'un Ordinateur 3
Chapitre 01 : Initiation à l'Algorithmiques 6
1. Qu'est-ce qu'un algorithme? 6
2. Squelette d'un algorithme 7
3. Langage Algorithmique 8
4. Organigramme 8
5. Notion d'objet 10
5.1. Les variables 10
5.1.1. Types fondamentaux (prédéfinis) 11
5.1.2. Les opérations de base 11
5.1.3. Déclaration des variables 13
5.2. Les constantes 13
5.2.1. Déclaration des constantes 13
6. Les commentaires 14
Chapitre 02 : Expressions et Instructions 15
1. Les expressions 15
2. Les instructions de base 16
2.1. L'affectation 16
2.2. La lecture 17
2.3 L'écriture 17
3. Les structures composées 18
3.1 Les structures alternatives ou conditionnelles 18
3.2 Les structures répétitives (itératives) 21
3.2.1 Boucle Pour 21
3.2.2 Boucle Tant que 23
3.2.3 Boucle Répéter 25
Chapitre 03 : Les Tableaux 27
1. Introduction 27
2. Les Tableaux à une dimension 27
2.1. Déclaration 27
2.2 Lecture (Remplissage) 28
2.3 L'écriture (Affichage) 28
3. Les Tableaux à deux dimensions 29

3.1. Déclaration 29
3.2 Lecture (Remplissage) 30
3.3 L'écriture (Affichage) 30
Chapitre 04 : Les chaînes de caractères 33
Motivation 33
1. Les caractères 33
1.1. Qu'est-ce que le code ASCII ? 33
1.2. Opérateurs sur le type caractère 35
2. Les chaînes de caractères 35
2.1. Manipulation des chaînes de caractères. 36
1. Affectation 36
2. Lecture et écriture 36
2.2. Opérations et fonctions prédéfinies 36
Chapitre 05 : *Les types définis par l'utilisateur* 38
Motivation 38
1. Notion de type 38
2. Les types définis par l'utilisateur 39
2.1. Les enregistrements 39
2.1.1. Déclaration 39
2.1.2. Manipulation 40
2.2. Les types énumérés 42
2.2.1. Déclaration 42
2.2.2. Manipulation 42
Chapitre 06 : *Les sous-programmes* 43
Motivation 43
1. Les types des sous-programmes 43
2. Syntaxe d'un sous-programme 44
2.1. Syntaxe d'une procédure 44
2.2. Syntaxe d'une fonction 44
3. Appel des sous-programmes 45
4. Variables globales et variables locales 46
5. Modes de passage de paramètres 48
5.1. Passage par valeur 48
5.2. Passage par variable 48
Références : 51

Chapitre 00
Introduction générale

Note préliminaire

Le contenu du cours du module Algorithmique et structure de données est principalement destiné aux étudiants de 1ère année MI du département des Mathématiques de la faculté des sciences exactes à l'université d'El-Oued. Il a été rédigé avec grand soin en tenant compte d'une approche pédagogique simple soutenue par des exemples pour assurer le maximum de clarté sans négliger la précision et la rigueur nécessaires.

Par conséquent, nous allons présenter dans cette première version de ce support de cours une introduction complète à l'étude des algorithmes informatiques. De nombreux algorithmes y sont présentés et étudiés en détail, de façon à rendre leur conception et leur analyse accessibles aux étudiants en 1ère années.

Autrement dit, l'objectif principal de ce support de cours est d'exposer en douceur les concepts de base de l'algorithmique pour aider l'étudiant à apprendre et à écrire des algorithmes.

1. Contexte scientifique

1.1. Qu'est ce que l'Informatique ?

Le mot ***Informatique*** est un nouveau terme proposé par Philippe Dreyfus en 1962 pour désigner le traitement automatique des informations. En effet, il est construit sur la contraction de des deux mots comme suit :

Informatique = Information + Automatique

Ce néologisme a été ensuite accepté par l'Académie française en 1966, et l'informatique est devenue officiellement "la science du traitement automatique de l'information", où l'information est considérée comme un concept des sciences de la communication et aussi un support bien capable de produire des connaissances.

Le mot français **Informatique** n'a pas un seul véritable mot équivalent en anglais, car aux Etats-Unis on parle de **Computing** Science (science du calcul) alors que **Informatics** est admis par les Britanniques [1].

Définition 1.1 : Informatique

L'informatique est la science du traitement automatique de l'information. Autrement dit, c'est l'ensemble des théories, méthodes et techniques utilisées pour le traitement automatique de **l'information** à l'aide des machines électroniques (**Ordinateurs**).

D'après la définition de l'Informatique, on trouve deux nouveaux mots qui sont : **Information** et **Ordinateur**.

1.2. Qu'est ce qu'une Information ?

Afin de répondre à cette question, on propose d'évoquer et tenter de clarifier les notions de donnée et information. En posant la question suivante :
Quelle est la différence entre une donnée et une information ?

- Selon wikipedia : « Une donnée est une description élémentaire d'une réalité. C'est par exemple une observation ou une mesure ».
- Le terme "donnée" est un terme général qui représente une notion abstraite, où une donnée peut être un paramètre, une valeur, un nom, une image, un résultat direct d'une mesure, ...etc.
- Une donnée n'a aucune signification en soi, mais quand elle est placée dans un contexte, une (ou plusieurs) donnée(s) devient (nent) une information. Autrement dit, les données permettent de produire des informations.

Exemples :

1. Dans une enquête par exemple, les enquêteurs partent recueillir des données sur le terrain et le traitement de ces données fournies des informations.
2. Quelqu'un trouve un mot écrit sur un papier, pour lui ce mot n'a aucun sens (donnée), mais si on lui dit que ce mot représente le nom du Major de promo de la 1ère année MI, ce mot devient une information.

En conclusion, on peut dire qu'une **information est une donnée** à laquelle on attribue un sens et/ou une interprétation.

1.3. Les aspects de l'Informatique ?

L'informatique traite deux aspects complémentaires : les programmes immatériels (**logiciel, software**) qui décrivent un traitement à réaliser et les machines (**matériel, hardware**) qui exécutent ce traitement.

Définition 1.2 : Logiciel

Un logiciel ou une application est un ensemble structuré d'instructions (et des programmes), qui permet à un ordinateur ou à un système informatique de traiter des informations, d'assurer une tâche ou une fonction particulière.

Définition 1.3 : Matériel

Le matériel informatique est l'ensemble des éléments physiques (microprocesseur, mémoire, écran, clavier, disques durs...) utilisés pour le traitement automatique des informations en utilisant des logiciels.

Définition 1.4 : Ordinateur

Le mot ordinateur est un terme générique qui a été créé en 1955 à la demande d'IBM, pour désigner un appareil ou une machine électronique programmable servant au traitement de l'information selon des séquences d'instructions (les programmes) qui constituent le logiciel.

Cette définition est un peu classique, car nous pouvons effectuer aujour'hui une variété des tâches que avec cet appareil telles que l'écriture, le dessin, la navigation du web et la communication avec d'autres personnes dans le monde entier ....etc.

L'intérêt d'un ordinateur revient principalement de sa capacité à manipuler **rapidement** et **sans erreur** un grand nombre d'informations, **mémoriser** des quantités numériques ou alphabétiques, rechercher, comparer ou classer des informations mémorisées.

1.4. Architecture Matérielle d'un Ordinateur

L'architecture matérielle d'un ordinateur repose sur le modèle de Von Neumann qui distingue classiquement 4 parties (Figure 1.1) :

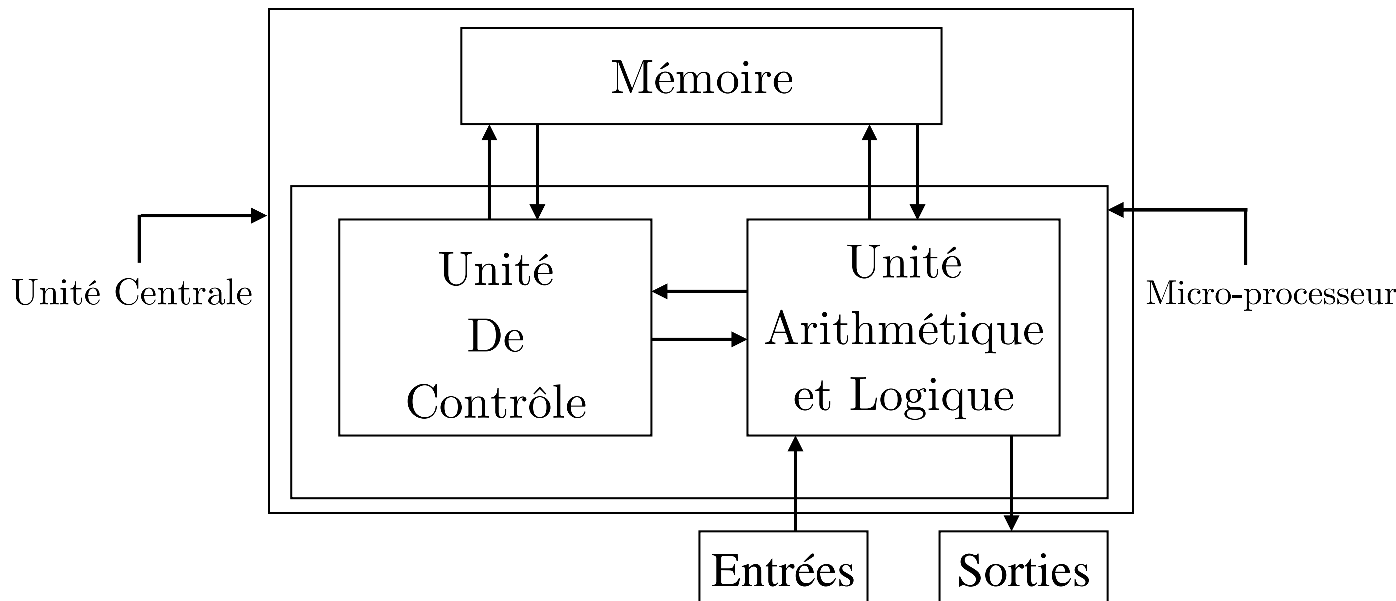


Figure 1.1 Architecture de Von Neumann

1. L'unité arithmétique et logique (unité de traitement), qui effectue les opérations de base.
2. L'unité de contrôle : séquence les opérations.
3. La mémoire : contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire vive volatile (programmes et données en cours de fonctionnement) et mémoire de masse permanente (programmes et données de base de la machine).
4. Les entrées-sorties : sont des dispositifs permettant de communiquer avec le monde extérieur (écran, clavier, souris...).

Les 2 premières parties sont souvent rassemblées au sein d'une même structure qui s'appelle « Le **micro-processeur** » et le micro-processeur avec la mémoire sont appelés l'**unité centrale** de l'ordinateur.

Pour conclure, l'ordinateur est une machine capable de :

- recevoir des données (de l'utilisateur le plus souvent) en mémoire via des périphériques d'entrée,
- traiter les données grâce à l'unité de traitement
- fournir les données (résultats) de sa mémoire vers des périphériques de sortie (écran par exemple).

La figure suivante illustre ces fonctions

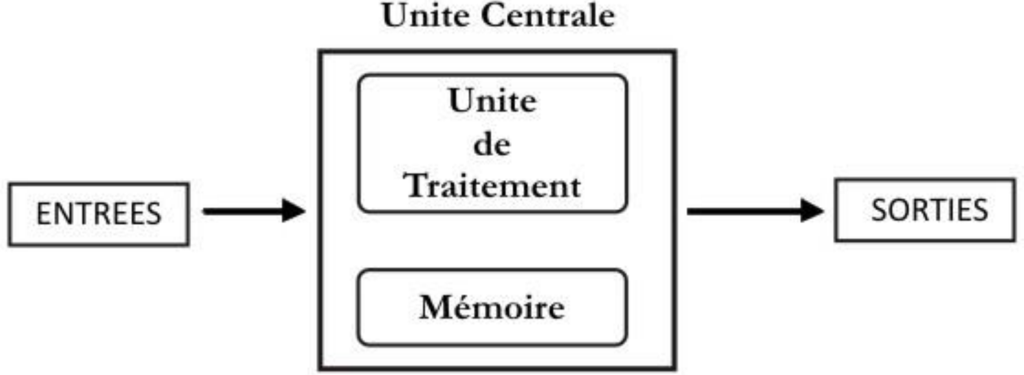


Figure 1.2 Structure simple d'un ordinateur

- Un ordinateur manipule exclusivement des informations **binaires**.
- Pour donner des ordres à l'ordinateur, il est nécessaire de pouvoir communiquer avec lui. Cette communication passe par un **langage de programmation**, dans lequel est écrit les programmes.

Première Partie

Dans cette partie, on présente les fondamentaux qui vont guider l'étudiant pour concevoir et analyser des algorithmes. Nous essayons de maintenir la simplicité des explications, en partant toujours des connaissances simples acquis par l'étudiant.

.

Chapitre 01
Initiation à l'Algorithmiques

En principe, la notion d'algorithmes est très générale et ne se limite pas au cadre de l'informatique. En effet, nous utilisons dans notre vie courante, des méthodes algorithmiques. Les étapes à suivre pour préparer du café, ou bien le manuel pour utiliser un appareil sont des algorithmes.

1. Qu'est-ce qu'un Algorithme ?

On appelle un algorithme l'ensemble d'opérations et de règles définissant « ce qu'il faut faire » et « dans quel ordre » pour résoudre un problème (ou une classe de problème).

Définition 1.1 : Algorithme

Un algorithme est une suite finie d'opérations élémentaires (instructions), à appliquer dans un ordre déterminé, à un ensemble des données pour résoudre un problème donné [2].

Cette définition se base sur le principe de décomposition d'une tâche quelconque en un ensemble de tâches élémentaires exécutés en séquence suivant un enchaînement strict.

Exemple :

Préparer du café, déterminer le plus court chemin entre deux points, calculer la somme de plusieurs nombres, ... sont autant d'algorithmes pour lesquels une suite d'actions sont à effectuer une ou plusieurs fois afin d'obtenir un résultat.

Signalons ici, que l'algorithmique est un terme d'origine arabe, plutôt vient du nom du mathématicien persan Abu Djafar Mohammed Ibn Musa surnommé El-Khowarizmi qui est l'un des fondateurs de l'arithmétique moderne [3].

Remarques :

1. Un algorithme doit être ***lisible***.
2. L'intérêt d'un algorithme est la possibilité d'***être codé dans un langage informatique*** afin qu'un ordinateur puisse l'exécuter rapidement et efficacement.

3. Les trois phases d'un algorithme sont, dans l'ordre :
   (a) l'entrée des données, (b) le traitement des données, (c) la sortie des résultats

Exemples :

L'algorithme qui calcul et affiche la surface et le périmètre d'un cercle de rayon R, peut être d'écrit comme suit :

- Introduire le rayon R
- Calculer la surface S suivant la loi $S = \pi * R^2$
- Calculer le périmètre P suivant la loi $P = 2 * \pi * R$
- Afficher les valeurs de : S et P

Cet algorithme nécessite le rayon R et fournie la surface et le périmètre du cercle.

Définition 1.2 : Algorithmique

Selon le dictionnaire "l'internaute", l'algorithmique est la science des algorithmes, laquelle porte sur la manière de résoudre des problèmes en mettant en oeuvre des suites d'opérations élémentaires.

L'algorithmique s'intéresse à l'art de construire des algorithmes ainsi qu'à caractériser leur validité, leur robustesse, leur réutilisation, leur complexité ou leur efficacité.

2. Squelette d'un algorithme

Chaque algorithme est composé de trois parties indissociables comme suit :

```
Algorithme  Nom_Algorithme
   Partie de Déclaration
Début
 |  Partie d'initialisation     }
 |  Partie de traitement        }     Corps de l'algorithme
 |  Partie de sortie des résultats }
Fin
```

Dans la partie d'***initialisation*** on doit définir les valeurs initiales des objets manipulés dans l'algorithme. La partie de ***traitement*** regroupe l'ensemble des instructions nécessaire pour résoudre le problème. Les résultats de l'algorithme seront affichés en utilisant les instructions d'affichage dans la dernière partie.

Autrement dit, on peut dire que tout algorithme est caractérisé par :

- un début et une fin.
- un ensemble d'instructions à exécuter.
- un ordre d'exécution de ces différentes instructions, déterminé par la logique

d'enchaînement et conditionné par les structures mises en œuvre.

En plus, La figure suivante illustre la position de l'algorithme dans le cycle de développement du programme.

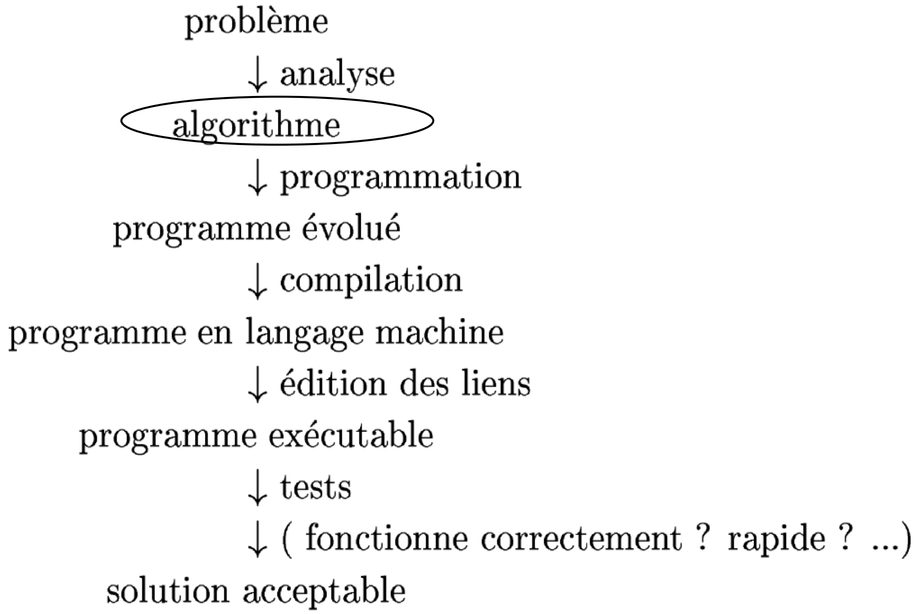


Figure 2.1 : cycle de développement d'un programme

3. Langage Algorithmique

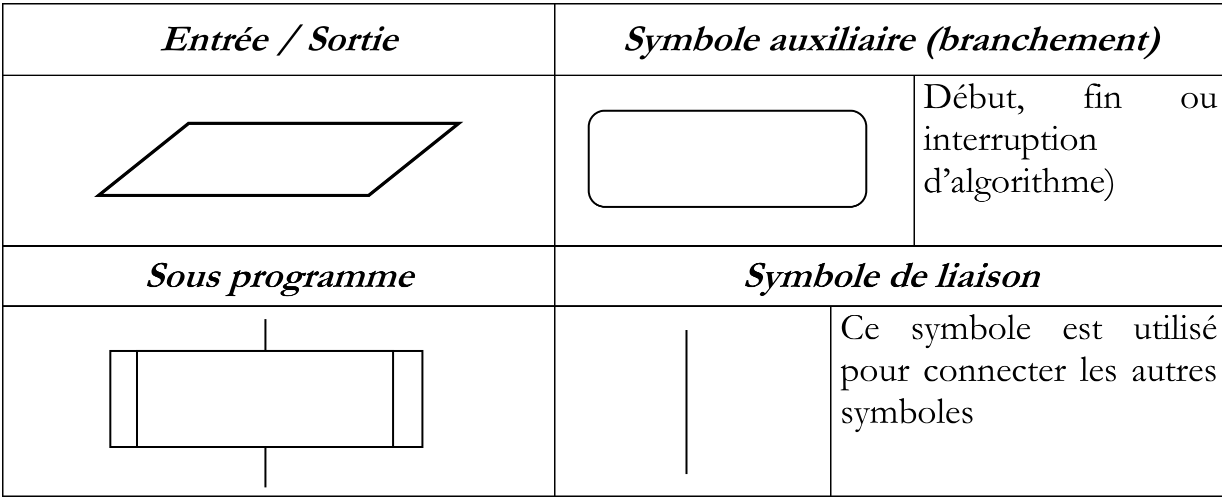
Un algorithme doit être écrit suivant la syntaxe d'un langage proche du langage naturel appelé **langage algorithmique** ou **pseudo-langage**. Ce langage utilise un ensemble de ***mots clés*** et de ***structures*** permettant de décrire de manière complète et claire, les objets manipulés par l'algorithme ainsi que l'ensemble des ***instructions*** à exécuter sur ces objets pour résoudre un problème [3]. Un tel langage présente un avantage réel, et les algorithmes peuvent être transcrit ensuite dans un langage de programmation structuré et lisible.

Il ne faut donc pas confondre algorithme et programme. Un programme est la traduction d'un algorithme en un langage compatible ou interprétable par un ordinateur. Ex. langage de programmation.

4. Organigramme

L'organigramme est l'une des modes utilisées pour représenter graphiquement l'algorithme. Pour cela, on utilise des symboles normalisés dont les principaux sont les suivants :

Symbole de traitement (général)	***Symbole de test (branchement)***



Entrée / Sortie	*Symbole auxiliaire (branchement)*	
		Début, fin ou interruption d'algorithme)
Sous programme	*Symbole de liaison*	
		Ce symbole est utilisé pour connecter les autres symboles

Figure 2.2 : Symboles utilisés pour déssiner un organigramme

Exemple :

Voici l'organigramme 01 d'un algorithme permettant de calculer la somme de deux nombres entiers, et l'organigramme 02 d'un algorithme permettant de trouver et d'afficher la plus grande valeur parmi trois valeurs données par l'utilisateur.

Début
Variables
A, B, C : Entier
Taper les notes
Lecture des notes
C = A + B
Affichage de C
Fin

Organigramme 01

Début
Variables
A, B, C : Entier
Lecture des valeurs
A > B
Oui
Non
A > C
Oui
Non
B > C
Non
Oui
Afficher A
Afficher C
Afficher B
Fin

Organigramme 02

5. Notion d’objet

Tous les objets qui deviennent partie intégrante de l'exécution d’un algorithme doivent être déclarés avant leurs utilisation; ceci constitue la partie déclaration d’un algorithme. Un objet peut être décrit par un nom (identificateur), un type, une valeur et une nature [3].

L’**identificateur** est utilisé pour donner un nom à un objet et le distinguer aux autres objets dans l’algorithme [4].

Syntaxe :

On appelle lettre un caractère de ’a’..’z’ ou ’A’..’Z’ ou ’_’.

On appelle digit un caractère de ’0’..’9’.

Un identificateur est une suite de lettres et/ou de digit accolés, commençant par une lettre. Par exemple : Nom, Code_Etudiant, Pi, X sont des noms d’objets corrects.

Remarques

- Un identificateur doit être significatif (représentatif).
- En l’algorithmique, il n’y a pas de différence entre minuscules et majuscules.
- Le caractère « _ » peut être utilisé pour construire des noms d’identificateurs composés de plusieurs mots.
- Un identificateur doit être différent des mots clés (algorithme, début, fin, Afficher….)

Le **type** d'un objet déclaré dans un algorithme détermine l’intervalle ou l'ensemble des valeurs permises pour cet objet ainsi que les opérations qui lui sont autorisées tout au long de l’algorithme.
La **valeur** d’un objet est celle lui a été assignée pour représenter la grandeur que contient cet objet.

La **nature** d’un objet signifie sa caractéristique (par exemple, constante ou variable). Un objet est de nature ***variable***, si sa valeur peut changer pendant l’exécution des actions de l’algorithme. Comme il est de nature ***constante*** si sa valeur est invariable.

5.1. Les variables

Lors de la préparation d’un algorithme, on aura besoin de stocker les valeurs des données manipulées ainsi que les résultats obtenus (calculés). Pour cela, on utilise des objets de nature variables, c-à-d, un objet d’un certain type et sa valeur peut varier lors de l’exécution d’un algorithme.

Définition 5.1 : variable

Une variable est un objet informatique qui associe un nom à une valeur qui peut varier au cours du temps de l’exécution d’un algorithme.

En effet, la variable est représentée dans la mémoire par un nombre de mots mémoire correspond à leur type (voir Figure 2.3).

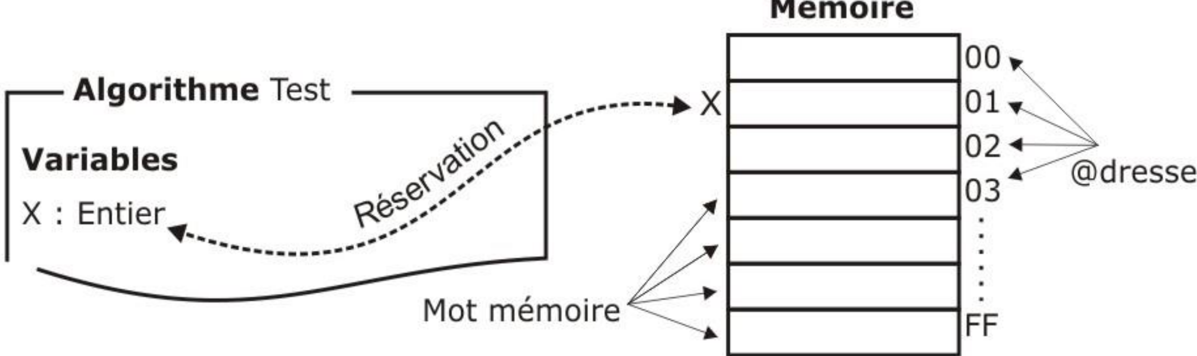


Figure 2.3 : Représentation d'une variable dans la mémoire

5.1.1. Types fondamentaux (prédéfinis)

Au début de ce cours, on ne s'intéresse que de quatre types qui sont fréquemment utilisés.

1. **Type entier :** Ce type représente le domaine des nombres entiers.
2. **Type réel :** Ce type représente le domaine des nombres réels.
3. **Type logique (booléenne):** Ce type représente le domaine logique qui contient deux valeurs logiques (vrai et faux).
4. **Type caractère :** Ce type représente le domaine des caractères qui contient les lettre alphabétiques minuscules, les caractères numériques, les caractères spéciaux (., ?, !, <, >, =, *, +, ...etc), et le caractère espace (ou blanc).

   Le code ASCII est le code le plus répandu pour la représentation des caractères où les caractères sont ordonnés et contigus.

Remarque :

- Une variable de type réel peut avoir une valeur de type entier, car le type entier est inclus dans le type réel, mais l'inverse est totalement faux.
- Nous découvrirons d'autres types de variables dans la suite du cours.

5.1.2. Les opérations de base

Définition 8.1 : Opérateur et opérande

- Un OPERATEUR est un outil qui permet d'agir sur une variable ou d'effectuer des calculs.
- Un OPERANDE peut être : une constante, une variable, un appel de fonction qui sera utilisé par un opérateur.

Exemple :

Dans l'expression « 8+ y», «+» désigne l'opérateur ; «8» et «y» sont les opérandes.

5.1.2.1. Types d'opérateurs

Il existe plusieurs types d'opérateurs :

- ***Les opérateurs arithmétiques*** qui permettent d'effectuer des opérations arithmétiques entre des opérandes numériques :

  Les opérateurs élémentaires sont : «+», «-», « ×», «/», «[/]» (division entière) et l'opérateur de changement de signe «-» (qui est similaire de l'opérateur de soustraction).

- ***Les opérateurs de comparaison*** («<», «>», «≤», «≥», «=» et «≠») qui permettent de comparer deux opérandes et produisent une valeur booléenne, en s'appuyant sur des relations d'ordre :
  - Ordre naturel pour les entiers et les réels
  - Ordre lexicographique ASCII pour les caractères

- ***Les opérateurs logiques*** qui combinent des opérandes booléens pour former des expressions logiques plus complexes :
  - Unaire : «non» (négation)
  - Opérateurs binaires : «et» (conjonction), «ou» (disjonction), «ou_exclusif»

Problème

Evaluer l'expression suivante : x + 5 * y -7

il y aura plusieurs possibilités comme suit :

Possibilité 1	Possibilité 2	Possibilité 3
(x + 5) * (y-7)	x + (5 * y) - 7	x + (5 *(y - 7))

5.1.2.2. Priorité des opérateurs

Pour lever toute ambiguïté lors de l'évaluation d'une expression, il est nécessaire d'associer une priorité à chaque opérateur pour définir son ordre d'exécution. Comme il est possible de modifier cet ordre par l'utilisation des parenthèses.

- **Ordre de priorité décroissante des opérateurs arithmétiques**

Symbole	Opération
«-»	changement de signe
« ×», «/», «[/]»	Multiplication, division et division entière
«+», «-»	Addition et Soustraction

- **Ordre de priorité décroissante des opérateurs logiques**

Symbole	Opération
«non»	négation logique (fonction inverse)
« et»	pour réaliser la conjonction logique entre deux valeurs booléennes
«ou»	pour réaliser la disjonction logique entre deux valeurs booléennes
«Xou»	pour réaliser le ou exclusif entre deux valeurs booléennes

Remarque :

- Il est préférable d'utiliser les parenthèses () pour éviter toute erreur, car les opérations entre parenthèses seront évaluées d'abord.
- Les opérateurs de même priorité dans une expression sont évalués de gauche à droite.
- Les valeurs logiques sont aussi ordonnées et que faux < vrai.
- Les opérateurs de comparaison ont tous le même ordre.
- La priorité entre les opérateurs diffère de celle des langages de programmation.

5.1.3. Déclaration des variables

Un objet de nature variable suit la déclaration suivante :

Variables
identificateur : ***Type***

Où ***identificateur*** et ***Type*** représente respectivement le nom et le type de la variable.

Exemple :

Variables
X : ***Entier*** // déclaration d'une variable X de type Entier
Y, Z : réel // déclaration d'une variable Y et Z de type réel

5.2. Les constantes

Une constante est objet informatique similaire à une variable en nom, type et nature. Elle se diffère d'une variable seulement dans sa valeur qui ne varie pas au cours de l'exécution de l'algorithme.

Définition 5.2 : constante

Une constante est désignée par un identificateur et une valeur, qui sont fixés en début de l'algorithme.

5.2.1. Déclaration des constantes

Un objet de nature constante suit la déclaration suivante :

Constantes
Nom de la constante = ***Valeur de la constante***

Où ***Nom de la constante*** et ***Valeur de la constante*** représente respectivement le nom et la valeur de la constante.

Exemple :

Constante
X = 10 // déclaration d'une constante X de type Entier et sa valeur = 10
PI = 3.14159 // déclaration d'une constante Y et sa valeur = 3.14159

Remarques :

- Le langage algorithmique utilise des mots-clés, que le concepteur ne peut pas utiliser pour désigner des objets comme les mots ***algorithme***, ***constante*** et ***variable*** (nous découvrirons d'autres mots-clés dans la suite du cours).
- Si plusieurs variables sont de même type, on cite la suite des noms de variables séparés par une virgule.
- Ne pas confondre la variable et son nom. En effet, la variable possède une valeur (son contenu) et une adresse (emplacement dans la mémoire où est stockée la valeur). Le nom n'est qu'un identificateur de la variable, c'est-à-dire un constituant de cette variable.
- Pour se familiariser avec les langages de programmation, on peut mettre toujours un point virgule après chaque instruction même s'il n'est pas nécessaire dans le langage algorithmique.

6. Les commentaires

Vu qu'un programme est écrit pour un être humain, il est donc très important d'en faciliter la lecture, en utilisant, les commentaires. Ces dernières permettent d'éclaircir certains passages de codes. Ainsi, il est très souvent agréable d'avoir des commentaires pour comprendre ce que le développeur à voulu faire. Ils sont utilisés aussi pour que le développeur lui-même se rappellera son travail. Pratiquement, on commente par l'utilisation du ***soulignage*** ou par le double "***slash***", comme ceci : //

Exemple

<u>Voici un premier commentaire</u> // et en voici un second

Chapitre 02
Expressions et Instructions

Nous rappelons, dans ce chapitre, qu'est ce qu'une expression et présentons les différents types d'instructions.

1. Les expressions

On ne s'intéresse dans ce cours qu'aux expressions arithmétiques et logiques. Une expression est formée d'opérandes et d'opérateurs. L'évaluation d'une expression faisant intervenir plus d'un opérateur repose sur des règles de priorité entre opérateurs.

L'expression entre parenthèses est évaluée indépendamment des opérateurs placés à droite et à gauche des parenthèses.

Pour déterminer le type d'une expression, il faut vérifier la syntaxe de cette expression (c'est-à-dire sa façon d'écriture), la compatibilité des types des opérandes qui la composent, ainsi que la validité des ses opérateurs. C'est le type des opérandes qui définit le type de l'expression.

Remarque :

Des opérandes réels et entiers peuvent figurer dans une même expression. Le type de cette dernière est réel.

Exemple :

Déterminer le type de l'expression suivante : A – C + B * D / 2 sachant que :

A, B : réel; C, D : entier;

- La syntaxe de cette expression est correcte;
- La compatibilité des types est vérifiée : les opérandes réels et entiers peuvent être membre d'une même expression.

➢ Les opérateurs : +, - , * et / sont valides sur les types entier et réel.
Donc cette expression sera évaluée dans l'ordre suivant :

1. B * D (de type réel)
2. (B * D)/2 (de type réel)
3. A - C (de type réel)
4. (A - C)+((B * D) / 2) (de type réel)

2. Les instructions de base

Les instructions de base permettent le transfert d'informations entre objets, et la communication entre un algorithme et son utilisateur. On distingue trois genres des instructions élémentaires : l'affectation, la lecture et l'écriture.

2.1. L'affectation

C'est l'action par laquelle nous pouvons placer dans une variable X une valeur (par exemple, résultat de l'évaluation d'une expression E). Le type de la partie droite de l'instruction doit être compatible avec le type de la variable X.
L'affectation est faite selon la syntaxe suivante :

X ⟵ E ;

Cette partie doit être une variable (X)

Cette partie peut être : (E)
- une valeur
- une variable ou une constante
- une expression de même type que X
- appel à une fonction (à voir plus tard)

Exemple 1:

Variables X , A : entiers

A ⟵ 5
X ⟵ A + 10

Ce qui veut dire :

- Initialiser (attribuer) la valeur 5 à la variable A.

- Évaluer l'expression A + 10 et mettre le résultat dans la place mémoire réservée pour X, et donc la valeur de la variable X sera égale à 15.

Remarque :

L'instruction d'affectation permet de :
- forcer le contenu d'une variable (initialisation de la variable).
- d'écraser la valeur contenue précédemment dans la variable.

2.2. La lecture

Cette instruction permet d'introduire des valeurs par l'utilisateur via un périphérique d'entrée dans la mémoire de la machine. Elle est représentée comme suit :

Saisir($x_1, x_2, \dots, x_n$ **)**

L'action **Saisir(** $x_1, x_2, \dots, x_n$ **)** signifie :

Mettre dans les places mémoire nommées $x_1, x_2, \dots, x_n$, les n données présentées sur l'unité d'entrée de la machine. Au moment de l'exécution, la machine reste en attente jusqu'à ce que l'utilisateur tape au clavier les valeurs des x_i. Ensuite, il doit appuyer sur la touche Entrée '↵' pour indiquer la terminaison de l'opération de saisie.

2.3 L'écriture

L'écriture permet d'afficher un résultat ou un message à l'utilisateur. Elle est représentée comme suit :

Afficher($x_1, x_2, \dots, x_n$ **)** // pour afficher les valeurs des variables $x_1, x_2, \dots, x_n$.

Afficher('Bonjour'**)**; // pour afficher un message à l'utilisateur sur l'écran.
// dans ce cas, le message doit être écrit entre simples quottes.

Remarque :

- Il est possible de regrouper aussi un message et une variable dans la même instruction d'écriture comme suit : **Afficher (**'Résultat = ' **,** x_1 **)**

Exemple 1 :

Écrire un algorithme qui permet de lire la longueur et la largeur d'un rectangle puis calcule leur surface.

```
Algorithme Surface_rectangle
Variables
    Long, Larg , Surf: réels
Début
     //lecture des données
       Afficher ('Donner la longueur : ')
           Saisir (Long) ;
       Afficher ('Donner la largeur : ')
           Saisir (Larg) ;
     // Calcule de la surface
       Surf ←── Long × Larg
     //Affichage de résultat
       Afficher ('Surface : ', Surf)
Fin
```

Exemple 2 :

Écrire un algorithme qui permet de permuter les valeurs des deux variables X et Y données par l'utilisateur entre eux.

```
Algorithme Permutation
Variables
   X, Y , Z: entier
Début
    //lecture des données
      Afficher ('Donner la valeur de X : ')
         Saisir (X)
      Afficher ('Donner la valeur  Y : ')
         Saisir (Y) ;
   // Permutation des valeurs par l'utilisation d'une variable intermédiaire
      Z ←— X
      X ←— Y
      Y ←— Z
   //Affichage des données après permutation
      Afficher ('nouvelle valeur de X : ', X)
      Afficher ('nouvelle valeur de Y : ', Y)
Fin
```

3. Les structures composées

Généralement, un algorithme ne comprend pas que des instructions simples de manipulation des données qui va faire les exécuter les unes après les autres, mais il comprend aussi des instructions dites conditionnelles et/ou de répétitives (boucles).

3.1 Les structures alternatives ou conditionnelles

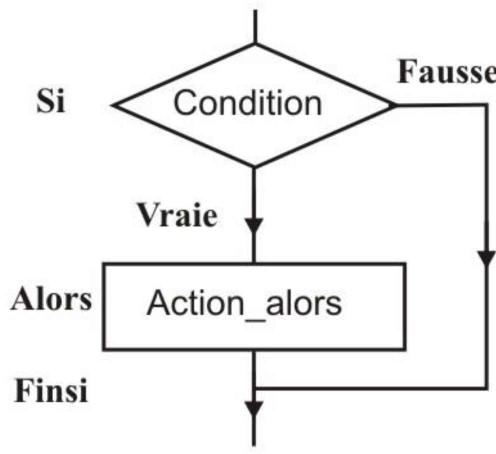
Le plus souvent, la résolution d'un problème implique la mise en place d'un test pour effectuer une tâche : si le test est positif, nous effectuons un certain traitement ; sinon, c'est-à-dire si le test est négatif, un autre traitement est effectué.

Cette situation peut être traduit à l'aide d'instructions conditionnelles comme suit:

- Pour exécuter un ensemble d'instructions suivant un test qui ne comporte que le cas où la condition est vérifiée, on utilise l'instruction suivante :

```
Si  Condition   alors
    Instruction 1
      ...
    Instruction n
Finsi
```

L'organigramme d'une telle structure est donné comme suit :

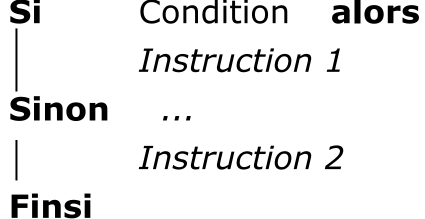


Exemple 3 :

Ecrire un algorithme qui permet de lire un nombre entier donné par l'utilisateur et d'afficher sa valeur absolue.

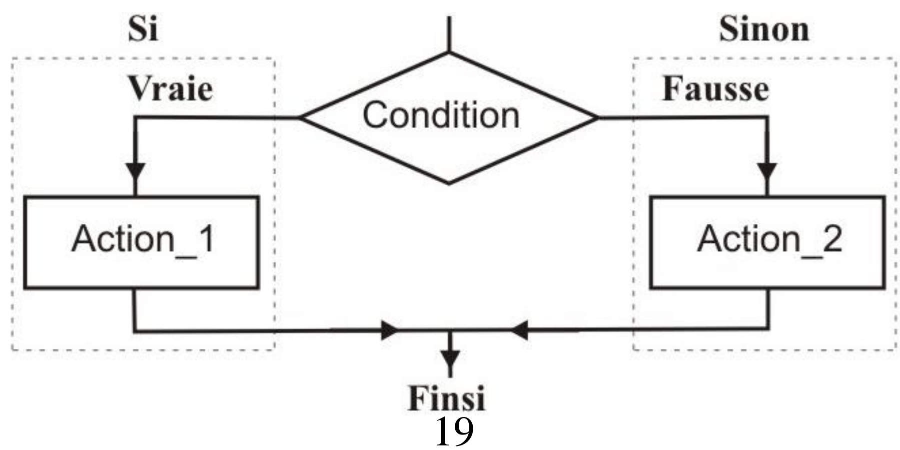
```
Algorithme Valeur_absolue
Variables      x, y : entier
Début
|      Afficher ('Donner une valeur x : ')       Saisir (x)
|       y ←—— x
|      Si (x < 0) alors
|      |  y ←—— - x
|      Finsi
|          Afficher ('La valeur absolue de : ', x , ' est : ', y)
Fin
```

➢ Pour exécuter un ensemble d'instructions dans le cas où une condition est vérifiée et d'autres instructions dans le cas où elle ne l'est pas, on utilise la structure suivante :



```
Si        Condition    alors
|         Instruction 1
Sinon     ...
|         Instruction 2
Finsi
```

L'organigramme correspondant à cette structure est donné comme suit :



Exemple 4 :

Ecrire un algorithme qui permet de vérifier si un point (a, b) est inclus dans un rectangle parallèle aux axes et spécifié par les coordonnées de son point en haut à gauche (x1, y1) et son point en bas à droite (x2, y2).

```
Algorithme Point_Inclus_rectangle
Variables    x1, y1, x2, y2, a, b : réels
Début
|     Afficher ('Donner les points : ')
|     Saisir (x1, y1, x2, y2, a, b) ;
|     Si ((a ≥ x1) et (a ≤ x2) et (b ≤ y1) et (b ≥ y2)) alors
|     |   Afficher ('Le point est inclus dans le rectangle')
|     Sinon
|     |   Afficher ('Le point est en dehors du rectangle')
|     Finsi
Fin
```

Exemple 5 :

Ecrire un algorithme qui permet de trouver la valeur maximale parmi trois valeurs entières distinctes données par l'utilisateur.

```
Algorithme Solution_Max1
Variables    x, y, z, max : entier
Début
|     Afficher ('Donner les valeurs x, y et z : ')
|     Saisir (x, y, z)
|     Si ((x > y) et (x > z)) alors
|     |   max ←—— x
|     Sinon
|     |   Si (y > z) alors
|     |   |  max ←—— y
|     |   Sinon
|     |   |  max ←—— z
|     |   Finsi
|     Finsi
|     Afficher ('x = ', x)
|     Afficher ('y = ', y)
|     Afficher ('z = ', z)
|     Afficher ('Max = ', max)
Fin
```

```
Algorithme Solution_Max2
Variables     x, y, z, max : entier
Début
      Afficher ('Donner les valeurs x, y et z : ')
          Saisir (x, y, z)
          max ← x
      Si (y > max) alors
          max ← y
      finsi
      Si (z > max) alors
          max ← z
      finsi
      Afficher ('x = ', x)
      Afficher ('y = ', y)
      Afficher ('z = ', z)
      Afficher ('Max = ', max)
Fin
```

Remarque :

- Comme l'avons déjà vu dans cet exemple, il peut exister plusieurs algorithmes pour le même problème.

- Il n'est pas toujours possible de remplacer une instruction (Si – sinon – finsi) par des instructions (Si – finsi).

3.2 Les structures répétitives (itératives)

Les instructions répétitives (boucles) permettent de répéter des instructions autant de fois que l'on souhaite. Plusieurs variantes sont possibles :

- répéter un bloc d'instructions un nombre de fois donné ;
- répéter un bloc d'instructions tant qu'une condition est vérifiée ;
- répéter un bloc d'instructions jusqu'à ce qu'une condition soit vérifiée.

3.2.1 Boucle Pour

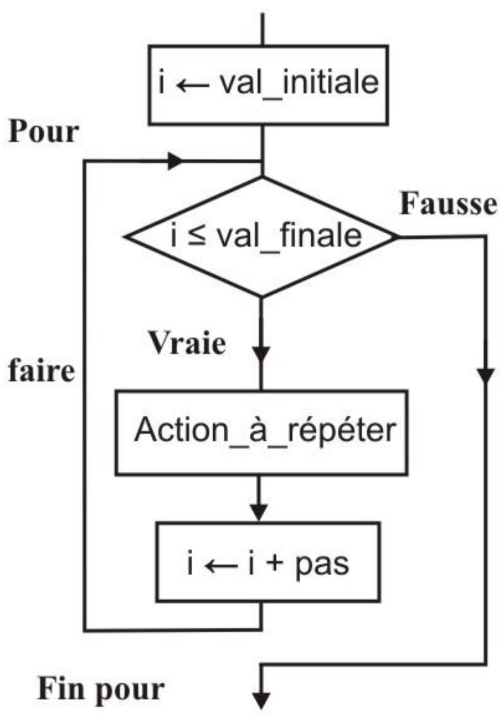
On n'utilise ce type de boucle que lorsqu'on connaît à l'avance le nombre de répétitions à effectuer. Cette boucle a la structure suivante :

```
Pour   I= valeur_initiale à valeur_finale   pas=val   faire

          Ensemble des instructions à répéter

Fin pour
```

L'organigramme de cette boucle est donné comme suit :



Remarques :

- La variable I est appelée (compteur de la boucle) et doit être déclarée de ***type Entier***.
- Le compteur est incrémenté (ou décrémenté) automatiquement de ±1 à chaque itération (tour) comme on peut préciser le pas pour incrémenter ou décrémenter le compteur par une autre valeur positive ou négative.
- Le pas peut être omis si sa valeur est de ±1.
- On peut utiliser la valeur du compteur pour faire des calculs à l'intérieur de la boucle, mais on ne doit pas en aucun cas modifier sa valeur [4].
- Si la valeur initiale et la valeur finale sont identiques, la boucle est effectuée une seule fois.

Exemple 6 :

Ecrire un algorithme qui permet d'afficher le message « Bonjour tout le monde» 10 fois.

```
Algorithme Bonjour_10_fois
Variables     I : entier
Début
|      Pour I=1 à 10 faire
|      |   Afficher ('Bonjour tout le monde ')
|       Fin pour
Fin
```

Exemple 7 :

Ecrire un algorithme qui permet de calculer la somme S = 1 + 2 + 3 + 4 + ….. + 100

```
Algorithme Somme
Variables      S, I : entier
Début
|       S ←── 0
|      Pour I=1 à 100 faire
|      |    S ←── S + I
|      Fin pour
|   Afficher ('Somme S = ', S)
Fin
```

3.2.2 Boucle Tant que

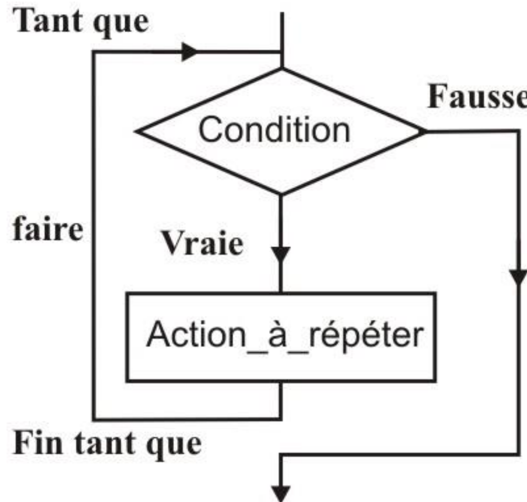
Ce type de boucle permet de répéter une série d'instructions tant qu'une certaine condition est vérifiée. On présente la boucle TANT QUE comme suit :

Tant que Condition **faire**

Ensemble des instructions à répéter

Fin Tant que

L'organigramme de cette boucle est donné comme suit :



Remarques :

- Au contraire de la boucle **pour**, le nombre de répétition est inconnu avant l'exécution.
- Le test de la condition est effectué avant d'entrer dans la boucle. Par conséquent, si la condition n'est pas vérifiée, on n'y entre pas, et on passe à l'instruction suivante.
- La boucle **Tant que** est plus générique et peut être utilisée pour remplacer la boucle **pour**.

- L'incrémentation d'un compteur (si c'est utilisé avec la boucle **Tant que**) n'est pas automatique comme dans le cas de la boucle **pour**. Par conséquent, il est obligatoire qu'une instruction doive changer sa valeur après (ou avant) chaque itération.

Exemple 8 :

Ecrire un algorithme qui permet de calculer la somme S = x + (x+1) + (x +2) + ... + 1000

Sachant que x est un nombre entier strictement positif inférieur à 1000 donné par l'utilisateur.

```
Algorithme Somme
Variables     S, I, x : entier
Début
|     S ←— x       I ←— 1
|     Afficher ('Donner x : ')
|     Saisir (x)     // lecture de x
|     Tant que (x ≤ 0) ou (x >999) faire          // contrôle de la valeur de x
|     |    Afficher ('Erreur, x doit être stritement positive et inférieur de 1000 ')
|     |    Afficher ('Donner x : ')
|     |    Saisir (x)
|     Fin Tant que
|     Tant que (x + I) ≤ 1000 faire
|     |    S ←— S + x + I
|     |    I ←— I + 1
|     Fin Tant que
|   Afficher ('Somme S = ', S)
Fin
```

Exemple 9 :

Ecrire un algorithme qui permet de calculer la somme des nombres pairs tant que cette somme est inférieure ou égale à 500. S = 2 + 4 + 6 + 8 + ….. + N

```
Algorithme Somme2
Variables     S, I : entier
Début
|     S ←— 0       I ←— 2
|     Tant que ((S + I) ≤ 500) faire
|     |    S ←— S + I
|     |    I ←— I + 2
|     Fin Tant que
|   Afficher ('Somme S = ', S)
Fin
```

Exemple 10 :

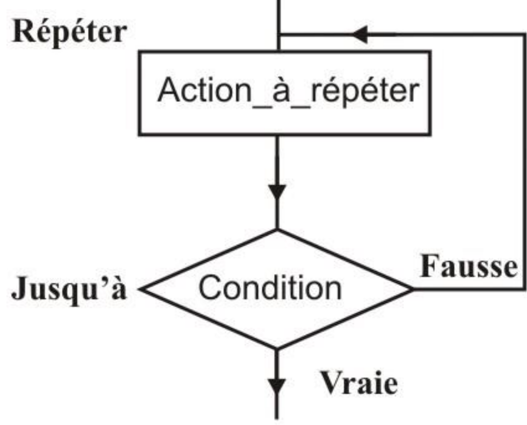
Ecrire un algorithme qui permet de calculer la somme. $S = 1 + \frac{3}{2} + 2 + \frac{5}{2} + \ldots + \frac{99}{2} + 50$.

```
Algorithme Somme3
Variables     S, I : réel
Début
        S ← 0        I ← 1
        Tant que I ≤ 50 faire
            S ← S + I
            I ← I + 1/2
        Fin Tant que
    Afficher ('Somme S = ', S)
Fin
```

3.2.3 Boucle Répéter

Cette structure est très similaire à la structure Tant Que du fait que leur exécution dépendent d'une condition. Elle permet de répéter une série d'instructions jusqu'à la vérification d'une certaine condition. On présente la boucle REPETER comme suit :

L'organigramme de cette boucle est donné comme suit :



Remarques :

- Le test de la condition n'est effectué qu'à la fin de la boucle. Par conséquent, cette boucle est exécutée au moins une seule fois même si la condition est vérifiée avant l'entrée.
- Une boucle Répéter peut être servis pour remplacer la boucle Tant que, en y ajoutant un test d'entrée si nécessaire.

Exemple 11 :

Ecrire un algorithme qui permet de calculer la somme de N termes de la série suivante :

$$S = 1 - \frac{3}{2} + 2 - \frac{5}{2} + 3 - \frac{7}{2} + \ldots\ldots\ldots + (-1)^{I+1} \times \frac{I+1}{2} + \ldots\ldots\ldots + (-1)^{N+1} \times \frac{N+1}{2}$$

```
Algorithme Somme4
Variables     S : réel        I, T, N : entier
Début
|       S ⟵ 0       I ⟵ 1      T ⟵ 1
|       Afficher ('Donner N : ')          Saisir (N)
|       Tant que I ≤ N faire
|       |    S ⟵ S + T × (I + 1)/ 2
|       |    I ⟵ I + 1
|       |    T ⟵ T × (-1)
|       Fin Tant que
|    Afficher ('Somme S = ', S)
Fin
```

Chapitre 03
Les Tableaux

1. Introduction

Jusqu'à présent, nous n'avons pas utilisé que des variables de type prédéfini simple (entier, réel, caractère ou logique). Dans le présent chapitre, nous allons voir un autre type de données, c'est le type structuré "Tableau" qui nous permet de regrouper des données de même type en mémoire, et d'y accéder et manipuler. Nous nous limiterons aux tableaux à une dimension (Vecteurs) et les tableaux à plusieurs dimensions (Matrices).

2. Les Tableaux à une dimension

Définition 3.1 : Tableau

On appelle un Tableau T d'ordre n, un ensemble de n éléments de même type (simple) disposés dans n cases adjacentes. Un Tableau possède un identificateur (nom du Tableau) et un indice qui indique le numéro de chaque case [5].

Exemple 1:

Soit T un tableau de 10 éléments entiers disposés dans 10 cases.

T									
-3	8	7	14	0	-1	13	1	4	2

2.1. Déclaration

Le type tableau à une dimension peut être déclaré comme suit :

tableau [*nombre_éléments*] *des type_élément* ;

Où ***nombre_éléments*** est le nombre des éléments dans le tableau, et ***type_élément*** indique le type des valeurs qui seront contenues dans les cases du tableau.

Exemple 2:

variables Tab1 : tableau [5] des entiers ; //Tab: est tableau de 05 éléments entiers.

Tab1				
-11	10	1	41	3

Tab1 : représente l'identificateur (le nom du tableau).

Tab1[i] : représente le i^{ième} élément dans le tableau Tab1.

Tab1[5] : est égal à 3.

Remarque :

- La déclaration d'un tableau T [N] où N est un nombre variable des éléments est incorrecte, car N ne doit pas être une variable.
- Le premier élément d'un tableau a un indice de 1.
- Comme toutes autres types, on peut déclarer plusieurs tableaux de même nombre et de même type des éléments, en les séparant par des virgules.
- Les éléments d'un tableau (Tab1 par exemple) peuvent être manipulés comme toutes autres types de variables comme suit :

```
Tab1[ 1 ] ←— x                    // x est une variable avec une valeur définie
Tab1[ 2 ] ←— 2
Tab1[ 3 ] ←— Tab1[1] + Tab[2]
Tab1[ i ] ←— Tab1[i - 1] + 2      // la variable i prend des valeurs dans l'intervalle de 1 à 5
```

2.2 Lecture (Remplissage)

On peut remplir un tableau T de m éléments par une suite d'instructions de lecture comme suit :

Saisir(T[1]) Saisir(T[2]) … Saisir(T[m])

Mais, on remarque que l'instruction Saisir est répétée m fois. Et afin d'éviter cette répétition, on va utiliser une des structures répétitives pour la lecture des éléments d'un tableau.

```
Pour i=1 à m faire
|       Saisir(T[ i ])
Fin pour
```

2.3 L'écriture (Affichage)

De la même façon, on peut afficher les m éléments d'un tableau T, comme suit :

Afficher(T[1]) Afficher(T[2]) … Afficher(T[m])

Ou bien, en utilisant la boucle "Pour " comme suit :

```
Pour i=1 à m faire
|       Afficher(T[ i ])
Fin pour
```

Exemple 3:

Soit l'algorithme suivant qui permet de lire les éléments d'un tableau T de 20 éléments entiers, puis multiplie les éléments d'u tableau par 2, et affiche le nouveau tableau.

```
Algorithme Multiple
Variables I : entier ; T : Tableau [20] des entier;
Début
  //lecture des éléments du tableau T
      Afficher ('Donner les elements du tableau T : ')
      Pour i=1 à 20 faire
           Saisir(T[ i ])
      Fin pour
  // Multiplication des éléments du tableau T par 2
      Pour i=1 à 20 faire
           T[ i ] ←── T[ i ] * 2
      Fin pour
  //Affichage des éléments du nouveau tableau T
      Afficher (' Le Tableau de résultat est :  ')
      Pour i=1 à 20 faire
           Afficher (T[ i ], '  ')
      Fin pour
Fin
```

3. Les Tableaux à deux dimensions

Dans ce cas, un Tableau T (ou bien, une matrice) peut être considéré comme une grille d'ordre n en lignes et d'ordre m en colonnes. Par conséquent, deux indices sont nécessaires pour indiquer les cases. Si une matrice a un nombre de colonnes égale au nombre de lignes, on parle d'une ***matrice carrée***.

Exemple 3:

Soit T une matrice de 5 lignes et 4 colonnes des éléments entiers. Ce tableau peut être vu comme suit :

Mat

1	2	3	4
11	21	18	-9
3	6	0	-14
-7	8	5	-2
0	21	41	1

Mat : représente l'identificateur de la matrice .

Mat[i , j] : représente l'élément dans la $i^{ième}$ ligne et la $j^{ième}$ colonne de la matrice Mat.

Mat[4 , 3] : est égal à 5.

3.1. Déclaration

La déclaration d'un tableau à deux dimensions est donnée comme suit :

T : Tableau [N,M] des entiers ; ou bien **T : Matrice** [N,M] des entiers ;

Où N représente le nombre des lignes et M représente celle des colonnes. Donc le tableau T aura un nombre des éléments = N * M.

Exemple 4:

Variables Mat1 : Tableau [3,3] des entiers
Mat2 : Matrice [10,5] des entiers

La première déclaration se correspond à une matrice carrée de trois lignes, alors que la deuxième représente une matrice de 10 lignes et 5 colonnes:

3.2 Lecture (Remplissage)

On peut lire les éléments de la matrice Mat1 comme suit :

```
Pour I = 1 à  N  faire
|   Pour J = 1 à M  faire
|   |   Saisir (Mat1[ I,J ])
|   Fin pour
Fin pour
```

Où on a utilisé deux boucles « Pour » imbriquées. L'indice I est utilisé pour parcourir les lignes et l'indice J pour parcourir les colonnes. En plus, car la boucle « Pour I » englobe la boucle « Pour J » ça implique que pour chaque ligne I on doit lire tous les colonnes. Autrement dit, les éléments de la matrice sont lus ligne par ligne. Par conséquent, si on veut lire les éléments de la matrice colonne par colonne on inverse les boucles comme suit :

```
Pour J = 1 à  M  faire
|   Pour I = 1 à N  faire
|   |   Saisir (Mat1[ I,J ])
|   Fin pour
Fin pour
```

3.3 L'écriture (Affichage)

De la même façon, on peut afficher les éléments de la matrice Mat1 comme suit :

```
Pour I = 1 à  N  faire
|   Pour J = 1 à M  faire
|   |   afficher (Mat1[ I,J ])
|   Fin pour
|     Afficher (' ') // instruction pour assurer une retour à la ligne
Fin pour
```

Remarque

- L'ajout de l'instruction **afficher** (' ') n'est pas obligatoire que pour assurer un bon affichage similaire à une matrice (avec des lignes et des colonnes).

Exemple

Ecrire un algorithme qui permet de lire les éléments de deux matrices carrés M1 et M2 de 5 lignes et 5 colonnes, puis calcule la multiplication de ces deux matrices.

```
Algorithme Multiplication
Variables I, J, K : entier
          M1, M2, M3 : matrice [5,5] des réels
Début
  // lecture des éléments de la matrice M1
   Pour I = 1 à  5  Faire
      Pour J = 1 à 5  Faire
         Afficher ('taper l''élément M1[ ', I, ', ',J, '] :  ')
         Saisir (M1[ i,j ] )
      Fin pour
   Fin pour
  // lecture des éléments de la matrice M2
   Pour I = 1 à  5  Faire
      Pour J = 1 à 5  Faire
         Afficher('taper l''élément M2[ ', I, ', ',J, '] :  ')
         Saisir (M2[ i,j ])
      Fin pour
   Fin pour
  // Multiplication des deux matrices
   Pour I = 1 à  5 Faire
      Pour J = 1 à  5 Faire
         M3[ i,j ] ←── 0;
         Pour K = 1 à  5 Faire
            M3[ i,j ]  ←──  M3[ i,j ] + M1[ i,k ] * M2[ k,j ];
         Fin pour
      Fin pour
   Fin pour
  // Affichage du résultat
    Pour I = 1 à  5  Faire
       Pour J = 1 à 5  Faire
          Afficher (M3[ I,J ]);
       Fin pour
    Fin pour
Fin
```

Remarques

- On note qu'il est nécessaire que le nombre de colonnes de la première matrice M1 doive être égal au nombre de ligne de la deuxième matrice M2 pour qu'on puisse effectuer la multiplication entre deux matrices M1 et M2.
- La matrice de produit aura un nombre de lignes égale au nombre de lignes de la première matrice et un nombre de colonne égale au celle de la deuxième matrice.

Chapitre 04
Les chaînes de caractères

Motivation

Dans notre monde, chaque objet a au moins un nom lui permettant d'être reconnu ou identifié. Ce ou ces noms peuvent évidemment être complétés par un ou plusieurs chiffres, cependant, il est rare que seul un code numérique soit utilisé. Cette situation implique que tout traitement informatique comportera des manipulations de tels noms qui sont appelés "chaînes de caractères". Formellement, on appelle une chaîne de caractères toute suite de zéro, un ou plusieurs caractères accolés [6].

1. Les caractères

Si une variable est déclarée de type caractère, il va occuper un octet en mémoire. Comme nous avons noté précédemment, ce type représente le domaine des caractères qui contient les lettre alphabétiques minuscules, les caractères numériques, les caractères spéciaux (., ?, !, <, >, =, *, +, ...etc), et le caractère espace ...etc.

1.1. Qu'est-ce que le code ASCII ?

Vu que les données sont stockées sous forme numérique dans la mémoire de l'ordinateur, la solution pour stocker les caractères est de donner à chaque caractère un code numérique équivalent. Là, le code ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) est apparu. C'est un code standardisé qui permet d'unifier la représentation des caractères ainsi qu la communication entre les ordinateurs.

Le code ASCII, ou table ASCII, est basé sur un principe assez simple où chaque caractère (chiffre, lettre, etc.) possède un code numérique pour pouvoir être stocké et interprété par un ordinateur.

Dans sa première version, le code ASCII représente les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127). Ensuite, il a été étendu pour utiliser 8 bits (2^8 = 256 caractères) afin de permettre le codage des caractères nationaux (non seulement anglais tels que les caractères accentués comme : ù, à, è, é, â,...etc) et les caractères semi-graphiques.

En effet, d'autres codages existent adaptés à diverses solution de stockage de l'information (Unicode sur 16 bits, DCB, EBCDIC,...).

La table ASCII regroupe donc l'ensemble des caractères comme suit :

1. Les codes 0 à 31 représentent des caractères de contrôle qui permettent de réaliser certaines actions précises comme le caractère NULL (#00), Bip sonore ("BEL" #07), un saut de ligne ("Entrée" #10), Retour chariot ("CR" #13), ... etc.
2. L'ensemble des chiffres de '0' à '9' (à partir du code #48 à #57).
3. Les lettres de l'alphabet en majuscules de 'A' à 'Z' (à partir du code #65 à #90).
4. Les lettres de l'alphabet en minuscules de 'a' à 'z' (à partir du code #97 à #122).

En plus, il regroupe aussi d'autres caractères semi-graphiques comme les barres verticales, les accolades ouvrante ou fermante, les crochets, les accents, ... etc.

La figure suivante représente la table ASCII à 8 bits

0		24	↑	48	0	72	H	96	`	120	x	144	É	168	¿	192	└	216	╪	240	≡
1	☺	25	↓	49	1	73	I	97	a	121	y	145	æ	169	⌐	193	┴	217	┘	241	±
2	☻	26	→	50	2	74	J	98	b	122	z	146	Æ	170	¬	194	┬	218	┌	242	≥
3	♥	27	←	51	3	75	K	99	c	123	{	147	ô	171	½	195	├	219	█	243	≤
4	♦	28	∟	52	4	76	L	100	d	124	|	148	ö	172	¼	196	─	220	▄	244	⌠
5	♣	29	↔	53	5	77	M	101	e	125	}	149	ò	173	¡	197	┼	221	▌	245	⌡
6	♠	30	▲	54	6	78	N	102	f	126	~	150	û	174	«	198	╞	222	▐	246	÷
7		31	▼	55	7	79	O	103	g	127	⌂	151	ù	175	»	199	╟	223	▀	247	≈
8		32		56	8	80	P	104	h	128	Ç	152	ÿ	176	░	200	╚	224	α	248	°
9		33	!	57	9	81	Q	105	i	129	ü	153	Ö	177	▒	201	╔	225	ß	249	∙
10		34	"	58	:	82	R	106	j	130	é	154	Ü	178	▓	202	╩	226	Γ	250	·
11	♂	35	#	59	;	83	S	107	k	131	â	155	¢	179	│	203	╦	227	π	251	√
12	♀	36	$	60	<	84	T	108	l	132	ä	156	£	180	┤	204	╠	228	Σ	252	ⁿ
13		37	%	61	=	85	U	109	m	133	à	157	¥	181	╡	205	═	229	σ	253	²
14	♫	38	&	62	>	86	V	110	n	134	å	158	₧	182	╢	206	╬	230	µ	254	■
15	☼	39	'	63	?	87	W	111	o	135	ç	159	ƒ	183	╖	207	╧	231	τ	255	a
16	►	40	(	64	@	88	X	112	p	136	ê	160	á	184	╕	208	╨	232	Φ		
17	◄	41	)	65	A	89	Y	113	q	137	ë	161	í	185	╣	209	╤	233	Θ		
18	↕	42	*	66	B	90	Z	114	r	138	è	162	ó	186	║	210	╥	234	Ω		
19	‼	43	+	67	C	91	[	115	s	139	ï	163	ú	187	╗	211	╙	235	δ		
20	¶	44	,	68	D	92	\	116	t	140	î	164	ñ	188	╝	212	╘	236	∞		
21	§	45	-	69	E	93	]	117	u	141	ì	165	Ñ	189	╜	213	╒	237	φ		
22	▬	46	.	70	F	94	^	118	v	142	Ä	166	ª	190	╛	214	╓	238	ε		
23	↨	47	/	71	G	95	_	119	w	143	Å	167	º	191	┐	215	╫	239	∩		

Figure 4.1 : La table ASCII

Astuce :

Le code ASCII peut être servis pour taper des caractères à l'aide de leur code (sur les ordinateurs avec Windows).

Par exemple, pour obtenir le caractère "(" (parenthèse ouvrante), on appuie et en continuant à appuyer sur la touche "alt". Puis, on tape le numéro "40", qui est le numéro du caractère "(" dans le tableau ASCII. Ensuite, on arrête d'appuyer sur la touche "alt" pour avoir le caractère correspondant.

Représentation des valeurs

Une constante de type caractère est représentée par un seul caractère mis entre deux quotes simples (apostrophes).

Exemple :

Soit x, y sont des variables de type caractère ;

x ⟵ 'A' y ⟵ '2'

Remarque

Le caractère ***apostrophe*** est doublé et placé entre quotes. ' '' '

1.2. Opérateurs sur le type caractère

Il existe plusieurs opérateurs, on cite ici les plus utiles.

1 : Les opérateurs de comparaison («**<**», «**>**», «≤», «≥», «**=**» et «≠») peuvent être utilisés pour comparer deux caractères car ils sont ordonnés.

2 : La fonction standard **Ordre(**caractère**)** renvoie le code correspondant au caractère mis en argument de la fonction.

3 : La fonction inverse **Car(**valeur**)**, fournit le caractère dont le code est l'argument de la fonction **Car**.

4 : Les fonctions **Succ(**caractère**)** , qui retourne le caractère suivant dans la table ASCII ; et **Pred(**caractère**)** : qui retourne le caractère précédent.

Exemple :

Supposant x, y,z sont des variables de type caractère et t de type entier;

Instruction	**Valeur de la variable après l'exécution**
x ⟵ ' '	x contient le caractère vide
y ⟵ **Car**(41)	y contient le caractère '**)**'
t ⟵ **Ordre**('B')	t contient le nombre 66
z ⟵ **Succ**('f')	z contient le caractère '**g**'
z ⟵ **Pred**('x')	x contient le caractère '**w**'

2. Les chaînes de caractères

Une variable déclarée de type **chaîne** (chaîne de caractère) est un objet qui peut contenir une suite de zéro, un ou plusieurs caractères accolés. Le type **chaîne** définit des variables "chaînes de caractères" ayant au maximum 255 caractères (cas de base). Une chaîne de caractère peut en contenir moins si cela est spécifié lors de sa déclaration où le nombre de caractères (compris entre 1 et 255) sera mis entre les crochets.

Exemple

Variables ch1: chaîne; ch2: chaîne [25];

// La longueur de la chaîne ch1 est 255

// La longueur de la chaîne ch2 est 25

Remarque :

- Le type chaîne est en fait un tableau de caractères à une dimension, mais il est enrichi par d'autres fonctions spéciales facilitant sa manipulation.
- Pour modifier un seul caractère de la chaîne, on utilise l'instruction suivante :

  ch[index] ⟵ lettre

  Tel que index représente l'ordre du caractère à modifier.
- Une chaîne qui n'est composée d'aucune lettre est appelée chaîne vide.
- Une constante de type "caractère" peut être considérée comme une constante de type "chaîne de caractères" de longueur 1.

2.1. Manipulation des chaînes de caractères.

1. Affectation

Lorsqu'une valeur est affectée à une variable chaîne de caractères, on procède comme pour un type numérique mais cette valeur doit être mis entre deux quotes simples. L'affectation est aussi possible entre deux chaînes de longueur quelconque, à condition que la taille de la chaîne affectée ne soit pas plus longue.

Exemple

Instructions	**Nombre de caractères après l'affectation**
ch ⟵ 'A'	le nombre de caractères dans ch = 1
ch1 ⟵ 'Bonjour'	le nombre de caractères dans ch1 = 7
ch2 ⟵ 'tout le monde'	le nombre de caractères dans ch2 = 13
ch3 ⟵ ch1 + ' ' + ch2	le nombre de caractères dans ch3 = 21

2. Lecture et écriture

La lecture des chaînes de caractères se fait avec l'instruction **Saisir** et l'écriture avec l'instruction **Afficher**.

Remarque

On note que les opérations de lecture et d'affichage des chaînes de caractère ne nécessite pas une boucle comme dans le cas des tableaux.

Exemple

Saisir (ch1) ; // pas de boucle pour lire une chaîne de caractères.

Afficher(ch1) ; // pas de boucle pour écrire une chaîne de caractères.

2.2. Opérations et fonctions prédéfinies

Il existe aussi plusieurs opérations et fonctions prédéfinies pour le type chaîne de caractères. On cite ici quelques uns.

1 : Les opérateurs de comparaison tels que : («=» et «≠», ...etc).

2 : L'opérateur d'addition (concaténation) : c'est l'opérateur + .

3 : La fonction ***Longueur*** : cette fonction retourne le nombre de caractères dans une chaîne de caractère.

Exemple

```
ch1 ←  '' ;
x   ←  longueur(ch1)                // x sera égale 0
ch2 ←  'abcd' ;
y   ←  longueur(ch2)                // y sera égale 4
ch3 ←  ' ' ;
z   ←  longueur(ch3)                // z sera égale 1
ch4 ←  ch1 + ch2 + ch3 + ch2        // ch4 sera égale 'abcd abcd'
t   ←  longueur(ch3)                // t sera égale 9
```

Chapitre 05
Les types définis par l'utilisateur

Motivation

Jusqu'à présent, notre connaissance des types et des instructions est relativement limitée. Nous n'avons vu que quelques types de base (entier, réel, caractère et logique). Ces types ont été utilisés pour décrire des structures composées de plusieurs éléments de même type, telles que les tableaux et les chaînes de caractères.

Néanmoins, ils nous ne permet pas de regrouper des informations de différent type liées au même objet dans une seule structure, par exemple: comment décrire un nombre complexe ? Ou bien des informations sur un étudiant ?

Par conséquent, nous allons approfondir nos connaissances dans deux directions, à savoir:

- La possibilité de définir de nouveaux types par l'utilisateur de façon plus agréable pour décrire ses objets.
- Présenter les instructions complémentaires et nécessaires pour faciliter le traitement de ces nouveaux types.

La notion de type définis par l'utilisateur nous permet de représenter des structures de données composées de plusieurs éléments ou des données diffèrent des types standard.

1. Notion de type

Un type est composé d'un ensemble de valeurs ainsi que des opérations possibles sur ces valeurs.

Exemples:

- Le type **réel** est formé d'un ensemble de valeurs réelles ainsi que des opérations arithmétiques (+, –, *, /) et des opérations de comparaison (<, <=, >, >=, =, ≠).
- Le type **Logique** est formé d'un ensemble de valeurs logiques (vrai, faux) ainsi que des opérations logiques (non, et, ou, ou exclusif).
- Le type caractère est formé d'un ensemble de 256 caractères (Table ASCII) et quelques opérations.

Déclaration des types

Afin de pouvoir utiliser un nouveau type dans un algorithme, il faut d'abord le déclarer dans la partie de déclaration. Cette partie commence par le mot réservé "type" et comprend la déclaration de tous les types définis par l'utilisateur qui seront utilisés dans l'algorithme.

En effet, chaque catégorie de type possède un style spécial de déclaration et seul le début de la déclaration d'un type est commun à tous les types qui peut être donné comme suit:

```
nom_du_type =
```

Remarques:

1. L'ordre des déclarations est ordonnées comme suit:
   - Déclaration des constantes
   - Déclaration des types
   - Déclaration des variables
2. Par convention, et afin de différencier les noms des types de celles des variables, on peut faire précéder le nom du type par le préfixe t_.

2. Les types définis par l'utilisateur

2.1. Les enregistrements

Même si la structure de tableau nous permet de décrire une structure formée de plusieurs éléments de même type, il nous ne permet pas de regrouper des informations liées au même élément et qui n'ont pas forcément le même type. Or il existe d'autres données qui sont composées d'éléments de types différents comme par exemple:

- Les dates (année, mois, jour)
- Les nombres complexes (partie réelle, partie imaginaire)
- Les fiches estudiantines ou personnelles (nom, prénom, date de naissance, adresse, ...)
- Les fiches bibliographiques (titre du livre, auteur, date de parution, ISBN...)

La nature différente de ces éléments a poussé les programmeurs à utiliser des structures appelées "Les enregistrements" qui sont mieux adaptées pour représenter ce type de données.

Définition

Un enregistrement est une structure de donnée formée d'un ou de plusieurs éléments (ou champs) pouvant être de types différents.

2.1.1. Déclaration

La déclaration des enregistrements se fait dans la partie de déclaration des types comme suit :

```
nom = enregistrement
champ1 : type1
champ2 : type2
. . . . . . . .
champN : typeN;
fin enregistrement
```

Exemple

Déclarons un enregistrement étudiant et un autre pour un nombre complexe.

```
Etudiant=enregistrement
matricule : chaîne[10]
nom : chaîne[25]
prénom : chaîne[25]
 age : entier
note1 : réel
note2 : réel
moy : réel
fin enregistrement
```

```
Complexe=enregistrement
    x : entier
 y : entier
 fin enregistrement
```

Remarques

- Les champs d'un enregistrement peuvent être de n'importe quel type (sauf un type fichier, à voir ultérieurement).
- Les enregistrements ne peuvent pas être manipulés (ou traités) tout ensemble, mais seulement à travers ses champs.
- Les opérations permises sur les enregistrements sont seulement l'affectation et le passage comme paramètre. Cependant, il possible de manipuler ses champs (un par un) comme toutes autres variables de type similaire.
- Il n'est pas possible de déclarer une constante de type enregistrement.
- Le type enregistrement ne peut pas être un résultat d'une fonction.

2.1.2. Manipulation

1. Un enregistrement doit être manipulé champ par champ où on accède à ses champs en indiquant le nom de l'enregistrement suivi d'un point et du nom du champ.

Exemple :

Prenons l'exemple précédent de l'étudiant, les champs d'un enregistrement peuvent être manipulés par l'affectation.

```
Etudiant.matricule ⟵ '20/S001'
      Etudiant.nom ⟵ 'Achraf'
   Etudiant.prénom ⟵ 'Boucherit'
     Etudiant.sexe ⟵ 'M'
    Etudiant.note1 ⟵ 19
    Etudiant.note2 ⟵ 18.75
```

Ou bien par lecture, affichage et d'autres opérations.

```
Saisir(Etudiant.matricule)
Saisir(Etudiant.nom)
Afficher(Etudiant.nom)
Etudiant.moy ⟵ (Etudiant.note1 + Etudiant.note2)/2
```

2. Une deuxième instruction **avec … faire**.
Cette instruction permet de manipuler directement les champs d'un enregistrement sans faire ajouter le nom de l'enregistrement et le point.

Exemple:
Ecrire un algorithme qui permet de lire les renseignements de 20 étudiants et de calculer leurs moyens, puis afficher la liste des étudiants admis.

```
Algorithme Moyen
Constantes N = 20
Types
    Etudiant=enregistrement
    matricule : chaîne[10]
    nom : chaîne[25]
    prénom : chaîne[25]
    note1 : réel
    note2 : réel
    moy : réel
    fin enregistrement
Variables  I : entier      Etud : Etudiant     Classe : tableau [N] des Etudiant
Début
| // Lecture des renseignements des étudiants
|  Pour I = 1 à  N Faire
|  |  avec Etud Faire
|  |  |   Afficher ('taper le matricule = ')  Saisir (matricule)
|  |  |   Afficher ('taper le nom = ')  Saisir (nom)
|  |  |   Afficher ('taper le prenom = ')  Saisir (prenom)
|  |  |   Afficher ('taper le note1 = ')  Saisir (note1)
|  |  |   Afficher ('taper le note2 = ')  Saisir (note2)
|  |  |      moy  ←  (note1 + note2) / 2
|  |   Fin avec
|  |         Classe [I] ←  Etud
|  Fin pour
| // affichage de la liste des étudiants admis
|  Pour I = 1 à  N Faire
|  |   avec classe [I] Faire
|  |   |   Si ( moy  >= 10)  alors
|  |   |   |  Afficher (nom, ' ', prenom : 'admis avec moyenne =', moy )
|  |   |   Finsi
|  |    Fin avec
|  Fin pour
Fin
```

2.2. Les types énumérés

Les types énumérés constituent un autre type qui peuvent être définis par l'utilisateur/programmeur, où il peut listez un ensemble de valeurs dans une énumération.
En d'autres mots, une énumération est une liste de valeurs définie par l'utilisateur.

2.2.1. Déclaration

La déclaration des types énumérés se fait avec une syntaxe spéciale et voici quelques exemples :

```
types
  t_Couleurs = (Rouge, Jaune, Vert, Marron, Bleu, Violet)
  t_Jours = (Samedi, Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi)
  t_Objets = (Table, Chaise, Tableau, Bureau)
  t_Logique = (Faux, Vrai)
  t_Mois = (Jan, Fev, Mar, Avr, mai, Juin, Jui, Aout, Sept, Oct, Nov, Dec)
```

Ensuite, une variable d'un type énuméré doit être déclaré. Par exemple :

```
Variables
  jour1, jour2 : t_jours        mois : t_mois
```

2.2.2. Manipulation

Les variables d'un type énuméré ne pourront pas avoir d'autres valeurs que celles que l'on a listées. Et on peut les manipuler dans un algorithme grâce à :

- l'Affectation :

```
jour1 ⟵ Vendredi             jour2 ⟵ jour1
```

- Fonctions prédéfinies

En principe, il existe trois fonctions qui sont :

pred(précédent), // retourne la valeur précédente

succ(suivant), // retourne la valeur précédente

ord(élément) // retourne l'ordre d'un élément (dans la déclaration), entre 0 et N-1).

Remarques :

- Chaque valeur de la liste (type énuméré) possède un rang associé, commençant à zéro.

- Le type énuméré peut être un type d'un champ dans un enregistrement.

Exemples :

```
pred(mardi) = lundi
succ(mardi) = mercredi
ord(mardi) = 3
```

Chapitre 06
Les sous-programmes

Motivation

Lors du développement d'un algorithme pour résoudre un problème complexe et/ou long, il est nécessaire de décomposer le problème principal en différents sous-problèmes moins complexes, afin d'alléger la tâche et d'éviter la répétition des traitements similaires, à plusieurs endroits dans l'algorithme. Par exemple :

Supposons que nous voulons effectuer le calcul suivant :

$$S = C_n^p + C_k^d \text{ telle que : } C_n^p = \frac{n!}{(n-p)!\,p!}$$

Dans cet exemple, l'opération de calcul de la factorielle sera répétée 3 fois pour chaque combinaison. Cependant, les langages de programmation ainsi que le langage algorithmique fournissent un moyen (sous-programmes) pour ne pas répéter les mêmes instructions.

Généralement, les sous-programmes nous permettent de :

- Eviter de copier un segment de code plusieurs fois dans le même algorithme.
- Décomposer le problème principal en un ensemble de petit sous problème, où chacun est résolu par un sous-programme indépendamment des autres.
- Constituer une bibliothèque des sous-programmes.

Définition

Un sous-programme est une séquence d'instructions ordonnée pour exécuter un traitement spécifique non élémentaire ou faire un calcul et il ne sera exécuté dans l'algorithme qu'après son appel, si nécessaire.

1. Les types des sous-programmes

Il existe deux types de sous-programmes:

1- **La procédure** est un sous-programme qui permet d'effectuer des traitements sur zéro, une ou plusieurs variable(s) et renvoyer les résultats au programme appelant. Par exemple : lecture, affichage, calcul et changement des valeurs des variables, ….

2- **La fonction** est un sous-programme qui permet de résoudre un problème précis (généralement un calcul) et renvoie un seul résultat de type simple (entier, réel, logique, caractère ou chaîne). La valeur retournée (calculée) est rendue disponible par l'intermédiaire du nom de la fonction.

Exemples

- Procédure : L'opération de permutation (échange du contenu) de deux variables du type entiers A et B ne peut pas être réalisée avec une instruction élémentaire. Par conséquent, le développeur doit la réalise en écrivant une procédure.
- Fonction : L'opération de déterminer la valeur maximale entre deux variables du type entiers A et B se fait par une instruction conditionnelle et retourne une valeur unique. Par conséquent, le sous-programme correspondent est une fonction.

Question :
Quel est le type du sous-programme correspondant pour déterminer le max et le min de trois nombres entiers (A, B et C).

2. Syntaxe d'un sous-programme

La syntaxe diffère selon le type du sous-programme

2.1. Syntaxe d'une procédure

```
procédure <nom_procédure>( <liste des paramètres> )
< déclaration des objets locaux de la procédure>
Début
  {corps de la procédure}
Fin
```

Cette ligne est appelée signature ou entête de la procédure

Exemple:

La procédure qui permet de dessiner une ligne sur l'écran avec un symbole et un nombre de répétition choisis par le développeur s'écrit comme suit :

```
procédure tracer_ ligne (c : caractère ; n : entier)
variables i : entier
Début
 | Afficher ("") // pour retourner à la ligne
 | Pour i = 1 à  n Faire
 | |   Afficher (c)
 | Fin pour
 | Afficher ("") // pour retourner à la ligne
Fin
```

2.2. Syntaxe d'une fonction

```
fonction <nom_fonction> ( <liste des paramètres> ) : <type de résultat>
< déclaration des objets locaux à la fonction>
Début
  {corps de la fonction}
  retourner(résultat)
Fin
```

Cette ligne est appelée signature ou entête de la fonction

La déclaration des paramètres (formelles) d'un sous-programme se fait similairement à la déclaration des variables

Exemple:

La fonction qui permet de calculer la moyenne de deux valeurs entières s'écrit comme suit :

```
fonction moyenne (x, y : entier) : réel
variable z : réel
Début
| z ⟵ (x+y)/2
| retourner (z)
Fin
```

Remarques :

- Les sous-programmes sont un moyen de réutilisation de segment de code. Par conséquent, ils peuvent être appelées plusieurs fois à partir du programme principal ou à partir d'autres sous-programmes, en recevant à chaque fois des paramètres avec des valeurs différentes.
- Les paramètres donnés dans l'en-tête d'un sous-programme sont appelés paramètres formels. Leurs valeurs ne sont pas connues lors de la création d'un sous-programme.
- Les paramètres spécifiés lors de l'appel d'un sous-programme sont appelés paramètres effectifs ou actuels. Ce sont les valeurs qui seront utilisées lors de l'exécution du sous-programme.
- La syntaxe de l'en-tête d'une fonction est assez proche de celle d'une procédure à laquelle on ajoute le type de la valeur qui sera retournée par la fonction.
- La fonction doit avoir une instruction à la fin en utilisant le mot clé "Retourner" pour renvoyer le résultat à l'algorithme ou au sous-programme appelant.

3. Appel des sous-programmes

Les sous-programmes peuvent être appelés (ou invoqué pour être exécutés) à partir de l'algorithme principal ou depuis un autre sous-programme.

Pour exécuter un sous-programme, il suffit de lui faire appel en écrivant son nom suivie des paramètres effectifs comme suit :

nom-sous-programme(para-1,para-2,...,para-n).

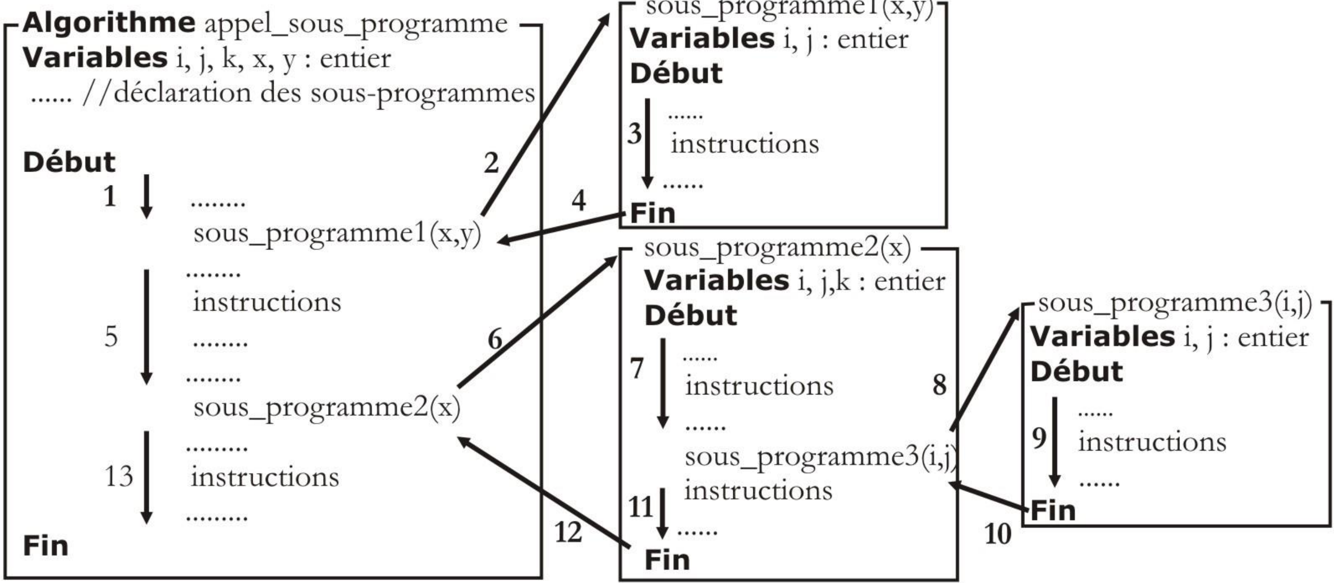
Plus précisément, l'appel d'une procédure est une instruction indépendante, du fait que la procédure exécute certain code et ne renvoie rien. Par exemple :

tracer_ligne ('*',25) // la procédure tracera une ligne avec 25 étoile

A la différence d'une procédure, une fonction est généralement appelée dans une instruction (affichage, affectation, ...) ou un autre sous-programme qui va utiliser sa valeur de retour. Par exemple : t ⟵ moyenne (x, y) * 10 / 2

Exemple

Soit l'exemple suivant qui illustre l'opération d'appel d'un sous-programme en numérotant le séquencement de l'exécution de l'algorithme.



4. Variables globales et variables locales

Définition

En programmation informatique, on appelle une ***variable globale*** toute variable définie au niveau de l'algorithme (ou du programme) principal ou au niveau supérieur. Par contre, une ***variable locale*** est une variable définie au sein d'un sous programme.

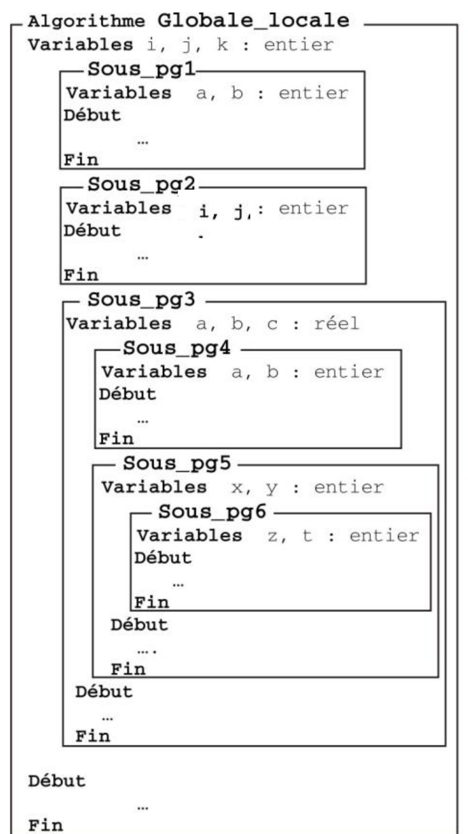
Notion de portée :

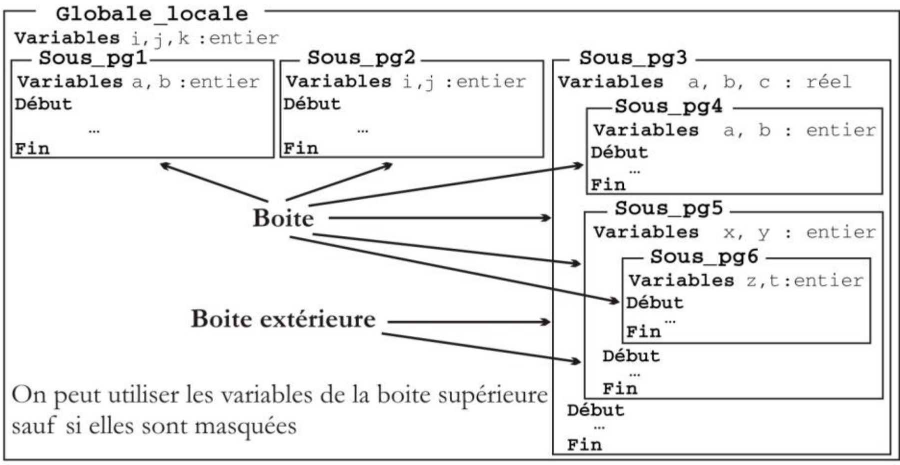
La portée d'une variable désigne l'ensemble des sous-programmes où cette variable va être accessible. Par conséquent, les variables peuvent avoir une portée locale et/ou une portée globale.

Remarques :

- La portée d'une variable globale est totale. C-à-d, elle sera accessible depuis tout les sous-programmes de l'algorithme (ou du programme) principal.
- La portée d'une variable locale est partielle. C-à-d, elle est limitée uniquement au sous-programme qui la déclare.
- Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale est masquée localement. C-à-d, la variable globale devient inaccessible dans ce sous-programme.
- Rappelons que les variables globales doivent être utilisées avec précaution, car elles créent des liens invisibles entre les sous-programmes. En plus, cette action risque de perdre l'utilité de la modularité.

Exemple :
Soit l'exemple suivant qui illustre la portée des variables globales et locales d'un algorithme.





Variables	Portée	
	Locale	Globale
Globale : i, j, k		Accessible partout sauf «Sous_pg2» Accessible partout
Sous_pg1 : a, b	Sous_pg1	
Sous_pg2 : i , j	Sous_pg2	
Sous_pg3 : a, b, c	Sous_pg3	Sous_pg5, Sous_pg6 Sous_pg4, Sous_pg5, Sous_pg6
Sous_pg4 : a, b	Sous_pg4	
Sous_pg5 : x, y	Sous_pg5	Sous_pg6
Sous_pg6 : z, t	Sous_pg6	

Exemple:

Ecrire un algorithme qui permet de calculer la combinaison C_n^p en utilisant les sous-programmes.

```
Algorithme Combinaison
Variables  i, n, p, C : entier
    fonction factorielle (x : entier) : entier
    variables f, i : entier
    Début
    |  f ⟵ 1
    |     Pour i=2 à x faire
    |     |  f ⟵ f * i
    |     Fin pour
    |  retourner (f)
    Fin
Début
|    // Lecture des valeurs positives de n et p
|     Répéter
|     |Afficher ('taper les valeurs de n et p = ')  Saisir (n,p)
|     Jusqu'à ((n≥0) et (n≥0))
|  C ⟵ factorielle(n)/(factorielle(n-p) x factorielle(p)) //appel de la fonction factorielle
|       Afficher ('C(',n,',',p,')=',C)
Fin
```

5. Modes de passage de paramètres

Revenons à l'exemple précédent et analysons-le. Nous avons défini la fonction factorielle, puis nous avons appelé cette fonction trois fois avec les paramètres effectifs n, p et n-p pour calculer la combinaison C_n^p. Lors de l'exécution, les paramètres formels seront substitués par les paramètres effectifs. On appelle cette opération "passage de paramètres", qui s'agit en effet d'un transfert de données entre l'algorithme principal et le sous-programme.
Généralement, on distingue deux modes de passages : passage par valeur et passage par variable (adresse ou référence).

5.1. Passage par valeur

Dans ce mode, la valeur de la variable passée en paramètre est copiée dans la variable locale définie dans la signature du sous-programme. Par conséquent, la valeur du paramètre transmis ne sera jamais affectée par les modifications dans le sous-programme. Autrement dit, les modifications ne seront pas effectuées que dans un seul sens. C-à-d, de l'algorithme principal vers le sous-programme (les modifications reste à l'intérieur du sous-programme). La déclaration d'un sous-programme utilisant ce mode de passage s'écrit à titre d'exemple comme suit :

```
Procedure nom_procédure (param1 :type1 ; param2 :type2)
Fonction nom_fonction (param1, param2 : type1) : Type_fonction
```

5.2. Passage par variable

Ce mode de passage de paramètre implique que toute modification du paramètre transmis dans le sous-programme appelé entraîne automatiquement la même modification sur la variable passée en paramètre dans l'algorithme principal. Autrement dit, les changements sont appliqués dans les deux sens (de l'algorithme principal vers le sous-programme et vice versa). En plus, la déclaration d'une variable formelle avec ce mode de passage de paramètres doit être précédé par le mot clé var (abréviation de variable) comme suit :

```
Procedure nom_procédure (Var param1, param2 :type1 ; param3 :type2)
Fonction nom_fonction (Var param1 : type1, param2 :type2) : Type_fonction
```

- Il est préférable de passer les variables globales en paramètre, s'il est nécessaire de les utiliser dans un sous-programme.
- Un même sous-programme peut utiliser les deux modes de passage de paramètre (par valeur et par adresse). Dans ce cas, il faut séparer les deux modes de passage par un (;).

Exemple:

Écrire le sous-programme et l'algorithme qui permet de trouver le PGCD de deux nombres entiers positifs A et B en utilisant la méthode d'Euclide (A>=B).

Soit R est le reste de la division de A par B :

- Si R = 0 alors le PGCD(A,B) = B, sinon, le PGCD(A,B) = PGCD(B,R).
- L'opération sera répétée jusqu'à ce que R s'annule.

```
Algorithme pgcd_euclide
variables a, b, pgcd : entier
  fonction trouve_pgcd(a,b: entier)
   variables r,q : entier
  Début
      Si (a<b) alors
        q ← a
        a ← b
        b ← q
      Finsi
     Répéter
        q ← [a/b]        // division entière
        r ← a – q * b  // reste de la division de a sur b
          si (r ≠ 0) alors
            a ← b
            b ← r
          Finsi
     Jusqu'à (r = 0)
      Retourner(b)
  Fin
Début
   Répéter
      afficher('Donner deux nombres entiers >0 : ')
      saisir(a, b)
   Jusqu'à ((a>0) et (b>0))
    pgcd ← trouve_pgcd(a,b)
   afficher ('pgcd de ', a, ' et ', b, ' = ', pgcd)
Fin
```

Exemple d'exécution :

Donner deux nombres entiers >0 : -16 8

Donner deux nombres entiers >0 : 18 12

pgcd de 18 et 12 = 6

Exemple:
Écrire le sous-programme et l'algorithme qui permet de permuter les valeurs de deux nombres entiers A et B donnés par l'utilisateur.

```
Algorithme permutation
variables a, b : entier
  procédure Lecture(var x, y : entier)
  Début
  |   afficher('Donner deux nombres entiers : ')
  |   saisir(x, y)
  Fin
  procédure permuter(var x, y : entier)
  variables z : entier
  Début
  | z ← x
  | x ← y
  | y ← z
  Fin
Début
|  Lecture(a,b)
|  afficher ('valeurs des variable avant la permutation')
|  afficher ('a =  ', a, ' et  b = ', b)
|  permuter(a,b)
|  afficher ('valeurs des variable après la permutation')
|  afficher ('a =  ', a, ' et  b = ', b)
Fin
```

Exemple d'exécution :
Donner deux nombres entiers : 5 8
valeurs des variable avant la permutation
a = 5 et b = 8
valeurs des variable après la permutation
a = 8 et b = 5

Références :

[1] J. TISSEAU, « Initiation à l'algorithmique », Presse Univ. Éc. Natl. D'Ingénieurs Brest, 2009.

[2] R. Christophe, « Bases d'algorithmique. Support de Cours au Lycée Vincent d'Indy ». 2015-2016.

[3] L, Baba Ahmed et S, Hocine, algorithmique et structure de données statistiques. OPU, 2016.

[4] E. Thiel, « Support de cours Algorithmes et programmation en Pascal ». Faculté des Sciences de Luminy, Université d'Aix-Marseille AMU, 1997.

[5] A. Rabia, F. Rachid, A. O. Mohand, B. Moufida, et Y. Smain, « Algorithmique : Cours et Exercices en Programmation Pascal ». Cours, Exercices et Programmation Pascal Première Année Universitaire, USTHB, 2018-2017.

[6] B. Pierre, « Introduction à l'informatique : Langage Pascal ». La Haute École d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD), 1992.