Cours Informatique

M1-Biochimie Appliquée

« Algorithmique et Structure de Données »

Préparé par : Dr. EUTAMENE AICHA

Année Universitaire 2020-2021

1. Introduction générale

Aujourd'hui avec la progression immense de la technologie et l'utilisation des applications informatiques dans presque tous les domaines de notre vie quotidienne ; la programmation des logiciels n'est pas devenu une tâche dédiée à un informaticien mais n'importe quel spécialiste dans son domaine (Médecine, chimie, automatique, biologie.....) doit apprendre à programmer pour résoudre ces problèmes. Pour cette raison nous avons rédigé ce cours à l'intention des étudiants de la première année du deuxième cycle universitaire LMD à la faculté de la science de la nature et de la vie, afin de motiver l'étudiant à programmer et préparer le pour les années prochaines dans son spécialité. A cet effet, ce polycopié constitue un manuel de cours et d'exercices d'apprentissage sur le domaine de l'algorithmique. Le contenu de ce cours est structuré en quatre chapitres : dans le premier chapitre, nous avons donné un aperçu générale sur l'algorithmique et nous décrivons ces notions élémentaires telles que: les variables et constantes et leurs types de base ainsi que les es instructions de traitement comme la lecture, l'écriture et l'instruction d'affectation.

2. Objectif du cours

A l'issue du présent cours, les compétences requises sont :

1. **Savoir résoudre** un problème bien précis en informatique.
2. **Savoir transcrire** les différentes étapes de résolution d'un problème sous forme d'algorithme ou un pseudo-code
3. **Prendre en main** les principes de programmation.

3. Plan du cours

- **Chapitre 1:** Principes de base de l'algorithmique
- **Chapitre 2:** Le traitement conditionnel
- **Chapitre 1 :** les boucles
- **Chapitre 2:** les tableaux

Chapitre 1 :
Principes de base de l'algorithmique

1.1 Pourquoi l'algorithmique ?

Un problème concret ne peut être résolu par un ordinateur que si les opérations nécessaires à cette résolution peuvent être décomposées en un nombre fini d'étapes élémentaires dont chacune peut être traitée individuellement. En d'autres termes, on doit indiquer à l'ordinateur de façon précise et détaillée comment un problème donné doit être résolu. Un tel procédé de résolution est appelé algorithme.
A cet effet, la résolution d'un problème donné par un ordinateur passe par une succession d'étapes à savoir :

1. Comprendre et analyser le problème.
2. Spécifier la méthode de résolution du problème en déterminant clairement les données à fournir à la machine et les résultats à obtenir.
3. Écrire l'algorithme
4. Traduire l'algorithme en un programme en utilisant un langage de programmation.
5. Exécuter le programme.

Dans ce chapitre, la notion d'algorithme sera détaillée.

1.2 La notion d'algorithme

Le mot « Algorithme » est inventé par le mathématicien « ». Un algorithme ALKHAWARISMI 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 langues : 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.

Figure 1. ALKHAWARISMI

L'algorithme permet de réaliser un traitement sur un ensemble de données en entrées *(Input)* pour produire des données en sorties*(Output)*. Les données en sorties représentent la solution du problème traité par l'algorithme.
Un algorithme peut être schématisé comme suit (figure ci-dessous):

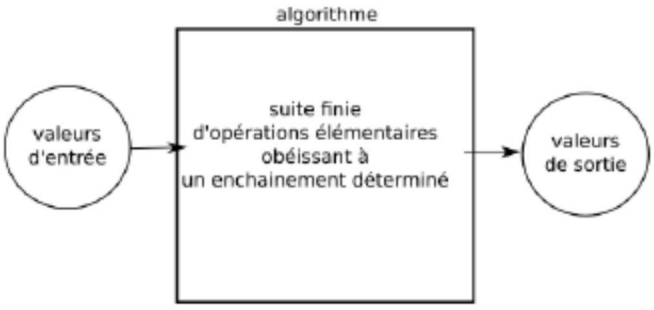


Figure 2. Input et output d'un algorithme

Le rôle de l'algorithme est fondamental. En effet, sans algorithme, il n'y aurait pas de programme (qui n'est que la traduction de cet algorithme dans un langage compréhensible par l'ordinateur). De plus, les algorithmes sont indépendants à la fois de l'ordinateur qui les exécute, des langages dans lesquels ils sont énoncés et traduits.

1.3 Définition générale d'algorithme

Un algorithme est une suite d'instructions élémentaires, qui s'appliquent dans un ordre déterminé à un nombre fini de données pour fournir un résultat.

N.B : Dans la définition, on dit "un" algorithme et non pas "l'" algorithme car, en général, la solution n'est pas unique.

1.4 La structure d'un algorithme

Écrire un algorithme consiste à décrire la méthode de résolution d'un problème c.à.d indiquer les actions simples que l'ordinateur doit accomplir pour obtenir les résultats désirés. Les actions sont généralement décrites par un symbole ou un verbe à l'infinitif choisi pour éviter les confusions.
En général, un algorithme comprend trois parties essentielles:

1. **La partie Entête :** cette phase consiste à donner un identifiant (nom) à l'algorithme.
2. **La partie déclaration** *:* cette phase correspond à la déclaration des variables nécessaires aux traitements et résolution du problème posé. Elles peuvent être de type :
   - Numérique (des nombres entiers ou réels)
   - Textuelle (des caractères, chaîne de caractères)
   - Booléenne (de type logique, à deux valeurs possibles « vrai » ou « faux »)
3. **La partie Corps :** Cette phase, qui correspond au corps de l'algorithme, consiste à spécifier toutes les étapes des instructions à donner pour une exécution automatique.

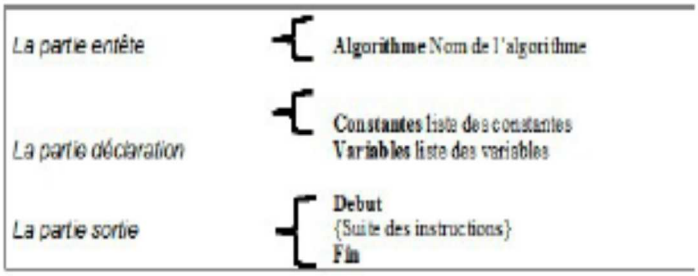


Figure 3. La structure d'un algorithme

1.5 Formalisme d'un algorithme

Il existe deux façons pour représenter un algorithme :

- Représentation textuelle en utilisant un pseudo-code
- Représentation graphique avec un Organigramme utilisant des symboles.

1.5.1 Pseudo-code

Un algorithme peut être écrit en utilisant LDA. Ce langage utilise un ensemble de mots clés et de structures permettant de décrire de manière complète et claire l'ensemble des opérations à exécuter sur des données pour obtenir des résultats. Cette série d'instructions proche d'un vrai langage de programmation, mais sans les problèmes de syntaxe. Série

d'instructions proche d'un vrai langage de programmation, mais sans les problèmes de syntaxe.

1.5.2 Organigramme

En algorithmique, on peut aussi utiliser un organigramme, c'est-à-dire représenter graphiquement l'algorithme à l'aide de symboles normalisés. 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.

Symbol	Signification
	Lecture et écriture
	Calcul et traitement
	Condition
	Connecteur
	Début et Fin
	Cheminement

Figure 4. Les symboles d'un organigramme

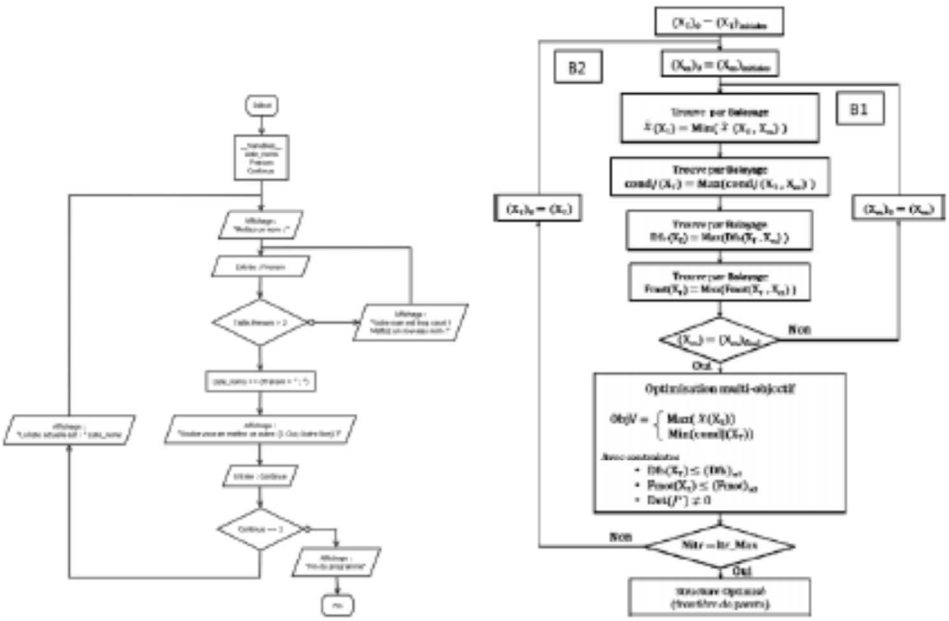


Figure 5. Exemples organigramme

1.6 Qualités d'un algorithme

Pour obtenir un bon programme, il faut partir d'un bon algorithme qui doit, entre autres, doit respecter les contraintes suivantes:

- Exploiter un ensemble restreint d'opérations de base (un ordinateur ne comprend que des instructions très simples)
- Produire les résultats voulus en exécutant un nombre fini de ces opérations (sinon l'ordinateur ne résoudra jamais le problème).
- être clair, facile à comprendre par tous ceux qui le lisent
- Présenter la plus grande généralité possible pour répondre au plus grand nombre de cas possibles.

1.7 Éléments de base de l'algorithmique

1.7.1 Variables et constantes

Il existe deux types dans un algorithme : les constantes et les variables.

a). Constantes

Les constantes et les variables sont des éléments fondamentaux, indispensables au bon déroulement d'un programme, caractérisés par un identificateur, une valeur et un type.

Une constante est un objet contenant une valeur qui ne peut jamais être modifiée. Elle peut représenter un chiffre, un nombre, un caractère,

Toutefois, contrairement à une variable la valeur d'une constante ne varie pas au cours de l'exécution du programme. Les constantes sont des objets contenant des valeurs non modifiables.

Syntaxe :

```
Constantes   Nom_Constante = Valeur
```

B). Variables

En informatique, une variable est une donnée stocké dans la mémoire de l'ordinateur. Elle contient une valeur pouvant être modifiée. Ce changement de valeur se fait par l'opération d'affectation.
La variable diffère de la notion de constante qui, comme son nom l'indique, ne prend qu'une valeur unique au cours de l'exécution de l'algorithme.
Une variable doit avoir un nom (ou identificateur), ainsi qu'un type (entier, réel etc...), qui définit la taille de la case mémoire.

Syntaxe :

```
Variables liste_variables1: Nom_Type1
          liste_variables2: Nom_Type2
          .......
          liste_variablesN: Nom_TypeN
```

1.7.2 Types de variable

Toutes les données (variable ou constante) d'un algorithme possèdent un type de données (domaine de valeurs possibles).

a). Définition : Le nom d'une variable (identificateur)

C'est une suite de caractères (lettres, chiffres, souligné ou tiret -8) dont le premier ne doit pas être un chiffre. Il permet de la retrouver (repérer) dans la mémoire de l'ordinateur.

b). Méthode : Règles d'écriture d'un identificateur

Le nom de la variable ou la constante obéit à des impératifs changeant selon les langages, En pseudo code algorithmique, on doit respecter ces règles, même si on est libre dans la syntaxe.

c). Types de base d'une variable

Les types déterminent l'ensemble des valeurs qu'elle peut prendre. Les types que nous utiliserons seront décrits dans les sections ci-dessous.

- **Le type Entier :** représente l'ensemble {..., -4,-3,-2,-1,0, 1, 2, 3, 4, ...}

Exemple :
Const x = 1 ;
Var A : entier ;
Var A, b : entier ;
Var coefficient_module : entier ;

- **Le type Réel : r**eprésente les valeurs numériques fractionnels et avec des virgule fixes (ou flottante)

Exemple :
Var A : réel ;
Var A, b : réel ;
Var moyennegénérale : réel ;

- **Le type Caractère :c**e type englobe tout les caractères nécessaires à l'écriture de texte comme :espace, lettres, chiffres, signes de ponctuation et caractères spéciaux (#, @, &, ...).

Exemple :
Var A : Caractère ;

- **Le type Booléen ou logique :** représente les deux valeurs logiques VRAI et FAUX

Exemple :
Var A : Booléen ;
Var A, C : Booléen;
Var ajournée: Booléen;

d) .Types d'opérateurs

Avant d'entamer cette section, ci- dessous quelques définitions importantes.

- ✓ **Opérateur :** est un outil qui permet d'agir sur une variable ou d'effectuer des calculs.
- ✓ **Opérande :** est une donnée utilisée par un opérateur.
- ✓ **Expression :** est un ensemble d'opérandes reliés par des opérateurs

- **Les Opérateurs Arithmétiques**

Ils permettent d'effectuer des opérations arithmétiques entre opérandes numériques (entier ou réel).
Les Opérateurs élémentaires : « + », « - », « * », « / » (division réelle), DIV (division entière)

Exemple :

(5 DIV 2 = 2)

Changement de signe : « - », « + »

Reste d'une division entière : « modulo » (ou « MOD » exemple 5 MOD 2 = 1)

N.B: les expressions arithmétiques sont évaluées de gauche vers la droite en respectant les priorités indiquées dans le tableau suivant:

Opérateurs	priorité
*, /, DIV, MOD	1
+ , -	2

- **Les opérateurs logiques (sur le type Booléen) :** qui combinent des opérandes booléens pour former des expressions logiques plus complexes:
  - ✓ Opérateurs unaire : la « négation » notée « NOT ».
  - ✓ Opérateurs binaires :la conjonction notée « AND », la disjonction notée « OR », la Disjonction Exclusive notée « XOR ».

Les tables de vérité des opérateurs logiques : Soit A et B deux variables de type booléen

A	B	NOT A	A AND B	A OR B	A XOR B
True	True	False	True	True	False
True	False	False	False	True	True
False	True	True	False	True	True
False	False	True	False	False	False

N.B

- ✓ La parenthèse des opérandes est nécessaire. Exemple : (A >= 0) and (A <= 10)
- ✓ Les opérations entre parenthèse sont prioritaires.
- ✓ L'évaluation des expressions logiques se fait de gauche vers la droite en respectant l'ordre de priorité décroissante des opérateurs logiques comme suite :
  - « NOT »
  - « AND »
  - « OR »
  - « XOR »

- **Les opérateurs de comparaison :** qui permettent de comparer deux opérandes et produisent une valeur booléenne.

Opérateurs en pascal	Opération
<	Inférieur
<=	Inférieur ou égale
>	Supérieur
=>	Supérieur ou égale
=	Egalité
<>	Différent (≠)

1.8 Les instructions de base d'un algorithme

Pour rédiger la partie corps d'un algorithme on a besoin des trois instructions élémentaires suivantes :

- ✓ L'affectation
- ✓ La lecture
- ✓ *l'écriture*
- ✓

1.8.1 La saisie (lecture des données)

Sa syntaxe est lire (indentificateur_de_la_variable). Cette instruction permettant de placer en mémoire les informations fournies par l'utilisateur pour qu'elles soient utilisées par le programme.

Exemple:

lire (a),

lire (d,c)

1.8.2 L'affectation

Cette instruction utilise le symbole flèche orientée gauche ←
Instruction permettant d'attribuer à la variable identifiée par l'élément placé à gauche du symbole d'affectation la valeur de l'élément placé à droite de ce symbole.

Exemple :

A ← 10 ;

B ← C ;

N.B : On peur incrémenter la valeur d'une même variable sans utiliser une deuxième variable

Exemple :

```
Algorithme affectation1
Variable A : Entier
Début
A ◄——— 15
A ◄——— A + 1
Fin
```

N.B L'affectation est également valable pour les caractères et les chaines de caractères

Exemple :

```
Algorithme  affectation2
Variable C : Caractère
S : Chaine de Caractères
Début
C ◄———  'A'
S ◄———  'Mardi'
S ◄——— C
Fin
```

1.8.3 L'Affichage

Sa syntaxe est écrire (variable, constante, expression). Elle permet l'affichage à l'écran des valeurs des variables après traitement.

Exemple:

```
ecrire (a);
ecrire ("bonjour");
ecrire ("le résultat est", b);
```

Exemple : qui contient les trois instructions de base : Le carré d'un nombre

```
Algorithme calculcarre;
Variables nb,carre: réels
Début
Ecrire('entrez un nombre')
lire (nb)
carre← nb*nb
écrire ( carre)
Fin
```

Exercice 1: Quelles seront les valeurs des variables A et B après exécution des instructions suivantes

Exercice 2 :
Quelles seront les valeurs des variables A, B, et C après exécution des instructions suivantes

Exercice 3: Quelles seront les valeurs des variables A et B après exécution des instructions suivantes ?

Algorithme Exercice 1
Variable A, B : Entier
Début
A ← 5
B ← A + 4
A ← A + 1
B ← A - 4
Fin

Exercice 4: écrire un algorithme permettant d'échanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.

Chapitre 2 :
Le traitement conditionnel

2.1 Définition (traitement conditionnel)

Un traitement conditionnel permet d'exécuter des instructions que sous condition et est exprimé à l'aide de l'instruction SI .... Alors.

- **Forme 1:**

```
Si expression booléenne Alors
   Instructions
FinSi
```

- **Forme 2:**

```
Si expression booléenne Alors
    Instruction 1
Sinon
     Instruction 2
FinSi
```

2.2 Expression booléenne

Une expression booléenne peut être VRAIE ou FAUSSE et peut être soit une variable booléenne soit une condition.

N.B

1. Dans la forme 1 : si l'expression booléenne est VRAIE les instructions se situant entre les mots clés SI et FinSi seront exécutées.

2. Dans la forme 2 : si l'expression booléenne est VRAIE les instructions se situant entre les mots clés SI et Sinon seront exécutées. Si elle est FAUSSE alors c'est l'instruction entre Sinon et FinSI qui seront exécutés (Jamais les deux ensemble)

2.3 Variable booléenne

Une variable booléenne peut prendre la valeur VRAIE ou FAUX en utilisant l'opération d'affectation.

Exemple :

```
Algorithme Exemple
Variable A : Bool_een
Début
A   VRAI
SI A ALORS
_Ecrire "Valeur vraie"
FinSi
Fin
```

2.4 Une condition

Une condition est la comparaison de deux valeurs du même type à l'aide d'un opérateur de comparaison.

Opérateur	Notation
Égal à	$=$
Différent de	$\neq$
Strictement inférieur	$<$
Strictement supérieur	$>$
Inférieur ou égal	$\geq$
Supérieur ou égal	$\leq$

Exemple 1:

```
Algorithme Exemple
Variable A : Booléen
Début
 A   VRAI
SI A = VRAIE ALORS
   Ecrire "Valeur vraie"
FinSi
Fin
```

Exemple 2:

```
Algorithme Positif ou négatif
Variable A : NOMBRE
Début
LIRE A
SI A > O ALORS
Ecrire "Nombre positif"
Sinon
Ecrire "Nombre n_egatif"
FinSi
Fin
```

Une condition composée utilise deux ou plusieurs conditions liées par des opérateurs logiques (ET, OU, XOR, NON).

Exemple 3:

```
Algorithme Exemple
Variable A : Booléen
Début
LIRE A
SI A > 0 ET A < 2
Ecrire "LA valeur de A est 1"
FinSi
Fin
```

Exercice: Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (le cas où le produit est nul n'est pas traité).

Chapitre 3 :
Les boucles

3.1 Définition

Le traitement itératif (les boucles) permet d'exécuter des instructions d'une façon répétitive sous condition.
Les conditions peuvent être exprimées à l'aide des instructions:

- Tantque .... faire
- Pour ... faire

3.2 La boucle : TantQue ... Faire

a) Syntaxe de la boucle TantQue ... Faire

```
TantQue Condition est vraie Faire
   Instruction 1
   Instruction 2
    ...
FinTantQue
```

b) Le principe de la boucle TantQue est le suivant:

- Dans la ligne TantQue, si la condition est vraie alors on exécute les instructions entre TantQue et FinTantQue ensuite on retourne à la ligne TantQue.
- Si la condition est fausse on saute toutes les instructions entre TantQue et FinTantQue et sort de la boucle

Exemple : un algorithme qui demande un nombre inférieur à 10 et qui affiche un message "entrée correcte" à chaque fois que l'utilisateur saisi un nombre inférieur à 10, dans le cas contraire il affiche le message "entrée erronée" et termine l'exécution.

```
Algorithme Exercice 1
Variable N : Entier
Début
N   0
TantQue N < 10 Faire
Ecrire "Saisir un nombre inférieur _a 10"
Lire N
Si N < 10 Alors
Ecrire "Entrée Correcte"
```

```
Sinon
Ecrire "Entrée erronée"
FinSi
FinTantQue
Fin
```

3.3 La boucle : Pour…faire

a). Définition

La boucle Pour...Faire est une variante de la boucle TantQue...faire et est utilisée lorsqu'on connait au préalable le nombre d'itérations que va effectuer la boucle.

b). Syntaxe de la boucle Pour…Faire

```
Pour  Compteur   Valeur initiale à Valeur finale Faire
Instruction 1
Instruction 2
FinPour
```

N.B
On peut définir le pas d'incrémentation du compteur (2, 3, 10,...etc) en ajoutant le mot clé Pas avant le mot clé Faire. Une valeur du pas égale à 2 veut dire que compteur va s'incrémenter de 2 à chaque itération de la boucle. S'il n'est pas spécifié il prendra par défaut la valeur 1.

c).Syntaxe de la boucle Pour…Faire avec un pas différent de 1

```
Pour Compteur   Valeur initiale à Valeur finale Pas = 2 Faire
Instruction 1
Instruction 2
FinPour
```

Exercice 1
Ecrire un algorithme à l'aide la boucle Pour...Faire ensuite à l'aide de la boucle TantQue...Faire qui affiche les nombres de 1 jusqu'à 100. Quelle est la différence entre les deux solutions ?

Exercice 2 :
Ecrire un algorithme à l'aide de la boucle Pour...Faire qui affiche les nombres pairs inferieurs à 100.

Exercice 3 :

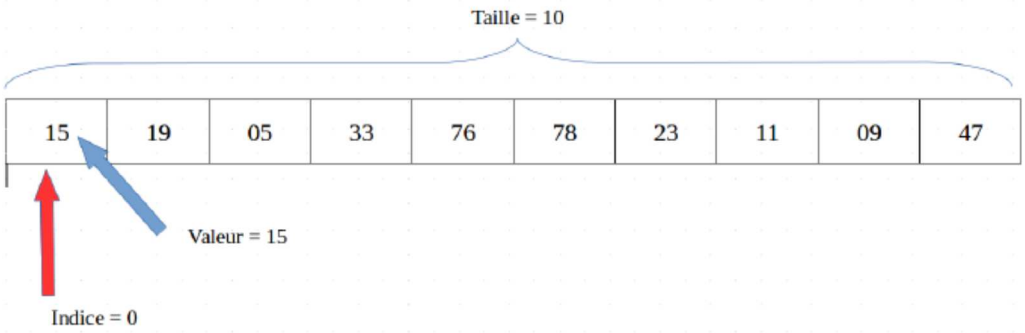
Ecrire un algorithme qui demande à l'utilisateur un mot de passe, si l'utilisateur saisit le mot de passe correcte qui est "informatique" le message

"Mot de passe correcte, Bienvenu !" sera affiché. Si l'utilisateur saisit un mot de passe erroné alors le message "Mot de passe incorrecte réessayer !" si l'utilisateur saisit trois fois un mot de passe erroné alors l'algorithme arrête l'exécution.

Chapitre 4 :
Les tableaux

4.1 Définition

Un tableau est une structure de données contenant un ensemble de valeurs qui ont le même type et qui sont repérées par un nombre (indice).



4.2 Déclaration et Accès aux éléments

a).Déclaration

On déclare un tableau dans la section variable en utilisant le mot clé Tableau.

b). Accès aux éléments

On accède à un élément d'un tableau en écrivant le nom du tableau suivit de l'indice entre crochets.

Exemple 1 :

```
Algorithme premier tableau
Variable tab : Tableau [10] d'entiers
Début
tab [0] <- 15
tab [0] <- 19
Fin
```

N.B

L'élément d'un tableau est traité comme une seule variable et peut donc être remplie en utilisant l'affectation ou une opération de lecture et son contenu afficher à l'aide de l'instruction Ecrire.

Exemple 2 :

```
Algorithme premier tableau
Variable tab : Tableau [10] d'entiers
Début
Lire (tab [0])
Lire (tab [1])
Ecrire (tab [0])
Ecrire (tab [1])
Fin
```

Les tableaux sont utilisés pour traiter un nombre important de variables. De ce fait, on utilise des boucles pour remplir ou afficher le contenu des tableaux.

Exemple 3 :

```
Algorithme premier tableau
Variable tab : Tableau [10] d'entiers
indice : entier
Début
Pour indice   0 ↕a 9 faire
Lire (tab [indice])
FinPour
Fin
```

N.B L'indice d'un tableau doit :
Commence par 0.
Doit être un nombre entier.
Doit être inférieure à la taille du tableau.

Exercice 1: Ecrire un algorithme qui déclare et remplisse un tableau de 7 valeurs numériques en les mettant toutes à zéro.

Exercice 2: Ecrire un algorithme qui déclare un tableau de 9 notes, dont on fait ensuite saisir les valeurs par l'utilisateur.

Exercice 3: Ecrire un algorithme qui permet d'identifier le nombre les plus grands parmi N nombres stockées dans un tableau.

Exercice 4 : Ecrire un algorithme qui permet de trier les éléments d'un tableau contenant N entiers dans un ordre croissant ou décroissant.