Université Amar Telidji Laghouat

Département d'Informatique

Cours du module : Algorithmique et Structure de Donneés

1ère Année L.I

Tahar ALLAOUI
t.allaoui@lagh-univ.dz

Merci de me signaler les erreurs qui peuvent exister dans ce document

Sommaire

Introduction

L'algorithmique, branche fondamentale de l'informatique, est au cœur de nombreuses disciplines scientifiques et applications technologiques. Elle étudie les méthodes et processus permettant de résoudre des problèmes de manière systématique et efficace à l'aide d'algorithmes. Un algorithme, en effet, constitue un ensemble d'instructions précises et ordonnées visant à accomplir une tâche spécifique ou à résoudre un problème donné, que ce soit dans les domaines des mathématiques, de l'intelligence artificielle, des systèmes embarqués, ou encore des réseaux de communication.

Le présent document propose un résumé structuré des principaux concepts abordés lors du cours d'algorithmique, destiné à fournir une vue d'ensemble des éléments essentiels à maîtriser pour résoudre des problèmes informatiques complexes. L'algorithmique, en tant que discipline, s'appuie sur des outils mathématiques pour formuler des solutions qui sont ensuite traduites en algorithmes, puis implémentées dans un langage de programmation. Dans cette perspective, une bonne compréhension des bases théoriques de l'algorithmique est indispensable pour aborder avec succès des problèmes pratiques et pour développer des logiciels efficaces.

Au fil du cours, plusieurs thèmes fondamentaux ont été explorés, chacun contribuant à la construction d'une méthode algorithmique rigoureuse et optimale. Parmi ces thèmes, on retrouve les structures de données simples, les instructions d'entrées/sorties, les instructions conditionnelles, les boucles, les tableaux, etc.. Chacun de ces concepts représente un outil clé permettant de résoudre une large gamme de problèmes, qu'ils soient théoriques ou appliqués. Ce document a pour but de résumer ces notions tout en mettant l'accent sur les aspects pratiques de leur mise en œuvre.

Une des premières notions abordées dans le cadre de ce cours est celle des structures de données qui occupent une place primordiale dans l'étude de l'algorithmique. Elles constituent les fondations sur lesquelles les algorithmes se construisent. Les structures les plus courantes, comme les variables simples ou complexes telles que les tableaux, les listes chaînées, les piles et les files d'attente, permettent de stocker et d'organiser les données de manière à faciliter l'accès, la modification et le traitement de ces dernières. Une maîtrise approfondie des structures de données est essentielle, car elle influence directement l'efficacité des algorithmes. Par exemple, les algorithmes de tri ou de recherche, qui sont étudiés en profondeur, reposent

souvent sur l'utilisation de structures de données spécifiques pour atteindre des performances optimales. Le cours aborde aussi les instructions conditionnelles qui constituent une partie majeure de l'algorithmique appliquée. Une autre partie clé du programme d'algorithmique est celle de la répétition des instructions qui peut être mise en place par le biais des boucles. Aussi, les structures de données dynamiques et les sous programmes feront l'objet de ce document.

Le but de ce document est donc de récapituler ces concepts et d'offrir un guide structuré permettant aux étudiants de réviser les points essentiels du cours d'algorithmique. À travers des exemples pratiques et des démonstrations, nous souhaitons que chaque étudiant soit en mesure de non seulement comprendre les principes théoriques sous-jacents, mais aussi d'appliquer ces principes à des situations concrètes. L'algorithmique est une discipline exigeante, mais elle constitue également un domaine fascinant, avec des applications potentielles illimitées. En maîtrisant les bases de l'algorithmique, les étudiants se préparent à résoudre des problèmes complexes dans le monde réel et à contribuer à l'innovation technologique.

Chapitre 1 :
Notions de base

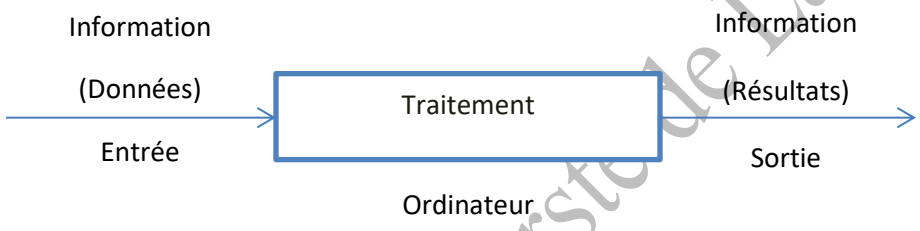
1. Introduction

Actuellement, l'informatique touche tous les domaines de notre vie grâce à ces avantages offerts, l'informatique a permis de rendre facile et rapide la réalisation des tâches difficiles et même complexes.

2. Définitions

2.1. Informatique

Le terme informatique est une combinaison de deux mots: information et automatique, c'est la science du traitement automatique de l'information par une machine "Ordinateur"



2.2. Ordinateur

L'ordinateur est une machine électronique permettant de résoudre des problèmes par l'exécution des instructions, c'est un ensemble de périphériques qui communiquent entre eux, pour réaliser un travail donné.

2.3. Utilité de l'informatique

On fait appel à l'informatique afin de résoudre des problèmes de natures différentes en un temps réduit.

Comment?

Une question importante est posée: comment expliquer à l'ordinateur la méthode de résolution d'un problème donné?

Pour répondre à cette question, on donne des exemples qui représente la méthode utilisée pour résoudre des problèmes simples de notre vie quotidienne.

Exemple 1 : Envoyer un SMS vers un ami:

Pour envoyer un SMS, l'utilisateur doit appliquer les actions suivantes:

1. Cliquer sur « Menu »
2. Aller à « Messagerie »

3. Choisir « Nouveau message »
4. Editer le message
5. Ecrire le numéro de destinataire
6. Cliquer sur le bouton « Envoyer »

Cet exemple montre la démarche à suivre pour envoyer un SMS, il est composé d'une suite ordonnée d'actions qui sont représentées sous forme de commandes ou instructions (*Cliquer, Donner, Editer, Choisir,...*) qui manipulent des données (*Menu, Messagerie, Numéro de destinataire,...*) afin de réaliser le travail désiré (*L'envoi d'un SMS*)

Exemple 2 : Calculer les solutions d'une équation de 2ème degré:

Pour résoudre ce problème, on applique les étapes suivantes:

1. Donner les valeurs de : *a, b,* et *c*.
2. Calculer la valeur de *Delta*
3. Calculer les valeurs des solutions selon la valeur de *Delta*.

De la même façon, dans cet exemple, on manipule des données (*a, b, c*) par une suite ordonnée d'instructions (*Calculer Delta, Calculer les solutions,...*) afin de trouver les solutions de l'équation.

On remarque bien que la résolution d'un problème nécessite l'application d'une suite d'actions sur un ensemble de données dans un ordre bien défini pour atteindre le résultat.

De la même façon, pour qu'un ordinateur puisse résoudre un problème, on doit lui "expliquer" les actions à appliquer et les données à manipuler. Cette explication qui représente la solution du problème à résoudre sera présentée sous forme d'actions (appelée aussi commandes ou instructions) qui manipulent un ensemble de données, l'ensemble des instructions et des données manipulées forme **un algorithme.**

3. Définition d'un algorithme

Un algorithme est une suite ordonnée d'instructions (actions, commandes) qui indiquent la démarche à suivre pour résoudre un problème donné.

Dans un algorithmes, l'ordre des instructions est important et a une grande influence sur le résultat obtenu.

Le mot **Algorithme** est d'origine arabe, il vient du nom de : **Al Khawarizmi (780-850)**, le savant musulman qui est considéré comme le père de l'algèbre.

4. Algorithmique

C'est la science des algorithmes, c'est un ensemble de règles et techniques utilisées dans la conception et la définition des algorithmes.

5. Langage de description des algorithmes

C'est un langage universel très proche du langage naturel, et qui permet d'écrire des algorithmes lisibles et compréhensible par les utilisateurs. C'est un langage riche et libre de la machine, qui offre la possibilité de correction et de modification. Ce langage offre également la possibilité de représentation graphique (Organigramme)

6. Organigramme

Un organigramme (ou algorigramme) est une représentation graphique d'un algorithme qui permet de faciliter la compréhension de l'algorithme.

À chaque type d'instructions est associée une forme graphique particulière.

Remarque : *C'est vrai qu'un algorithme représente une abstraction de la solution d'un problème donné, mais cette solution n'est pas compréhensible par la machine (le langage algorithmique est très proche du langage naturel). Pour que l'algorithme soit compréhensible par la machine, on doit écrire un programme équivalent à cet algorithme.*

7. Programme

C'est la traduction d'un algorithme dans un langage de programmation particulier.

L'ensemble d'instructions d'un programme est appelé code source.

L'exécution d'un programme (code source) par l'ordinateur produit un programme exécutable (le code cible ou le programme objet), ce programme exécutable manipule les différentes données du problèmes pour produire les résultats cherchés.

8. Programmation

La programmation est l'ensemble des activités permettant l'écriture des programmes, c'est l'activité de rédaction du code source d'un programme.

9. Langage de programmation

C'est un ensemble de vocabulaires et de règles bien définies permettant à l'être humain d'écrire des programmes qui seront exécutés par l'ordinateur.

10. Les types des langages de programmation

10.1. Langage machine

Appelé aussi code machine, c'est un langage de niveau 0 compréhensible par la machine, et dont la forme est en binaire.

10.2. Langage assembleur

Appelé aussi *langage d'assemblage* ou tout simplement *assembleur*, c'est un langage de bas niveau qui représente le langage machine sous une forme lisible par l'humain.

10.3. Les langages évolués

Appelés aussi les langages de haut niveau, ce sont des langages avec un vocabulaire riche, facile à comprendre et à maintenir avec des règles d'écriture bien déterminées.

11. Représentation interne de l'information

Le seul langage compréhensible par l'ordinateur est le langage machine, donc, les données manipulées par les utilisateurs sont représentées dans la machine d'une autre façon, elles sont représentées par une très longue suite codée en binaire.

11.1. Bit

Un **bit** (***bi****nary digi****t***) est un chiffre binaire : 0 ou 1, il représente l'unité élémentaire de l'information dans les ordinateurs.

11.2. Octet (Byte):

Une suite de **8 bits** qui représente une unité de mesure de quantité de données (1 Octet = 8 Bits)

Les multiples d'octet

Nom	**Symbole**	**valeur**
Kilo-octet	Ko	2^{10}=1024 octets
Méga-octet	Mo	2^{20}=1024 Ko

Giga-octet	Go	2^{30}=1024 Mo

Série de TD 01

Exercice 1

Donner une définition pour les termes suivants:

Algorithme, Organigramme, Programme, Bit, Byte, Octet.

Exercice 2

1. Donner les valeurs binaires des valeurs décimales suivantes :

   21 , 79 , 63 , 511 , 514

2. Donner les valeurs décimales correspondantes aux valeurs binaires suivantes :

   11 , 1011 , 111101100 , 1111000010101 , 11010111

Exercice 3

Parmi les valeurs suivantes, lesquelles sont numériques et lesquelles sont non-numériques ?

-53.6 , ‘Quatre’ , ‘52’ , 52 , ‘TP’ , ‘T.D’ , +59 , ‘Valeur numérique’ , ‘Valeur non-numérique’

Exercice 4

Relever les erreurs si elles existent

41.8.8 , ‘Dimache’ , foux , ‘foux’ , ‘jus’te’ , +.13 , -86.9 , 1X , ‘2Y’

Exercice 5

Donner le résultat et l'ordre d'évaluation des expressions suivantes :

- (-5 * 3) + (5 – 2) - (4 ↑2)
- 9 * ((12 – 3) / (1 + 2)) ↑ (2 + 2)
- (-3 * 2) + (5 – 9) - (2 ↑3)
- -3 * (2 + (5 – 9)) - (2*4)↑2
- (P et vrai) ou (non P et non vrai)
- (P ou non P) et (vrai et vrai) et (5 > 3)
- (3 > 2) et (21 = (19 + 2)) ou (8 < 10)

Exercice 6

Donner la négation des expressions logiques suivantes :

$X \geq 2$ et $X > 5$ $X \geq 2$ et $X < 5$ $X \leq 2$ ou $X < 5$

Chapitre 2 :
Les structures de données et les instructions de base

1. Introduction

Afin de résoudre les différents problèmes en informatique, on fait appel à des programmes compréhensibles et exécutables par la machine, mais avant de passer à la programmation, on doit d'abord représenter les solution sous forme des algorithmes.

C'est vrai que l'algorithme n'est qu'une représentation de la solution qui est écrite en langage universel, mais cet algorithme a une forme générale bien défini, et son écriture doit respecter des règles pour rendre facile le passage vers la programmation.

2. Structure générale d'un algorithme

Un algorithme a généralement la structure suivante :

Algorithme Nom de l'algorithme } L'en-tête
Les déclarations } La partie de déclaration
Début
  Instruction 1 ;
  Instruction 2 ;
  …
  Instruction n ;
Fin

Le corps de l'algorithme

- **L'en-tête** : chaque algorithme doit avoir un nom qui l'identifier dans l'en-tête.
- **Les déclarations** : dans un algorithme, on peut manipuler des données d'un problème posé pour obtenir des résultats, la partie de déclaration contient la liste des données utilisées et manipulées dans l'algorithme ainsi que leurs natures.
- **Le corps de l'algorithme** : c'est la partie qui représente le travail réalisé par l'algorithme, toutes les instructions et les opérations exécutées par l'algorithme se trouvent dans cette partie.

3. L'en-tête

Cette partie contient le nom (l'identificateur) de l'algorithme, ce nom doit être une suite de lettres et de chiffres, qui commence obligatoirement par une lettre et qui ne contient par un espace.

4. La partie de déclaration

Les données utilisées dans l'algorithme doivent être définies (déclarées) pour faciliter leur manipulation, pour cela on a besoin d'utiliser les structures de données.

4.1. Structure de données

Définition : Une structure de données est une structure logique destinée à contenir des données, c'est un support logique qui permet de conserver une données.

Chaque structure de données est caractérisée par :

1. **Un identificateur** : c'est le nom de la structure de données, il est composé de lettres et de chiffres, ce nom doit respecter les mêmes règles du nom du l'algorithme, par exemple x1, y2, tab,…représentent des identificateurs des structures de données.
2. **Un type** : Chaque structure de données a un type qui indique la nature de l'informations stockées dans cette structure de données. On peut distinguer cinq types de base :
   L'entier : une suite de chiffres qui peut être précédée par un signe (+ ou -).
   Le réel : une suite de chiffres qui peut être précédé par un signe, et qui peut contenir un point décimal.
   Le booléen : Une donnée ayant seulement deux valeurs possibles : vrai ou faux.
   Le caractère : 'a'..'z', 'A'..'Z', '5', ' !', '?', '@'
   La chaîne de caractères : une concaténation de plusieurs caractères, par exemple : 'informatique ', 'salut !'
3. **La valeur d'une structure de données** : qui indique la valeur de la donnée qui se trouve dans la structure de données, pendant l'exécution d'un algorithme, cette valeur peut être **fixe** ou **variable**.

Remarque : *selon la valeur de la structure de données (**fixe** ou **variable**), on peut distinguer deux grandes familles des structures de données : les constants et les variables.*

4.2. Les constants

Un constant est une structure de données dont **la valeur est fixe** et ne change pas dans tout l'algorithme.

Déclaration des constants

Pour déclarer un constant, on utilise le mot clé **const** dans la partie de déclaration.

Exemple : const X=2 ;
Y= 'bon jour' ;

- X est un constant de type entier dont la valeur est 12.
- Y est un constant de type chaîne dont la valeur est : bon jour.

4.3. Les variables

Une variable est une structure de données dont **la valeur est variable** et peut être changée dans l'algorithme.

Déclaration des variables

Pour déclarer des variables, on utilise le mot clé **var** dans la partie de déclaration.

Exemple : var P1 : booléen ;

a, b, c : réel ;

- P1 est une variable de type booléen
- a, b, et c sont des variables de type réel.

Remarque : *On remarque bien que les valeurs des variables ne sont pas définies dans la partie de déclaration, la valeur d'une variable est donnée dans le corps de l'algorithme.*

5. Le corps de l'algorithme

Cette partie contient toutes les instructions et les opérations réalisées par l'algorithme. Dans le corps de l'algorithme on trouve les instructions qui donnent des valeurs aux différentes variables et d'autres instructions qui représentent la démarche à suivre pour résoudre le problème, il est important donc de respecter un ordre bien défini entre ces instructions afin d'atteindre la solution.

6. Les instructions de base

6.1. Les entrées-sorties

L'algorithme a besoin des données en entrée pour fournir des résultats en sortie, pour cela, on fait appel aux instructions d'entrée- sortie qui permettent aux utilisateurs d'introduire des informations et de recevoir d'autres.

La Lecture des données

Cette instruction permet à l'utilisateur de définir la valeur d'une ou de plusieurs variables, la valeur introduite par l'utilisateur sera stockée dans la variable concernée.

Il existe plusieurs forme pour l'instruction de lecture:

1. **Lire une seule donnée** : lire (nom de variable) ;

Exemple : lire(X)

Cette instruction permet d'attribuer la valeur donnée par l'utilisateur à la variable X, si l'utilisateur tape 10 par exemple, la variable X aura la valeur 10.

2. **Lire plusieurs données** : lire (variable1, varaible2, …)

Exemple : lire (X,Y, Z) ;

Cette instruction permet d'attribuer les valeurs données par l'utilisateur aux variables X, Y et Z en respectant cet ordre, si l'utilisateur donne 4, 9 et 11 dans cet ordre, alors X aura la valeur 4, Y aura la valeur 9 et Z aura la valeur 11.

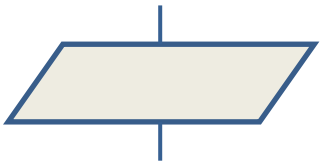
Remarque: *lors d'une opération de lecture, l'utilisateur doit donner une valeur ayant le même type avec la variable concernée, le cas contraire est une erreur.*

Ecriture de données

Cette instruction permet d'afficher des données sur l'écran, selon la nature de la donnée affichée on peut distinguer plusieurs formes:

1. **Afficher la valeur d'une variable** : écrire (nom de variable) ;
2. **Afficher un texte** : écrire ('Texte à afficher') ;
3. **Afficher des textes et des valeurs** : écrire ('Texte à afficher', variable, 'Texte à afficher', variable,…) ;

Remarque : Les instructions d'entrée-sortie sont représentées dans l'organigramme par le symbole suivant :



6.2. Les expressions

Les variables et les constants sont manipulés dans les algorithmes à l'aide des expressions.

Définition : Une expression est une combinaison d'opérandes (variables ou constants) et des opérateurs. selon les types des opérandes et des opérateurs on peut distinguer deux grandes familles d'expressions à savoir les expressions arithmétiques et les expressions logiques.

Les expressions arithmétiques

C'est une expression dont les opérandes sont numériques (entiers ou réels) et les opérateurs sont arithmétiques.

Les opérateurs arithmétiques	
↑	La puissance
*	La multiplication
/	La division
+	L'addition
-	La soustraction
DIV	La division entière (euclidienne)
MOD	Le reste de la division euclidienne.

Exemples :

5 + 3 – 2 ↑ 3

(2 * 6) + (5 / 3) * 9

2 * (6 + 5) / 3 * 9

L'ordre d'évaluation

- Certains opérateurs sont plus prioritaires que les autres, pour évaluer une expression arithmétique, on doit suivre l'ordre suivant :

1. La puissance.
2. La multiplication et la division
3. L'addition et la soustraction

- Si on a une expression qui contient des opérateurs de la même priorité, on commence l'évaluation de gauche à droite.
- En cas de présence des parenthèses, on commence par l'évaluation des parenthèses.
- On commence par les parenthèses de gauche à droite si l'expression contient plusieurs parenthèses.

Les expressions logiques :

Une expression logique peut être une combinaison des opérandes logiques et des opérateurs logiques ou peut être une combinaison des opérandes numériques et des opérateurs de comparaison.

La 1ère forme: Opérandes numériques et opérateurs de comparaison.

Les opérateurs de comparaison	
>	**Supérieur**
<	**Inférieur**
>=	**Supérieur ou égal**
<=	**Inférieur ou égal**
=	**égal**
≠	**Différent**

Exemple :

12 > 13

(5 + 2) < = 10

La 2ème forme: Opérandes logiques et opérateurs logiques.

Les opérateurs logiques	
NON	La négation
ET	La conjonction
OU	La disjonction

Exemple : NON (vrai ET faux)

Remarque : *le résultat d'une expression logique doit être toujours un booléen : vrai ou faux*

6.3. L'affectation

C'est une instruction qui permet d'affecter une valeur à une variable, autrement dit, l'instruction d'affectation permet de remplir le support logique par une valeur donnée par l'utilisateur ou calculée par une expression.

Les formes générales de l'instruction d'affectation

1. **1ère forme** : Var ← Valeur fixe, dans ce cas la variable prend la valeur.

Exemple : X ← 17 ; se lit : X reçoit 17

2. **2ème forme** : Var1 ← Var2

La variable Var1 prend la valeur de Var2 sans changer la valeur de Var2

Exemple : X ← Y

3. **3ème forme** : Var ← expression

La variable prend la valeur du résultat de l'expression après son évaluation.

Exemples

- nbr ←3+5-4 (la variable nbr reçoit la valeur 4)
- P1← vrai et faux (la variable P1 reçoit la valeur faux)

Remarques :

1. *Dans cette forme, la variable et le résultat de l'expression doivent avoir le même type.*
2. *Dans un algorithme, on peut affecter des valeurs à une variable plusieurs fois, la nouvelle affectation va écraser l'ancienne valeur.*

7. Exercice d'application

1. Ecrire un algorithme qui permet d'afficher la somme de deux valeurs entières données par l'utilisateur.
2. Donner l'organigramme correspondant à cet algorithme

7.1. L'analyse

3. Dans cet algorithme, l'utilisateur va donner deux valeurs entières, on a besoin donc de deux variables de types entier afin de stocker les valeurs.
4. On doit utiliser également une troisième variable pour stocker le résultat de l'addition.

5. Le résultat doit être affiché.

5.2. L'algorithme

Dans cet algorithme, on utilise une variable **nbr1** pour la 1ère valeur, et une variable **nbr2** pour la 2ème valeur. On utilise également la variable **result** pour le résultat

L'algorithme sera donc :

```
algorithme Somme ;
var   nbr1, nbr2, result : entier ;
début
    lire (nbr1, nbr2) ;
    result ← nbr1 + nbr2 ;
   écrire (result) ;
fin
```

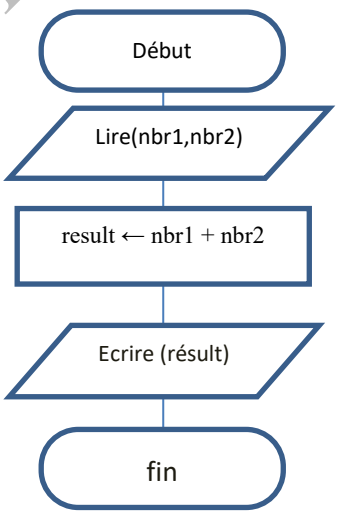
5.3. L'organigramme

Un organigramme est une représentation graphique d'un algorithme permettant de faciliter sa compréhension.

Remarques :

1. Les mots clés début et fin sont représentés par des rectangles à coins arrondis.
2. Les expressions et les affectations sont représentées par des rectangles.

L'organigramme correspondant à l'algorithme précédent sera donc :



Série de TD 02

Exercice 1

Donner toutes les formes possibles des instructions suivantes :

- La lecture.
- L'écriture.
- L'affectation.

Exercice 2

1. Ecrire un algorithme permettant de calculer la longueur et la surface d'une trapèze .
2. Ecrire un algorithme permettant de calculer la longueur et la surface d'un carré.
3. Donner les organigrammes correspondants à ces algorithmes.

Exercice 3

1. Ecrire un algorithme qui affiche la moitié, le double, le triple et le carré d'un nombre donné par l'utilisateur.

   Exemple : si l'utilisateur donne la valeur 5, l'algorithme doit afficher :

   La moitié de 5 est 2.5 ;

   Le double de 5 est 10 ;

   Le triple de 5 est 15 ;

   Le carré de 5 est 25 ;

2. Donner l'organigramme correspondant à cet algorithme.
3. Ecrire l'algorithme précédent en utilisant deux variables.
4. Peut-on écrire cet algorithme en utilisant une seule variable? Si oui, Comment?

Exercice 4

1. Ecrire un algorithme qui échange la valeur de deux variables données par l'utilisateur.

Exemple : Si a=5 et b=8, l'algorithme doit donner a=8 et b=5.

2. Donner l'organigramme correspondant à cet algorithme.

Exercice 5

Etant donnés les algorithmes suivants.

1. Préciser et corriger les erreurs qui existent dans les algorithmes.
2. Expliquer ce que fait chaque algorithme.
3. Donner les organigrammes correspondants.

```
algorithme valeur
var: X entier
début:
    écrire: Donner la valeur de X
    lire X
    écrire (la valeur qui suit X est X+1);
fin;
```

```
algorithme calcul:
var x; y, s, p: entiers;
début
     écrire( Donner les deux valeurs );
     lire (x, y);
    s = x+y ;
    p= x * y;
    écrire ('La somme de X et Y est, P);
    écrire ('Le produit de X et Y est; s);
fin.
```

```
algorithme moyenne;
var: X1, X2, X3: entier;
      M: réel;
début:
     lire(X1, X2, X3);
     M= X1+X2+X3/3;
     écrire( 'La moyenne est', 'M');
fin;
```

Chapitre 3 :
L'instruction Conditionnelle

1. Introduction

Jusqu'à maintenant, nous avons vu des algorithmes qui suivent un seul chemin pour résoudre un problème, c'est-à-dire, les algorithmes exécutent séquentiellement des instructions sans rupture, cette structure d'algorithmes est appelée **la structure linéaire**.

Dans ce chapitre, nous allons découvrir une nouvelle structure qui permet d'exécuter des instructions selon un choix, la structure de l'algorithme dans ce cas devient **une structure alternative**.

Pour comprendre l'utilité de cette structure, on donne les exemples suivants.

Exemple 1

Ecrire un algorithme qui calcule la moyenne d'un étudiant ayant trois modules

La résolution de ce problème est simple, on doit lire les notes des modules, ensuite calculer la moyenne de ces notes, l'algorithme correspondant sera donc :

```
Algorithme moyenne
Var n1, n2, n3, moy :réel ;
Début
  Lire (n1, n2, n3) ;
 moy← (n1+n2+n3)/3 ;
 écrire (moy) ;
Fin
```

Exemple 2

Écrire un algorithme qui calcule la moyenne d'un étudiant, et affiche « admis » dans le cas où la moyenne de cet étudiant est supérieure ou égale à 10

On remarque dans ce problème que l'affichage de « admis » dépend de la valeur de moyenne, l'exécution de l'instruction d'affichage n'est possible qu'après la vérification d'une condition.

Pour cela, on doit utiliser une nouvelle instruction **: l'instruction conditionnelle**

2. Instruction conditionnelle

C'est une instruction alternative, qui consiste à exécuter une ou plusieurs instructions après la vérification d'une condition.

La condition doit être une expression logique, c'est la raison pour laquelle on appelle cette instruction **le test logique**

3. Les différentes formes de l'instruction conditionnelle

3.1. La 1ère forme : L'instruction conditionnelle réduite

Si condition alors

Actions ;

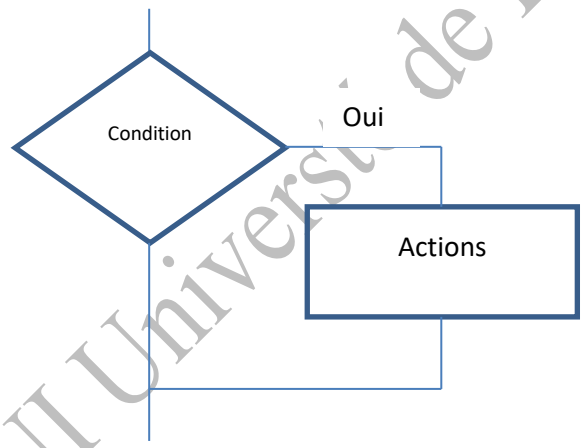
Fin si

L'exécution

Si la condition est vérifiée (sa valeur est vraie) on exécute les actions, ensuite on continue l'exécution de l'algorithme. Dans le cas contraire (la valeur de la condition est fausse) on exécute directement les instructions qui viennent après la "fin si" sans exécuter les actions.

L'organigramme

Dans les organigrammes, l'instruction conditionnelle peut être représentée par la forme suivante



L'algorithme correspondant à l'exemple précédent

```
Algorithme moyenne
Var n1, n2, n3, moy :réel ;
Début
 Lire (n1, n2, n3) ;
 moy←(n1+n2+n3)/3 ;
 Si moy>=10 alors
    écrire ('Admis') ;
 fin si
 écrire(moy) ;
Fin.
```

Exemple 3

Écrire un algorithme qui affiche **Admis** dans le cas où la moyenne est supérieure ou égale à 10, et affiche **Ajourné** dans le cas contraire.

Dans ce cas, l'instruction conditionnelle réduite ne permet pas de résoudre le problème car nous avons deux possibilités, on doit utiliser donc une nouvelle forme de l'instruction conditionnelle

3.2. La 2ème forme : L'instruction conditionnelle complète

```
Si condition alors
   Action 1
Sinon
   Action2 ;
Fin si
```

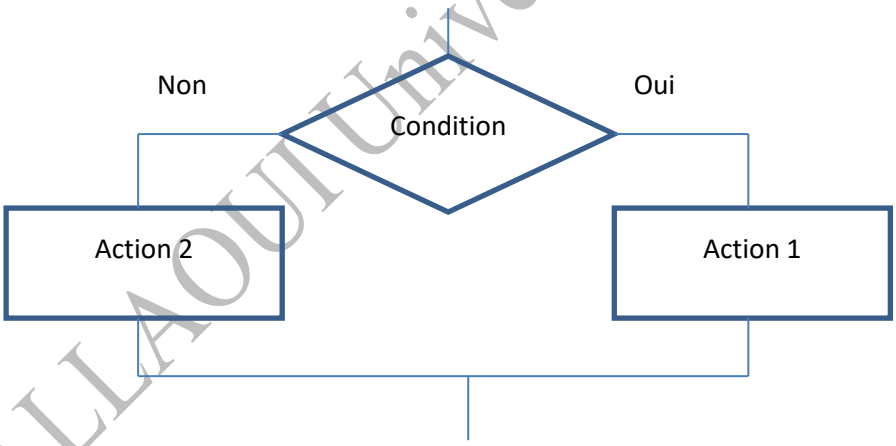
L'exécution

Si la condition est vérifiée, seule l'action1 est exécutée, ensuite, on va sauter l'action2 pour exécuter le reste de l'algorithme après la "fin si".

Si la condition n'est pas vérifiée, l'action1 sera ignorée, l'action2 seulement sera exécutée.

L'organigramme

L'instruction conditionnelle complète est représentée dans les organigrammes par la forme suivante



L'algorithme correspondant à l'exemple 3

```
Algorithme moyenne
Var n1, n2, n3, moy : réel ;
Début
  Lire (n1, n2, n3) ;
  moy← (n1+n2+n3)/3 ;
  écrire (moy) ;
  si ( moy>=10) alors
     écrire ('Admis')
     sinon
```

```
      écrire ('Ajournée') ;
    fin si
  Fin.
```

Exemple 4

Ecrire un algorithme qui lit un nombre donné par l'utilisateur, ensuite affiche 'positif', 'négatif' ou 'nul' selon la valeur de ce nombre.

Il est clair qu'on doit utiliser une instruction conditionnelle pour résoudre ce problème, mais dans l'instruction conditionnelle complète la condition peut avoir deux valeurs seulement, c'est-à-dire, on peut traiter deux cas seulement, mais dans ce problème, on a 3 cas à traiter, pour cela, on doit utiliser plusieurs instructions conditionnelles

3.3. Les instructions conditionnelles imbriquées

La forme générale

```
Si condition1 alors
    Action 1 ;
   Sinon
   si condition 2 alors
      Action2 ;
     Sinon action3
   Fin si
Fin si
```

Remarque : chaque **fin si** correspond au **Si** le plus proche

L'algorithme de l'exemple 4

```
Algorithme nombre
Var   nbr : entier ;
Début
   Lire (nbr) ;
   Si (nbr>0) alors
     Ecrire ('positif')
    sinon
     Si nbr<0 alors
         écrire('négatif')
         sinon écrire('nul') ;
     fin si
   fin si
Fin.
```

3.4. L'instruction de choix

Dans certains algorithmes, on peut trouver plusieurs instructions imbriquées ce qui rend l'algorithme difficile à comprendre et difficile à manipuler.

Pour éviter ça on utilise une autre forme de l'instruction conditionnelle qui est l'instruction de choix.

Cette instruction consiste à vérifier une donnée ordinale (entier, caractère, booléen), et exécuter des actions selon les valeurs possibles de cette donnée.

La forme générale de l'instruction de choix

```
Cas de variable parmi
    Valeur 1 : Action 1 ;
    Valeur 2 : Action 2 ;
    Valeur3, valeur 4 : Action3 ;
    Valeur 5..valeur6 : Action4 ;
    …
    Sinon Action n
Fin Cas ;
```

L'exécution

On compare la variable avec la valeur 1, s'il y a une égalité, on exécute l'action 1, et on sort de l'instruction, sinon, on compare la variable avec la valeur 2, on exécute l'action 2 s'il y a une égalité et on sort, sinon, on compare avec valeur3, et ainsi de suite, si la valeur de variable est différente de toutes les valeurs, on exécute donc l'action n.

Remarques

- La forme *Valeur1, Valeur2, Valeur3* **:** *Action* : Signifie que plusieurs valeurs différentes peuvent entrainer l'exécution de la même action.
- La forme *Valeur1* **..** *Valeur2 : Action* : Signifie que l'action doit être exécutée avec toutes les valeurs de cette intervalle.

Exemple

Ecrire un algorithme qui affiche le nombre de jours d'un mois donné par l'utilisateur

```
Algorithme mois ;
Var m :entier ;
Début
```

```
Lire(m) ;
Cas de m parmi
   1, 3, 5, 7, 8, 10, 12 : écrire('le nombre de jours est 31') ;
   4, 6, 9, 11 : écrire('Le nombre de jours est 30') ;
   2 : écrire ('le nombre de jours est 28') ;
   Sinon écrire ('Erreur') ;
Fin Cas ;
Fin.
```

Remarque *: Dans cet algorithme nous avons supposé que le nombre de jours de février est toujours 28.*

Série de TD 03

Remarque : *On vous demande de donner l'organigramme correspondant à chaque algorithme de cette série.*

Exercice1

1. Quelles sont les formes possibles de l'instruction conditionnelle ?
2. Quelles sont les formes possibles de l'instruction de choix **Cas**?
3. À votre avis, quelle est la meilleure instruction ?

Exercice 2

4. Ecrire un algorithme qui vérifie si un nombre donné par l'utilisateur est paire ou impaire.
5. Ecrire un algorithme qui demande un nombre à l'utilisateur, ensuite affiche la racine carrée de ce nombre (N'oubliez pas qu'un nombre négatif n'a pas une racine carrée réelle).
6. Ecrire l'algorithme qui demande une valeur à l'utilisateur, affiche positive ou négative selon la valeur et affiche également sa valeur absolue.
7. Ecrire l'algorithme qui affiche la valeur positive suivante si la valeur donnée par l'utilisateur est une valeur positive, et la valeur négative précédente si la valeur donnée par l'utilisateur est une valeur négative.

Exercice 3

Ecrire l'algorithme qui permet de résoudre une équation de deuxième degré tel que A, B et C sont des nombres réels.

Exercice 4

Ecrire un algorithme qui détermine la mention d'un étudiant en fonction de sa moyenne :

Moyenne < 10 : Mention : Ajourné

10 ≤ Moyenne < 12 : Mention : Passable

12 ≤ Moyenne < 14 : Mention : Assez bien

14 ≤ Moyenne < 16 : Mention : Bien

16 ≤ Moyenne : Mention : Très bien

Exercice 5

Dans le département d'informatique, nous avons 5 salles d'examens : S1, S2, S3 ,S4 et S5, de 40 places chacune. Le nombre des étudiants de la 1ère année est de 180, et chaque étudiant a un numéro.

1. Ecrire un algorithme qui demande le numéro d'un étudiant pour l'orienter à une salle d'examen.
2. Ecrire le même algorithme en utilisant l'instruction « **Cas** ».

Exercice 6

1. Ecrire un algorithme qui convertit le format du temps d'une horloge électronique du format de l'heure avant midi (AM) et après midi (PM) au format de 24 heures.
2. Ecrire un algorithme qui à partir d'une date donnée affiche le nombre de jours restants avant la fin du mois.
3. Ecrire un algorithme qui demande à l'utilisateur le numéro d'un mois pour afficher le nombre de jours du mois suivant.
4. Ecrire un algorithme qui demande à l'utilisateur une date pour afficher la date du jour suivant.
5. Ecrire un algorithme qui demande à l'utilisateur une date pour afficher la date du jour précédent.
6. Ecrire les algorithmes précédents en utilisant l'instruction **Cas**.

Exercice 7

Un laboratoire médical fait des analyses pour les patients. Le résultat de chaque analyse est obtenu après 15 jours de la date d'analyse.

1. Ecrire un algorithme qui demande à l'utilisateur la date de l'analyse (jour, mois, année) pour lui donner la date de résultat.
2. Ecrire l'algorithme précédent en utilisant l'instruction **Cas**.

Exercice 8

Ecrire un algorithme qui affiche la conjonction et la disjonction de deux variables booléennes (On doit donner à l'utilisateur la possibilité de définir les valeurs de ces variables).

Exercice 9

Un étudiant prépare ses examens finaux qui vont commencer à une date donnée, et il veut connaitre à n'importe quel jour le nombre de jours restants avant cette date importante.

- Ecrire l'algorithme qui demande la date à l'utilisateur pour afficher le nombre de jours restants avant le début des examens.

Exercice 10

Dans une gare routière d'une ville, on trouve l'affichage suivant :

Distance (en Km)	Tarifs (Pour le kilomètre)
Moins de 100	2.00 DA
Entre 100 et 199	1.75 DA
Entre 200 et 399	1.50 DA
400 ou Plus	1.25 DA

Le prix du billet dépend de la distance entre cette ville et les différentes destinations.

Chaque voyageur connait la distance entre cette ville et sa destination, il doit donc payer un billet et attendre le prochain bus.

1. Ecrire un algorithme qui demande au voyageur la distance pour définir le prix du billet.
2. Ecrire l'algorithme précédent en utilisant l'instruction « **Cas** ».

Exercice 11

Un magasin offre des réductions sur les achats de ses clients comme suit :

- Si le montant d'achat est inférieur à 500 DA, donc il n'y a pas de réduction.
- Si le montant d'achat est compris entre 500 DA et 1000 DA, le taux de réduction est 2.5%.
- Si le montant d'achat est compris entre 1000 DA et 5000 DA, le taux de réduction est 5%.
- Si le montant d'achat est supérieur à 5000 DA, le taux de la réduction est 10%.

Ecrire l'algorithme qui détermine le prix net à payer.

Exercice 12

Une agence touristique organise des sorties familiales avec une prise en charge du transport et d'hébergement, chaque personne doit payer 2000 DA pour le transport, et 3000 DA pour l'hébergement. L'agence propose des remises des prix pour les enfants selon le tableau suivant :

Age	Remise
Moins de 5 ans	75 %
Entre 6 et12 ans	50 %
Entre 13 et16 ans	25 %

Une famille composée de deux parents et 3 enfants va assister à cette sortie.

1. Ecrire un algorithme qui demande les âges des enfants pour définir et afficher le montant total à payer pour cette famille.
2. Ecrire l'algorithme précédent en utilisant l'instruction **Cas**.

Exercice 13

Dans un examen oral, 3 candidats passent l'un après l'autre par un examinateur qui pose des questions à chaque candidat pour lui donner une note.
La table suivante indique la durée d'examen de chaque candidat :

Candidat	Durée de l'examen
E1	10 minutes et 52 secondes
E2	9 minutes et 37 secondes
E3	11 minutes et 24 secondes

Après le passage de chaque candidat, l'examinateur prend 3 minutes pour donner la note et commencer avec le candidat suivant.
Si l'examen commence à l'instant T (H:M:S), écrire l'algorithme qui affiche le temps de début et de fin de passage de chaque étudiant.

Exercice 14

On veut créer un jeu éducatif permettant aux élèves d'apprendre la table de multiplication.

Le principe de ce jeu est comme suit : on demande à l'utilisateur une valeur, ensuite, on commence à afficher les opérations de multiplication de 1 à 10 par la valeur donnée, l'utilisateur doit à chaque fois donner le résultat de l'opération, si la réponse est correcte, on lui affiche **Bien**, sinon, on affiche **faux** avec la réponse correcte, à la fin, on affiche le nombre de réponses correctes. Le joueur est gagnant si le nombre de réponses correctes dépasse 7 réponses.

- Ecrire l'algorithme de ce jeu

Chapitre 4 :
Les boucles

1. Introduction

Lorsqu'on a un traitement qui doit se faire plusieurs fois dans l'algorithme, ce n'est pas pratique de réécrire un bloc d'instruction plusieurs fois dans l'algorithme.

Il faut donc trouver un moyen permettant d'éviter l'écriture da la même instruction plusieurs fois dans le même algorithme, mais qui permet également de répéter son exécution.

2. Les boucles

Une boucle est une action qui permet de répéter l'exécution d'une ou de plusieurs instructions sans avoir besoin de répéter l'écriture plusieurs fois.

On peut distinguer deux types de boucles selon le nombre de répétition des instructions :

1. Le nombre de répétition est connu à l'avance : La boucle Pour

Dans cette boucle le nombre de répétition est connu à l'avance grâce à l'utilisation d'un compteur qui varie entre une valeur initiale et une valeur finale, ce compteur représente la variable de contrôle de la boucle, lorsque le compteur atteint la valeur finale, la boucle s'arrête automatiquement.

La forme générale de la boucle pour

```
Pour  variable :=valeur initiale à valeur finale
Faire
  Instructions
Fin pour
```

Propriétés

- Chaque passage par les instructions de la boucle est dit **itération**.
- Le nombre d'itérations est connu à l'avance :
  Nombre d'itération = valeur finale- valeur initiale +1.
- L'exécution des instructions de la boucle précède la vérification de la condition d'arrêt, il y a donc au moins une exécution.

Exemple

Ecrire un algorithme permettant de calculer le factoriel d'un nombre entier positif

```
Algorithme Factoriel ;
Var fact, n, i : entier ;
Début
  Lire(n) ;
```

```
fact ← 1 ;
Pour i=1 à n faire
   fact ← fact * i ;
fin pour
écrire (fact) ;
Fin.
```

2. Le nombre de répétitions n'est pas connu à l'avance

Pour pouvoir exécuter une boucle sans avoir besoin de connaitre le nombre de sa répétition à l'avance, il faut éviter l'utilisation d'un compteur, la répétition de la boucle dans ce cas dépend de la valeur d'une condition, lorsque la valeur de condition est changée, la boucle s'arrête.

On peut utiliser deux boucles dans ce cas:

1. La boucle Tant que

Dans cette boucle, la répétition des instructions dépend d'une condition, tant que la condition est vérifiée (sa valeur est vraie) on exécute à nouveau les instructions de la boucle.

La forme générale de la boucle tant que

```
Tant que condition
Faire
   Instructions
Fin tant que
```

Remarque : *il est clair que la condition doit être une expression logique.*

Propriétés

- Dans cette boucle, le nombre d'itérations n'est pas connu à l'avance.
- La vérification de la condition se fait avant l'exécution de la boucle, si la condition n'est pas vérifiée, on sort de la boucle, il se peut donc que le nombre minimal d'itération soit 0.
- À la sortie de la boucle, la condition de la boucle est toujours fausse.

2. La boucle Répéter …jusqu'à

Dans cette boucle, les instructions se répètent lorsque la valeur de la condition est fausse, une fois la condition est vérifiée (sa valeur devient vraie), on sort de la boucle.

La forme générale

```
Répéter
   Instructions
```

Jusqu'à Condition

Propriétés

- Dans cette boucle, le nombre d'itération n'est pas connu à l'avance.
- La condition est vérifiée après l'exécution des instructions, donc il existe au moins une exécution des instructions de la boucle.
- Grâce à cette propriété, la boucle Répéter est utilisée lorsqu'on doit exécuter au moins une fois les instructions de la boucle.
- À la sortie de la boucle, la condition d'arrêt est toujours vraie.

Attention ! Remarque importante

Il faut s'assurer que les itérations d'une boucle permettent de modifier la valeur de la condition de boucle, sinon, la boucle ne s'arrête jamais, et on aura donc une boucle infinie qui conduit à un plantage de l'ordinateur.

Comment choisir la boucle à utiliser dans un algorithme ?

Le choix de la boucle se fait selon la nature du problème :

- Si le nombre de répétition est **connu**, on utilise donc la boucle **Pour**
- Si le nombre de répétition **n'est pas connu**, mais on a au moins **une exécution**, on utilise la boucle **Répéter**
- Si le nombre de répétition **n'est pas connu**, et **peut même être zéro**, on doit utiliser la boucle **Tant que**.

Remarque : *n'importe quelle boucle* ***Pour*** *peut-être remplacée par la boucle* ***Tant que*** *ou* ***Répéter****, mais l'inverse n'est pas toujours vrai.*

Série de TD 04

Remarque : Essayer si possible d'écrire les algorithmes demandés en utilisant les trois boucles.

Exercice 1

Ecrire un algorithme qui permet de :

1. Demander deux nombres entiers positifs, ensuite calculer le premier en puissance de deuxième.
2. Calculer le factoriel d'un nombre donné par l'utilisateur.
3. Afficher les diviseurs d'un nombre entier N.
4. Vérifier si un nombre est premier ou non.
5. Vérifier si le nombre est parfait ou non.
6. Calculer le PGCD de deux nombres entiers A et B.

Exercice 2

Une agence touristique organise des sorties familiales avec une prise en charge du transport et d'hébergement, chaque personne doit payer 2000 DA pour le transport, et 3000 DA pour l'hébergement. L'agence propose des remises des prix pour les enfants selon le tableau suivant :

Age	Remise
Moins de 5 ans	75 %
Entre 6 et12 ans	50 %
Entre 13 et16 ans	25 %

Une famille composée de deux parents et **N** enfants va assister à cette sortie. (La valeur de N est donnée par l'utilisateur).

- Ecrire un algorithme qui demande les âges des enfants pour définir et afficher le montant total à payer pour cette famille.

Exercice 3

Un laboratoire médical fait des analyses pour les patients. Le résultat de chaque analyse est obtenu après un certain nombre de jours selon la nature de l'analyse.

1. Ecrire un algorithme qui affiche la date de résultat d'une analyse à partir de la date de l'analyse. (Le nombre de jours nécessaire pour le résultat de cette analyse est donné par l'utilisateur).
2. Refaire la question précédente en utilisant l'algorithme qui calcule la date du jour suivant.

Exercice 4

Dans un examen oral, les candidats passent l'un après l'autre par des examinateurs qui posent des questions à chaque candidat pour lui donner une note. Après le passage de chaque candidat, les examinateurs prennent 3 minutes pour donner la note et commencer avec le candidat suivant.

- Si l'examen commence à l'instant T (H:M:S), écrire l'algorithme qui affiche le temps de début et de fin de passage de chaque étudiant (le nombre de candidats et la durée de l'examen de chaque candidat sont données par l'utilisateur).

Exercice 5

Un étudiant prépare ses examens finaux qui vont commencer à une date connue, et il veut connaitre à n'importe quel jour le nombre de jours restants avent cette date importante.

- Ecrire l'algorithme qui demande une date à l'utilisateur pour afficher le nombre de jours restants avant le début des examens.

Exercice 6

On veut créer un jeu éducatif permettant aux élèves d'apprendre la table de multiplication.

Le principe de ce jeu est comme suit : on demande à l'utilisateur une valeur, ensuite, on commence à afficher les opérations de multiplication de 1 à 10 par la valeur donnée,

l'utilisateur doit à chaque fois donner le résultat de l'opération, si la réponse est correcte, on lui affiche **Bien**, sinon, on affiche **faux** avec la réponse correcte, à la fin, on affiche le nombre de réponses correctes. Le joueur est gagnant si le nombre de réponses correctes dépasse 6 réponses.

- Ecrire l'algorithme de ce jeu.

Chapitre 5 :
Les Tableaux

1. Introduction

Lorsqu'on écrit des algorithmes, on peut manipuler des variables et des constants de différents types, et on peut déclarer autant de variables qu'on a besoin pour résoudre les différents problèmes.

Il se peut que dans un algorithme donné, on aura besoin de manipuler 100 variables de même type, comme par exemple le calcul de moyenne des étudiants. Il est possible donc de déclarer 100 variables de type réel.

Un tel algorithme n'est pas pratique, et la manipulation d'un nombre important de variables est très difficile. Il faut donc trouver un moyen permettant de manipuler facilement un nombre important de données ayant le même type.

La solution consiste à utiliser une nouvelle structure de données : **les tableaux**

2. Les tableaux

2.1. Définition

Un tableau (appelé également vecteur) est une structure de données complexe qui permet de regrouper plusieurs données ayant le même type.

Ces données ont toutes le même nom (le nom du tableau), le même type, et peuvent avoir des valeurs différentes.

2.2. Elément du tableau

Chaque donnée dans le tableau est appelée élément du tableau.

Un élément est caractérisé par son nom (le nom du tableau), un type (le type du tableau), et une valeur.

Chaque élément est identifié par un indice (un rang) qui indique sa position dans le tableau.

2.3. Dimension d'un tableau

La dimension (ou la taille) d'un tableau désigne le nombre des éléments de ce tableau.

2.4. Déclaration d'un tableau

Un tableau est une variable qui a un nom, une dimension, et un type qui représente le type de tous ces éléments, la forme générale de la déclaration d'un tableau est donnée comme suit :

Var nom du tableau : Tableau [1..Dimension] de type ;

Exemple

Var tab : Tableau [1..5] d'entier ;

Tab est un tableau de cinq éléments de type entiers, et on peut imaginer sa forme comme ça:

12	-5	3	0	29

2.5. Accès à un élément dans un tableau

L'accès à un élément dans le tableau se fait par son indice, il faut donc mentionner le nom du tableau suivi par la position de l'élément dans le tableau.

Exemple

Tab [1] : le 1er élément du tableau

Tab [2] : le 2ème élément du tableau

…

Tab [N] : le dernier élément du tableau tel que N est la dimension du tableau.

Remarques

- Chaque élément du tableau peut être traité indépendamment des autres éléments.
- Les opérations possibles avec un élément du tableau sont toutes les opérations possibles avec le type de cet élément.

3. Exercice d'application

Soit T un tableau de trois éléments entiers.

Donner l'algorithme qui permet d'afficher la somme des éléments de ce tableau.

```
Algorithme Somme ;
Var T : tableau [1..3] d'entier ;
    S : entier ;
Début
  Lire ( T[1], T[2], T[3]) ;
  S← T[1] +T[2]+T[3] ;
  Ecrire (S) ;
Fin
```

Remarque

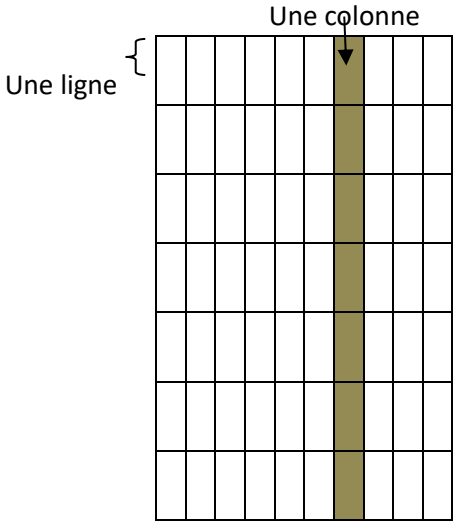
jusqu'à maintenant, nous avons manipulé la structure de données **Tableau** (qu'on a appelé également **Vecteur**) qui nous a permis de traiter les problèmes qui nécessitent la manipulation de plusieurs données ayant le même type. Par exemple, si on veut calculer la moyenne d'un étudiant, il suffit de déclarer un vecteur avec N cases tel que N est le nombre de modules étudiés par cet étudiant. Et pour calculer la température moyenne d'une ville pendant un mois, il faut déclarer un tableau de 31 cases, où chaque case représente la température d'un jour de ce mois. Maintenant, si on veut calculer la moyenne des étudiants d'une section de 150 étudiants, il nous faut 150 tableaux, chaque tableau pour un étudiant ! Et pour calculer la température moyenne

de toutes les wilayas, il faut déclarer 48 tableaux !!

Il est claire que l'utilisation d'un nombre élevé de tableaux dans un algorithme n'est pas pratique, il faut donc faire appel à une nouvelle structure de données qui permet de faciliter l'utilisation d'un nombre important de tableau tout en simplifiant leur manipulation, la solution consiste à concaténer plusieurs vecteurs dans la même structure de données, on utilise donc **Les tableaux à deux dimensions** ou autrement dit : **les Matrices**.

4. Les tableaux à deux dimensions

Un tableau à deux dimensions (Matrice) est très similaire à un ensemble de vecteurs concaténés, chaque ''vecteur'' représente une ligne dans la matrice, et toutes les cases du même rang représentent une colonne dans la matrice. La figure suivante représente une matrice composée de 7 lignes et 10 colonnes.



4.1. Dimensions d'une matrice

Une matrice est caractérisée par deux dimensions : la première dimension désigne le nombre de lignes dans cette matrice, et la deuxième dimension désigne le nombre de colonnes.

Remarque :

Dans une matrice, toutes les lignes ont le même nombre de cases.

4.2. Déclaration d'une matrice

Un tableau à deux dimensions est une variable qui a un nom, et un type qui représente le type de tous ses éléments, la forme générale de la déclaration d'un tableau est donnée comme suit :

Var nom du tableau : Tableau [1..Nombre de lignes, 1..Nombre de colonnes] de type ;

Exemple

Var tab : Tableau [1..3,1..4] d'entier ;

Tab est un tableau de trois lignes et 4 colonnes et dont les éléments sont de type entier.

4.3. Elément d'une matrice

Un élément dans une matrice est caractérisé par son nom (le nom du tableau), un type (le type du tableau), et une valeur.

Chaque élément est identifié par deux indices : un indice indiquant le numéro de la ligne, et un autre indice indiquant le numéro de la colonne.

Pour accéder à un élément dans la matrice, il faut mentionner le numéro de la ligne, et le numéro de la colonne.

Exemple

Tab[1,3] : représente l'élément de la ligne 1 et la colonne 3

4.4. La matrice carrée

La matrice carrée est une matrice particulière dont le nombre de lignes et le même avec le nombre de colonnes.

La déclaration d'une matrice carrée peut être donnée comme suit :

Var Mat : tableau [1..n, 1..n] d'entier;

5. Manipulation des matrices par les boucles

En général, la manipulation des matrices par les boucles nécessite deux boucles imbriquées, car on a besoin d'une boucle pour se déplacer d'une ligne à une autre, et d'une autre boucle pour se déplacer d'une colonne à une autre.

Selon les emplacements des boucles, on aura deux modes de traitement :

Si la boucle qui fait varier l'indice de la ligne est la boucle externe, la matrice est donc manipulée ligne par ligne, car avec chaque changement d'un indice de la ligne, on change tous les indices des colonnes.

Si la boucle qui fait varier l'indice de la colonne est la boucle externe, la matrice est donc manipulée colonne par colonne, car avec chaque changement d'un indice de la colonne, on change tous les indices des lignes.

Exemple

Prenons l'exemple de l'écriture d'une matrice Tab de N lignes et M colonnes, on peut écrire les éléments de cette matrice par deux façons différentes :

```
…

Pour i := 1 à N

Faire

    Pour j :=1 à M
```

```
…

Pour j := 1 à M

Faire

    Pour i :=1 à N
```

```
    Faire
        Ecrire (Tab [i , j ]) ;
    Fin pour
Fin pour
….
```

La matrice est écrite ligne par ligne

```
    Faire
        Ecrire (Tab [i , j ]) ;
    Fin pour
Fin pour
….
```

La matrice est écrite colonne par colonne

5.1. Exercices d'application

Exercice 01

Soit Mat une matrice de 3 lignes et 5 colonnes dont les éléments sont des entiers. Donner l'algorithme qui permet d'afficher la somme des éléments de cette matrice.

```
Algorithme Somme_matrice ;
Var Mat : tableau [1..3, 1..5] d'entier ;
    i, j, S : entier ;
Début
Pour i := 1 à 3
Faire
    Pour j :=1 à 5
    Faire
        lire (Mat [i , j ]) ;
    Fin pour
Fin pour
S := 0 ;
Pour i := 1 à 3
Faire
    Pour j :=1 à 5
    Faire
        S := S + Mat [i , j ] ;
    Fin pour
Fin pour
Ecrire ('La somme des éléments de cette matrice est :', S) ;
Fin.
```

Série de TD 05

Exercice 1

Soit Tab un tableau de 100 éléments entiers, donner l'algorithme qui permet d'afficher :

1. La somme des éléments de ce tableau.
2. La valeur max et la valeur min de ce tableau avec leurs positions.
3. La première occurrence d'une valeur donnée par l'utilisateur.
4. Le nombre d'occurrence d'une valeur donnée par l'utilisateur.

Exercice 2

Soit **Tab** un tableau de **N** éléments entiers, et soit **Val1** et **Val2** deux valeurs différentes données par l'utilisateur.

1. Ecrire un algorithme permettant de calculer et afficher le nombre d'éléments supérieurs à **Val1** dans une variable **Supval1**, et le nombre d'éléments inférieurs à **Val2** dans une variable **Infval2**.
2. Ecrire un algorithme permettant de calculer le nombre d'éléments compris entre **Val1** et **Val2**.

Exercice 3

Soit **T** un tableau de **2*N** éléments entiers, ce tableau contient **N** éléments positifs et **N** éléments négatifs, et Soit **Tab** un tableau d'entiers.

- Ecrire un algorithme permettant de mettre les valeurs positives du tableau **T** dans les cases ayant un indice impair dans le tableau **Tab**, et les valeurs négatives du tableau **T** dans les cases ayant un indice pair dans le tableau **Tab**.

Exercice 4

Soit **Tab** un tableau de 100 éléments entiers, et soit **Val** une valeur entière qui se répète dans ce tableau.

- Par deux méthodes différentes, écrire un algorithme permettant d'afficher l'indice de la dernière occurrence de la valeur **Val** dans le tableau **Tab**.

Exercice 5

Soit T1[N] et T2[M] deux tableau **triés par ordre croissant** et qui contiennent des éléments entiers. On veut fusionner les deux tableaux dans un seul tableau T qui doit être **trié par ordre décroissant**.

1. Quelle est la dimension du tableau T
2. Donner l'algorithme qui réalise cette fusion.

Exercice 6

Soit Tab un tableau de 100 éléments entiers, ce tableau contient un seul élément nul.

- Ecrire un algorithme permettant de décaler les éléments de ce tableau vers la gauche (décalage circulaire) de telle sorte que le premier élément de ce tableau (Tab [1]) soit l'élément nul.

Exemple

Si le tableau Tab est donné comme suit :

2	6	-4	0	12	-33	9	1	8	-21

Le résultat doit être :

0	12	-33	9	1	8	-21	2	6	-4

On suppose maintenant que le tableau Tab contient deux éléments nuls, le premier se trouve dans la première moitié du tableau (son indice est inférieur à 51), et le deuxième se trouve dans la deuxième moitié (son indice est supérieur à 50).

- Ecrire un algorithme qui permet de mettre les deux éléments nuls au milieu du tableau Tab

Exemple

Si le tableau Tab était donné comme suit :

2	6	-4	0	12	-33	9	1	5	-21	8	-1	0	27

Le résultat doit être :

12	-33	9	2	6	-4	0	0	27	1	5	-21	8	-1

Exercice 7

Pour créer un jeu éducatif pour les enfants, on utilise un tableau de 20 éléments qui contient des valeurs entières.

Le principe de ce jeu est comme suit : l'enfant donne le numéro d'une case, sa valeur sera affichée, donc l'enfant doit écrire le double de cette valeur. Si la réponse est correcte on lui donne 2 points, sinon on l'informe que la réponse est fausse. Le jeu s'arrête si le joueur atteint 20 points ou il donne 3 réponses fausses.

- Ecrire l'algorithme de ce jeu.

On veut ajouter une autre condition à ce jeu, le principe reste le même, mais le jeu s'arrête si le joueur atteint 20 points, donne 4 réponses fausses, ou donne 2 réponses fausses **consécutives**.

- Ecrire le nouvel algorithme de ce jeu.

Exercice 8

Soit **T** un tableau de 100 cases, ce tableau contient **K** éléments entiers triés par ordre croissant (**K<100**), on suppose que les autres cases sont vides.

Ecrire l'algorithme qui permet d'insérer une nouvelle valeur donnée par l'utilisateur dans le tableau **T**. **Notez bien que le tableau T doit rester trié après l'insertion de la nouvelle valeur**.

Exemple

Si le tableau **T** est donné comme suit :

-2	-1	0	2	3	4	7	11	15	18				

Et lorsqu'on insère la valeur **5**, le résultat doit être :

-2	-1	0	2	3	4	5	7	11	15	18			

Exercice 9

Soit **T** un tableau de 100 éléments entiers triés par ordre décroissant, et soit **Val** une valeur qui se trouve dans ce tableau.

- Ecrire l'algorithme qui permet de retirer la valeur **Val** de ce tableau. (Notez bien que le tableau **T** doit rester trié après la suppression de la valeur **Val**)

Exercice 10

Soit A [M,N] une matrice d'entiers à M lignes et N colonnes. Donner les algorithmes permettant de calculer :

1. La somme et le nombre des éléments supérieurs à une valeur X donnée par l'utilisateur.
2. Le nombre des éléments pairs dans une ligne donnée par l'utilisateur.
3. Le nombre des éléments impairs dans une colonne donnée par l'utilisateur.

Soit A [N,N] une matrice carrée. Donner les algorithmes permettant de :

1. Calculer la trace de la matrice A.
2. Calculer la somme des éléments de $2^{ème}$ diagonale de cette matrice.

Exercice 11

Dans une entreprise privée, 150 employés ont des salaires déjà fixés, et ils peuvent travailler des heures supplémentaires. L'employé peut bénéficier de 100 dinars de plus pour chaque heure supplémentaire, et le nombre des heures supplémentaires pendant un mois est calculé à la fin du mois pour ajouter le montant au salaire de l'employé. On utilise trois tableaux: **S_initial** un tableau qui contient le salaire initial de chaque employé, un tableau **Sup** qui contient le nombre

des heures supplémentaires de chaque employé, et un tableau **Salaire** qui contient le salaire final, c'est à dire, le salaire initial en plus du montant des heures supplémentaires.

- Ecrire l'algorithme qui permet de remplir et afficher les salaires finaux des employés.

Exercice 12

1. Ecrire un algorithme permettant de calculer le nombre de lettres voyelles dans une chaine de caractères donnée par l'utilisateur.

Exercice 13

Soit S une chaîne de caractères composée des lettres voyelles et consonnes et des caractères blancs. Cette chaine de caractères se termine par la lettre '#'.

1. Ecrire l'algorithme qui affiche le nombre de lettres consonnes dans cette chaîne de caractères (utiliser l'instruction conditionnelle Si ou l'instruction de choix Case).
2. Ecrire l'algorithme qui permet de calculer le nombre de mots dans cette chaine.

Chapitre 6 :
Les sous-programmes :
Les fonctions & les procédures

1. Introduction

Pour résoudre un problème en algorithmique, il faut écrire des algorithmes et des programmes permettant de trouver les solutions de ces problèmes. Dans ce chapitre nous allons découvrir une nouvelle notion qui facilite l'écriture et la manipulation des algorithmes : les sous-programmes.

2. Les sous-programmes

2.1. Pourquoi les sous-programmes ?

Pour comprendre l'intérêt des sous-programmes, on prend l'exemple suivant :

Ecrire un algorithme qui permet de calculer le factoriel d'une valeur entière positive.

L'algorithme est simple, et peut être donné comme suit :

```
Algorithme factoriel ;
Var  x, fact, i : entier ;
Début
Lire (x) ;
Fact := 1 ;
Pour i := 1 à x
Faire
        Fact := fact * i ;
Fin pour
Ecrire (fact) ;
Fin.
```

Supposons maintenant qu'on veut écrire l'algorithme qui calcule le factoriel de deux valeurs entières positives, dan ce cas, on doit écrire la boucle deux fois dans l'algorithme.

On suppose maintenant qu'on a un tableau de 100 éléments, et on veut calculer le factoriel de chaque élément, dans ce cas, on aura un algorithme qui contient 100 boucles ! Une chose qui n'est pas pratique.

Pour éviter ces situations, on peut écrire le bloc d'instructions qui calcule le factoriel une seule fois sans préciser la valeur, ensuite, on peut appeler ce bloc plusieurs fois avec des valeurs différentes.

Ce bloc d'instruction représente un **Sous programme.**

2.2. Définition

Un sous programme (dit aussi **Action paramétrée**) est un ensemble d'instructions permettant de résoudre un problème donné, cet ensemble d'instructions est écrit une seul fois dans l'algorithme, et peut être utilisé autant de fois que nécessaire.

Remarques

- Dans un algorithme, on peut trouver plusieurs sous-programmes.
- La déclaration des sous-programmes se fait après la déclaration des variables globales de l'algorithme.
- Chaque sous-programme est caractérisé par un nom unique.
- Un sous-programme peut être utilisé dans l'algorithme principal, et peut être également utilisé dans un autre sous-programme.
- L'utilisation du sous-programme est dite **Appel de sous programme**, on a donc le **sous programme appelé** qui est le sous programme utilisé, et le **(sous-)programme appelant** qui utilise le sous programme appelé.
- La déclaration d'un sous-programme appelé doit figurer avant la déclaration de (sous)programme appelant.

2.3. Les paramètres d'un sous-programme

Les paramètres sont les informations traitées par le sous-programme pour qu'il puisse fournir des résultats, ces paramètres peuvent être des variables ou des constants.

On peut distinguer trois types de passage de paramètres :

Le passage de paramètres en entrée

- Dans ce cas, la valeur du paramètre est utilisée dans le sous-programme sans changement, la valeur de l'entité de (sous-)programme appelant est copiée dans le paramètre, et la valeur reste inchangée.
- Les paramètres d'entrée représentent les données traitées par le sous-programme.
- Les paramètres d'entrée peuvent être des constants.

Le passage de paramètres en sortie

- Avec le passage de paramètres en sortie, la valeur du paramètre est obligatoirement

modifiée par le sous-programme.

- Les paramètres de sortie contiennent généralement les résultats fournis par le sous-programme.
- Les paramètres de sortie ne peuvent pas être des constants.

Le passage de paramètres en entrée-sortie

- C'est un type de passage qui combine les deux autres types, et qui est utilisé lorsque le sous-programme doit utiliser et/ou modifier la valeur de variables de (sous-)programme appelant.
- Avec les paramètres d'entrée-sortie, on ne peut pas utiliser des constants.

3. Les types des sous-programmes

On peut distinguer deux types de sous-programmes : les fonctions et les procédures

4. Les fonctions

4.1. Définition

Une fonction est un sous-programme utilisant des paramètres pour fournir un seul résultat de type scalaire.

4.2. Caractéristiques d'une fonction

- Une fonction a un type qui est le type de résultat retourné par cette fonction.
- Une fonction accepte des paramètres d'entrée seulement, on peut donc utiliser des variables ou des constants lors de l'appel de la fonction.
- Le résultat (ou la valeur de paramètre de sortie) est donné dans le nom de la fonction.

4.3. Déclaration d'une fonction

La forme générale de déclaration d'une fonction est :

```
Fonction  nom de fonction (Les paramètres d'entrée) : Type de résultat ;
Déclaration des variables locales
Début
   Le corps de la fonction (Les instructions de la fonction)
Fin ;
```

Remarques

- Le corps de la fonction doit contenir une instruction qui affecte un résultat au nom de la fonction.
- Le nom de la fonction doit figurer seulement dans la partie gauche d'une instruction d'affectation.

4.4. Exemple

La déclaration d'une fonction qui calcule le factoriel :

```
Fonction fact (n : entier) : entier ;
Var i, ft : entier ;
Début
        ft := 0 ;
        Pour i :=1 à n
        Faire
                ft := ft * i ;
        Fin pour
        fact := ft ;
Fin ;
```

5. Les procédures

5.1. Définition

Une procédure est un sous-programme utilisé pour réaliser des actions (instruction d'affichage par exemple), ou pour retourner plusieurs résultats.

5.2. Caractéristiques des procédures

- Une procédure n'a pas un type.
- Dans une procédure, les paramètres d'entrée représentent les données.
- Les résultats de procédure sont stockés dans les paramètres de sortie ou dans les paramètres d'entrée-sortie.
- Pour faire la différence entre les paramètres d'entrée d'une part et les paramètres de sortie et d'entrée-sortie d'autre part, ces derniers doivent être précédés par le mot clé var dans la déclaration.

5.3. Déclaration d'une procédure

La forme générale de déclaration d'une procédure est :

```
Procédure  Nom de procédure (Paramètres d'entrée, Var Paramètres de sortie, Var Paramètres
d'entrée-sortie) ;
Déclaration des variables locales ;
Début
        Le corps de la procédure (Les instructions de la procédure)
Fin ;
```

5.4. Exemples

On va donner 3 exemples :

1. Une procédure utilisant des paramètres d'entrée seulement :

On déclare une procédure Comparer qui compare deux valeurs et affiche la plus grande entre elle :

```
Procédure Comparer (a, b : entier) ;
Début
        Si a > b alors
                Ecrire (a, 'est le maximum')
                Sinon si a < b alors
                        Ecrire (b, 'est le maximum')
                        Sinon écrire (a, 'égale ',b) ;
                    Finsi
        Finsi
Fin ;
```

On remarque dans cet exemple qu'on n'a pas besoin des paramètres de sortie car la procédure fait seulement une action d'affichage.

2. Une procédure utilisant les paramètres d'entrée et des paramètres de sortie :

```
Procédure Calculer (a, b : entier, var s, m : entier) ;
```

```
Début
        S := a + b ;
        M := a * b ;
Fin ;
```

3. Une procédure utilisant des paramètres d'entrée-sortie :

```
Procédure Permuter (var x, y : entier) ;
Var z : entier ;
Début
        z := x ;
        x := y ;
        y := z ;
Fin ;
```

6. Appel des fonctions et des procédures

Les fonctions et les procédures déclarées dans un algorithme peuvent être appelées par l'algorithme principal ou par d'autres fonctions ou procédures dans cet algorithme.

L'appel se fait par l'écriture du nom de fonction ou de procédure suivie par les paramètres qui doivent être des variables globales (ou des constants dans le cas des paramètres d'entrée).

Exemple

L'algorithme suivant permet de calculer le factoriel des éléments d'un tableau et de mettre les résultats dans un autre tableau :

```
Algorithme Factoriel ;
Var T, Fact_T : Tableau [1..100] d'entier ;
    indice : entier ;
Fonction fact (n : entier) : entier ;
  Var i, ft : entier ;
  Début
        ft := 0 ;
        pour i :=1 à n
        faire
```

```
                ft := ft * i ;
        fin pour
        fact := ft ;
  fin ;
Début
  pour indice := 1 à 100
  faire
        lire (T[ indice]) ;
  fin pour
  pour indice := 1 à 100
  faire
        Fact_T[indice] := fact( T[indice]) ;
  Fin pour
  pour indice := 1 à 100
  faire
        Ecrire (Fact_T[indice] ) ;
  Fin pour
Fin.
```

Série de TD 06

Remarque : Dans cette série d'exercices, il faut préciser pour chaque action paramétrée demandée s'il s'agit d'une fonction ou d'une procédure, en désignant les paramètres d'entrée et les paramètres de sortie.

Exercice 1

1. Ecrire une action paramétrée permettant de vérifier si un nombre réel est positif ou non.
2. Utiliser cette action dans l'algorithme qui calcule la racine carrée d'une valeur donnée.

Exercice 2

1. Ecrire l'action paramétrée qui prend en entrée deux valeurs entières positives a et b, et calcule le premier en puissance de deuxième (a^b).
2. Ecrire l'action paramétrée qui calcule le produit de deux valeurs entières positives a et b par la méthode d'addition successive.

Exercice 3

Un nombre parfait est un entier naturel **N** tel que la somme de ces diviseurs entiers positifs = 2 * N .

Cela revient à dire qu'un entier naturel est parfait s'il est égal à la moitié de la somme de ses diviseurs. Ainsi 6 est un nombre parfait car $2 \times 6 = 12 = 1 + 2 + 3 + 6$, ou encore $6 = 1 + 2 + 3$. On veut écrire un algorithme qui affiche tous les nombres parfaits dans un intervalle de valeurs donné par l'utilisateur, pour cela, on utilise des actions paramétrées.

1. Ecrire une action paramétrée permettant de vérifier si un nombre entier est premier ou non.
2. Utiliser l'action paramétrée précédente dans l'action paramétrée qui affiche les diviseurs d'un nombre donné.
3. Utiliser l'action paramétrée précédente dans l'action paramétrée qui permet de vérifier si un nombre entier est parfait ou non.

4. Utiliser l'action paramétrée précédente dans l'algorithme qui affiche tous les nombres parfaits dans un intervalle compris entre deux valeurs X et Y.

Exercice 4

1. Ecrire une action paramétrée permettant de calculer la somme des éléments d'un vecteur.
2. Utiliser l'action paramétrée précédente dans l'algorithme qui calcule la somme des éléments de trois vecteurs d'entiers A, B, et C de 100 éléments chacun.
3. Peut-on utiliser l'action paramétrées précédente dans un algorithmes permettant de calculer la somme des éléments d'une matrice? Si oui, Comment?
4. Ecrire une action paramétrée permettant de trouver le max et le min d'un vecteur.
5. Ecrire un sous- programme qui vérifie si un vecteur est trié.

Exercice 5

1. Ecrire une action paramétrée Est_voyelle qui prend en entrée un caractère, et qui rend vrai si ce caractère est une lettre voyelle.
2. Utiliser cette action paramétrée dans l'algorithme qui calcule le nombre de lettres voyelles dans une chaine de caractères.

Exercice 6

1. Ecrire une action paramétrée permettant d'afficher le mot miroir d'une chaîne de caractères.
2. Ecrire une action paramétrée permettant de vérifier si un mot est palindrome.
3. Peut on utiliser l'une des action paramétrées précédentes pour écrire l'autre? Si oui, Comment?

Exercice 7

Ecrire un algorithme permettant de calculer le nombre d'occurrence d'une valeur donnée par l'utilisateur dans une matrice A[M, N] d'entiers.

On veut maintenant réaliser le même travail de l'algorithme précédent en utilisant une action paramétrée appelé **Occurrence**.

1. L'action paramétrée utilisé doit être une fonction ou une procédure ? justifier votre réponse.
2. Ecrire l'action paramétrée **Occurrence** en précisant ses paramètres.
3. Utiliser l'action paramétrée **Occurrence** dans un algorithme permettant de calculer le nombre d'occurrence de 3 valeurs différentes X1, X2, et X3 respectivement dans 3 matrices d'entiers : A [M, N], B [M, N] et C [M, N].

Exercice 8

Soit A [N, N] une matrice d'entiers.

1. Ecrire l'algorithme qui permute les éléments des deux diagonales de cette matrice (les éléments permutés sont les éléments de la même ligne).

On veut maintenant réaliser le même travail de l'algorithme précédent en utilisant les action paramétrées, pour cela on utilise une action paramétrée appelé **Permute** qui permet de permuter deux valeurs entières.

1. L'action paramétrée utilisé doit être une fonction ou une procédure ? justifier votre réponse.
2. Ecrire l'action paramétrée **Permute** en précisant ses paramètres.

3. Utiliser l'action paramétrée **Permute** dans un algorithme qui permute les éléments des diagonales d'une matrice A [N, N].

Exercice 9

1. Soit **Mat[50, 40]** une matrice d'entiers, cette matrice contient le même nombre des éléments positifs et négatifs (La matrice ne contient pas 0).

- Ecrire un algorithme permettant de transférer les éléments de la matrice **Mat** vers une autre matrice **A** de telle sorte que les éléments positifs soient dans les lignes ayant un indice pair, et les éléments négatifs soient dans les lignes ayant un indice impair.

2. On veut maintenant réaliser le même travail de l'algorithme précédent en utilisant les actions paramétrées, pour cela on utilise une action paramétrée **Transfert**.

- L'action paramétrée **Transfert** est une fonction ou une procédure ? justifier.
- Ecrire le nouvel algorithme qui contient l'action paramétrée **Transfert**.

Chapitre 7 :
Les Enregistrements

1. Introduction

L'utilisation des structures de données complexes telles que les vecteurs et les matrices nous a permis de manipuler plusieurs données ayant le même type. Il se peut que pour certains traitement, on aura besoin de manipuler plusieurs données ayant des types différents, mais qui concernent une seule entité, cela est possible en utilisant une nouvelle structure de données : Les enregistrements.

2. Les enregistrements

2.1. Définition

L'enregistrement est une structure de données qui permet de regrouper un ensemble de données ayant des noms différents, des types différents et des valeurs différentes, chaque donnée représente un élément de l'enregistrement, et elle est appelé champ.

2.2. Déclaration d'un enregistrement

Un enregistrement est caractérisé par ses champs, et chaque champ a un nom et un type

La forme générale de déclaration est

```
Type  Nom de l'enregistrement = enregistrement (record)
        Nom d'un champ : type de ce champ ;
        Nom d'un champ : type de ce champ ;
        ….
       Nom d'un champ : type de ce champ ;
Fin ;
```

Après la déclaration de l'enregistrement, on doit déclarer des variables dont le type est le nom de l'enregistrement.

Exemple

```
Type Etudiant = enregistrement
          num : entier ;
          nom, prenom : chaîne ;
          année_étude : entier ;
fin ;
```

```
var E1, E2, E3 : Etudiant ;
```

2.3. Clé d'un enregistrement

La clé (l'identifiant) d'un enregistrement est un champ dont la valeur est unique, c'est-à-dire, la valeur de ce champ est différente de toutes les autres valeurs du même champ dans les autres enregistrements, cette unicité permet d'identifier l'enregistrement.

2.4. Accès aux champs d'un enregistrement

Pour accéder à un champ dans un enregistrement, on doit écrire le nom de l'enregistrement suivi par un point, suivi par le nom du champ.

Nom de l'enregistrement (nom de variable) . Nom du champ

Exemple

- L'accès au champ Num de l'enregistrement E1 de l'exemple précédent se fait comme suit : E1.num.
- E2.nom désigne le champ nom dans l'enregistrement E2.

2.5. Manipulation des champs

Les manipulations possibles avec un champ sont toutes les manipulations possibles avec le type de ce champ.

Exemple

```
Lire (E1.num) ;

E3.année_étude := 3 ;
```

On peut même réaliser des opérations entre les mêmes champs des différents enregistrements, et on peut manipuler globalement des enregistrements.

Exemple

```
E1.nom :=E2. Nom ;
```

Cette instruction permet d'affecter le contenu du champ nom de E2 au champ nom de E1

```
E3 := E1 ;
```

Cette instruction permet d'affecter tous les champs de E1 aux champs correspondants de E3

3. Enregistrements d'enregistrements

On peut déclarer dans un enregistrement des champs qui peuvent être également des enregistrements.

Exemple

Si on veut ajouter la date de naissance à l'enregistrement Etudiant, on peut déclarer un enregistrement date, et cet enregistrement va représenter un champ dans l'enregistrement Etudiant.

La déclaration sera comme suit :

```
Type  Date = enregistrement
        Jour : 1..31 ;
        Mois : 1..12 ;
        Année : 0..99 ;
Fin ;
Type  Etudiant = enregistrement
        Num : entier ;
        Nom : chaîne ;
        Prénom : chaîne ;
        Année_étude : entier ;
        Date_naissance : date ;
Fin ;
Var Etud : Etudiant
```

Si on veut par exemple accéder au champ jour de la date de naissance de la variable Etud, l'accès se fait comme suit :

```
Etud.date_naissance.jour := 21 ;
```

4. Exercice d'application

1. Donner la structure de l'enregistrement Etudiant qui contient les informations concernant un étudiant ayant 3 modules.
2. Ecrire un algorithme permettant de calculer la moyenne d'un étudiant.

```
Algorithme notes ;
Type  Date = enregistrement
        Jour : 1..31 ;
        Mois : 1..12 ;
        Année : 0..99 ;
Fin ;
Type  Etudiant = enregistrement
        Num : entier ;
        Nom : chaîne ;
        Prénom : chaîne ;
        Année_étude : entier ;
        Date_naissance : date ;
        Mudule1, Module2, Module3, Moy: réel ;
Fin ;
Var  Etud : Etudiant ;
Début
         Lire(Etud.num) ;
        Lire(Etud.nom) ;
        Lire(Etud.Prenom) ;
        Lire(Etud.année_étude) ;
        Lire(Etud.date_naissance.jour) ;
        Lire(Etud.date_naissance.mois) ;
        Lire(Etud.date_naissance.année) ;
        Lire(Etud.module1) ;
        Lire(Etud.module2) ;
        Lire(Etud.module3) ;
        Etud.moy := (Etud.module1+ Etud.module1+ Etud.module1) / 3 ;
Fin.
```

5. L'instruction Avec

Dans l'exemple précédent, on remarque que le nom de variable est répété avec chaque accès aux champs, pour éviter ça, on utilise l'instruction avec qui nous permet d'écrire le nom de variable une seule fois, ensuite seul le nom de l'enregistrement est mentionné.

La forme générale de l'instruction Avec est donnée comme suit :

```
Avec  nom_de_varaible faire
Début
        Nom d'un champ…
        ….
Fin ;
```

Exemple

Pour l'exemple précédent la lecture des champs de la variable Etud se fait comme suit :

```
Avec  Etud  faire
Début
        Lire(num) ;
        Lire(nom) ;
        Lire(module1) ;
        Lire(module2) ;
        Lire(module3) ;
        Moy := (module1 + module2 +module3) / 3 ;
Fin ;
```

Remarque

On peut utiliser des instructions Avec imbriquées dans le cas des enregistrements d'enregistrements.

Exemple

La variable Etud est un enregistrement d'enregistrements, on peut donc utiliser des instructions Avec imbriquées comme suit :

```
Avec Etud faire
Début
        Lire (num) ;
        Lire (nom) ;
        Avec  Date_naissance faire
        Début
                Lire (jour) ;
                Lire (mois) ;
                Lire (année) ;
        Fin ;
        Lire (module1) ;
        Lire (module2) ;
        Lire (module3) ;
        Moy := (module1 + module2 +module3) / 3 ;
Fin ;
```

Série de TD 07

Exercice 1

On veut gérer le passage des étudiants de la 1ère année à la 2ème année dans un département donné, pour cela, chaque étudiant doit préciser son choix entre deux spécialités SP1 et SP2. Le choix de l'étudiant qui choisit la spécialité SP1 sera accepté si sa moyenne générale est supérieure ou égale à 12, et que le nombre des étudiants qui sont déjà affectés à cette spécialité n'a pas dépassé 80, dans ce cas, il sera mis dans un tableau SP1, sinon, l'étudiant sera affecté à la 2ème spécialité SP2 et il sera mis dans un tableau SP2. Le nombre des étudiants de la 1ère année est 200, et chaque étudiant est caractérisé par toutes ses informations (nom et prénom, adresse, moyenne générale, ...etc.) en plus de son choix.

- En utilisant les enregistrements et les tableaux, écrire l'algorithme qui permet de gérer ce problème.

Exercice 2

Une agence immobilière lance des annonces de vente des logements. Chaque logement est caractérisé par un numéro, un nombre de chambres, un prix et une adresse. Toutes les annonces sont stockées dans un tableau **Annonces** de 300 éléments où chaque élément représente un logement.
Un client peut lancer une recherche en précisant le nombre de chambres demandé et le prix qu'il peut payer, le résultat sera donc le logement ayant le même nombre de chambres demandé et le prix le plus proche au prix proposé par le client.

1. Donner la structure de l'enregistrement **Logement.**
2. Ecrire l'algorithme qui permet d'afficher à un client le logement le plus adéquat.

Exercice 3

Dans un magasin de produits alimentaires, chaque produit est caractérisé par un numéro, un nom, une quantité, un prix, une date de fabrication et une date d'expiration. On utilise un tableau appelé Stockes de 500 éléments pour garder la liste de tous les produits dans ce magasin.
Au début de chaque mois, le magasin propose une remise de prix de 50% sur les produits qui expirent durant ce mois.

- Ecrire l'algorithme qui permet de gérer les prix des produits de ce magasin.

Exercice 4

Le nombre des employés dans une entreprise est de 300 employés, chaque employé est caractérisé par un numéro, un nom, un salaire, et par le nombre des années d'expérience.
À l'occasion de la rentrée sociale, l'entreprise décide d'ajouter un montant de 2000 DA aux salaires des employés dont l'expérience dépasse 3 ans, et un montant de 4000 DA aux salaires des employés dont l'expérience dépasse 5 ans
Chaque employé est représenté par un enregistrement, et la liste de tous les employées est stockée dans un tableau de 300 éléments.
On veut résoudre ce problème en utilisant les sous-programmes, pour cela, une procédure Ajouter qui permet de calculer et appliquer l'ajout au salaire d'un seul employé sera utilisée.

- Ecrire l'algorithme qui utilise le sous-programme Ajouter pour augmenter les salaires des employés de cette entreprise.

Exercice 5

Dans une école primaire, chaque élève est caractérisé par un numéro, un nom, une date de naissance, une année d'étude, et une moyenne générale.
Un tableau **Elèves_5** contient les informations de tous les élèves de la 5ème année primaire (120 élèves).

- En utilisant un sous-programme qui vérifie si un élève est admis ou non, écrire un algorithme permettant de calculer le taux de réussite des élèves de la 5ème année, ainsi que le numéro, le nom et la moyenne générale de l'élève ayant la meilleure moyenne.

Exercice 6

Dans un parking de véhicules, chaque véhicule doit payer 100 DA pour le stationnement pour la première heure, et paie 2 DA pour chaque minute supplémentaire, si le véhicule reste moins d'une heure, il doit payer 100 DA.

À la sortie de véhicule de parking, on donne au conducteur un ticket qui contient le numéro de place qui a été occupée par le véhicule, le temps d'entrée, le temps de sortie, et le prix à payer.

- Ecrire l'algorithme qui calcule le prix de ticket d'un véhicule.

Chapitre 8 : Les Structures de données dynamiques: Les pointeurs , les listes chaînées, les piles et les files

1. Les pointeurs

1.1. Définition

Un pointeur est une variable dont la valeur est une adresse mémoire

Un pointeur P pointe sur une variable dynamique notée P^

1.2. Variable dynamique

- Une variable dynamique peut être créée lors de l'exécution du programme.
- On lui alloue un espace mémoire, cet espace à une adresse mémoire qui sera affectée automatiquement au pointeur.
- Elle peut être détruite c-à-d l'espace mémoire qu'elle occupait est libéré
- L'accès à la valeur se fait à l'aide d'un pointeur.

Exemple

On donne l'algorithme qui utilise un pointeur à un entier, on doit lire sa valeur ; l'afficher, ensuit le libérer.

```
algorithme pointeur
Var     p: ^entier;
début
     nouveau  (p) ;
     lire (p^)  ;
    écrire(' la valeur de pointeur' , p^) ;
    Libérer ( p);
fin.
```

Remarque

Les variables dynamiques peuvent être manipulées à l'aide de pointeur comme les variables simples

Exemples

L'incrémentation de la valeur

p^←p^ +1 ;

L'affectation

x ←p^ ;

p^ ←y ;

Remarques importantes

1. Les deux opérations suivantes n'ont pas le même résultat :

p ←q : dans cette opération, l'adresse de q sera affectée à p, donc p et q pointent vers la même variable.

p^ ←q^ : dans cette opération, on affecte la valeur de la variable pointée par q à la variable pointée par p

2. Il faut d'abord crier la variable par l'opération **new** avant de lui attribuer une valeur .

3. Les variables dynamiques peuvent être de type simple ou complexe.

4. Une variable dynamique n'a pas de nom , on ne peut le manipuler qu'à travers un pointeur qui contient son adresse .

2. La liste chaînée

2.1. Définition

Une liste linéaire chainée est une structure de données permettant de représenter un ensemble de valeurs qui sont chainées entre elles formant une suite, chaque élément est appelé maillon.

C'est une structure qui contient un champ "suivant", qui représente un pointeur vers le maillon suivant.

L'adresse du 1er maillon est dite la tête de la liste, c'est une adresse importante qui doit être sauvegardée dans une variable pour manipuler la liste.

Le champ "suivant" de dernier maillon contient la valeur nil qui indique la fin de la liste .

2.2. Opérations pour la manipulation des listes

- **nouveau (p)** : permet de créer un nouveau maillon et affecter son adresse dans le pointeur p , les valeurs des champs sont encore indéterminé , donc il faut leur affecter des valeurs .
- **libérer (p)** : une procédure qui détruit le maillon pointé par p

2.3. Opération sur les listes :

1. Création d'une liste

On donne deux méthodes pour créer une liste de 4 éléments :

Méthode 1

Dans cette méthode, la liste est créée à partir du 1er élément vers le dernier, pour cela on utilise 3 pointeurs.

```
algorithme listes;
type élément= enregistrement
    val: entier;
    suivant: ^élément
fin;
```

```
var L, P, Q : ^élément;
i: entier;
début
     nouveau (P);
     Lire( P^.val);
     P^.suivant ← nil;
     L ← P;
    Q ← P;
    pour i ← 1 à 3
    faire
        nouveau(P);
        lire(P^.val);
        P^.suivant← nil;
        Q^.suivant ← P;
        Q ← P;
   fin pour
fin.
```

Méthode 2

Dans cette méthode, la liste est créée à partir du dernier élément vers le premier élément.

```
algorithme listes;
type élément= enregistrement
    val: entier;
    suivant: ^élément
fin;
var L, P: ^élément;
i: entier;
début
     L ← nil;
     pour i ← 1 à 4
     faire
        nouveau(P)
        lire(P^.val);
        P^.suivant←L;
        L ← P;
      fin pour
fin.
```

2. Afficher tous les éléments de la liste:

Dans cet algorithme la tête de la liste est L

```
algorithme affichage;
type élément= enregistrement
```

```
    val: entier;
    suivant: ^élément
fin;
var L, P: ^élément;
début
    P ← L;
    TQ P <> nil
    faire
       écrire( P^.val);
       P ←P^.suivant;
    fin TQ
fin.
```

3. Trouver une valeur dans la liste

Dans cet algorithme, la tête de la liste est L, et on suppose que la liste n'est pas vide

```
algorithme listes;
type élément= enregistrement
    val: entier;
    suivant: ^élément
fin;
var L, P: ^élément;
X: entier;
début
   Lire (X)
   P ←L;
   trouve ← faux
   TQ  P<> nil et non trouve
   faire
       si X = P^.val alors
            trouve ←vrai
            sinon
            P ← P^.suivant
       finsi
    finTQ
    si trouve = vrai alors
         écrire ('La valeur existe dans la liste')
         sinon
          écrire( 'La liste ne contient pas cette valeur');
     finsi
fin.
```

3. Les Files

3.1. Définition

Une file, ou file d'attente, est une liste chaînée où les éléments ne peuvent être ajoutés qu'à la fin de la file (la queue de la file), et ne peuvent ne peut être supprimés qu'à la tête de la file.

Il s'agit d'une structure de type FIFO (First In First Out). Les données sont retirées dans l'ordre où elles ont été ajoutées.

Pour éviter les parcours de toutes la file pour chaque ajout, on utilise deux pointeurs, un pointeur pour la tête, et un autre pointeur pour la queue, donc la queue sera directement accessible.

3.2. Déclaration d'une file

On donne le fragment du code qui permet de déclarer une file dont les éléments sont des valeurs entières.

```
algorithme Files;
type élément= enregistrement
    val: entier;
    suivant: ^élément
fin;
Type File= enregistrement
       T, Q : ^ élément;
fin;
Var f: file;
…..
……
…..
```

Dans cette déclaration nous avons créé une file f dont la tête est T et la queue est Q

3.3. Opérations pour la manipulation des files

- **Enfiler** : permet d'ajouter un élément à la queue de la file.
- **Défiler** : permet de supprimer un élément à la tête de la file.
- **Vérifier si la file est vide**.

Remarque: Si la file est vide, l'opération Défiler ne s'applique pas.

4. Les Piles

4.1. Définition

Une pile est une liste chaînée où les éléments ne peuvent être ajoutés qu'au sommet de la pile, et ne peuvent être retiré que du sommet.

Il s'agit d'une structure de type LIFO (Last In First Out). On ne travaille que sur le sommet de la pile.

4.2. Déclaration d'une pile

On donne le fragment du code qui permet de déclarer une pile dont les éléments sont des valeurs entières.

```
algorithme Piles;
type élément= enregistrement
    val: entier;
    suivant: ^élément
fin;
Type Pile= enregistrement
       Sommet : ^ élément;
fin;
Var P: Pile;
…..
……
…..
```

4.3. Opérations pour la manipulation des piles

- **Empiler** : permet d'ajouter un élément au sommet de la pile.
- **Dépiler** : permet de supprimer un élément au somment de la pile.
- **Vérifier si la pile est vide**.

Remarque: Si la pile est vide, l'opération Dépiler ne s'applique pas.

Série de TD 08

Exercice 1

1. Ecrire un algorithme qui possède un pointeur sur un tableau d'entier (utiliser la dimension de votre choix). L'algorithme doit remplir le tableau ensuite affiche la valeur max dans ce tableau.
2. Ecrire un sous programme qui permet d'afficher la valeur max du tableau de la question précédente en précisant ses paramètres.

Exercice 2

Soit une liste simplement chaînée d'entiers dont la tête est L, écrire l'algorithme (ou l'action paramétrée) qui permet de:

1. Créer cette liste.
2. Vérifier si la liste est vide ou non.
3. Ajouter un élément au début de la liste.
4. Ajouter un élément à la fin de la liste.
5. Ajouter un élément à la position N dans la liste, si cette position n'existe pas dans la liste, l'élément sera ajouté à la fin de la liste.
6. Supprimer un élément de la liste (suppression par valeur)
7. Supprimer un élément de la liste (suppression par position).
8. Calculer le nombre d'éléments dans la liste (La longueur de la liste).
9. Calculer le nombre d'occurrence d'une valeur X dans la liste.
10. Supprimer toutes les occurrences de la valeur X dans la liste.
11. Rendre cette liste une liste circulaire.
12. Rendre cette liste doublement chaînée.
13. Eclater la liste L en deux liste de telle sorte que les éléments pairs soient dans une liste L_pairs et les éléments impairs soient dans une liste L_impairs.

Chapitre 9 :
Les Fichiers

1. Introduction

Dans ce chapitre, nous allons découvrir la notion des fichiers d'enregistrements, et leurs avantages par rapport aux structures de données qu'on a déjà manipulé.

2. Les fichiers

2.1. Définition

Un fichier est un ensemble d'enregistrements de même structure relatifs à une même entité, et dont le nombre d'éléments peut varier d'un traitement à un autre.

2.2. Noms d'un fichier

Le fichier doit exister réellement sur un support de stockage (Disque dur, disquette, CD,…), il a donc un nom reconnu par le système d'exploitation, ce nom représente le nom physique du fichier.

La forme générale d'un nom physique est :

Nom de partition:\ nom d'un dossier\ nom du fichier.dat

Exemple

On peut trouver un fichier ayant le nom physique suivant :

C:\ Pascal \ TP \Mes_programmes \ Prog1.dat

La manipulation des noms de cette forme dans les programmes est lourde, pour cela, on utilise des noms logiques pour les fichiers. Un nom logique est une variable de type fichier déclarée dans le programme, et qui permet de représenter un fichier, ce nom peut varier d'un traitement à un autre.

Après la déclaration du nom logique d'un fichier, on doit faire une correspondance entre le nom logique et le nom physique par une instruction d'assignation.

2.3 Déclaration d'un fichier

Le fichier est un ensemble d'enregistrements, donc, avant de déclarer le fichier, on doit déclarer la structure des enregistrements qu'ils le constituent.

La forme générale de déclaration d'un fichier est :

```
Var nom du fichier : file of  enregistrements ;
```

Avec 'enregistrements' un type d'enregistrement déjà déclaré.

Exemple

On veut créer un fichier des étudiants, on aura donc la déclaration suivante :

```
Type  Etudiant = enregistrement
        Num : entier ;
        Nom, Prenom : Chaîne ;
        Année_étude : entier ;
Fin ;
Var  Section : File of Etudiant ;
```

2.4. Création d'un fichier

Pour créer un fichier pour la 1ère fois, on doit :

1. Déclarer une variable de type fichier dont le nom représente le nom logique du fichier.
2. Faire la correspondance entre le nom logique et le nom physique par l'instruction d'assignation.
3. Créer le fichier.

L'exemple suivant montre les étapes à suivre pour créer un fichier pour la 1ère fois :

```
Algorithme Création_Fichier;
Type  Etudiant = enregistrement
        Num : entier ;
        Nom, Prenom : Chaîne ;
        Année_étude : entier ;
Fin ;
Var  Section : File of Etudiant ;
Début
        Assign (Section, 'C:\ fichier.dat') ;
        Rewrite (Section) ;
        Close (Section) ;
Fin.
```

- L'instruction **Assign** permet d'établir la correspondance entre le nom logique (Section) et le nom physique ('C:\ fichier.dat').
- L'instruction **Rewrite** permet de créer le fichier, il est maintenant vide. L'utilisation de

cette instruction avec un fichier déjà existant va détruire son contenu.

- L'instruction **Close** permet de fermer le fichier.

2.5. Remplissage d'un fichier

On veut créer un fichier contenant des informations concernant les étudiants de la 1ère année, on doit donc écrire les informations dans le fichier, pour cela on doit utiliser une variable intermédiaire permettant l'écriture dans le fichier.

Le nombre des étudiants peut varier d'une section à une autre, on suppose que le nombre des étudiants est donné par l'utilisateur.

L'algorithme suivant montre bien les étapes à suivre pour remplir un fichier :

```
Algorithme Remplissage_Fichier;
Type  Etudiant = enregistrement
        Num : entier ;
        Nom, Prenom : Chaîne ;
        Année_étude : entier ;
Fin ;
Var  Section : File of Etudiant ;
     i, nbr : entier ;
     etd : Etudiant ;
Début
        Assign (Section, 'C:\ fichier.dat') ;
        Rewrite (Section) ;
        Lire (nbr) ;
        Pour i := 1 à nbr
        Faire
                Avec  Etd faire
                Début
                        Lire(num) ;
                        Lire (nom, prenom) ;
                        Lire( année_étude) ;
                Fin ;
                Write (Section, etd) ;
        Fin pour
        Close (Section) ;
Fin.
```

Le remplissage se fait comme suit :

1. Les informations d'un élément doivent être écrites d'abord dans la variable etd.
2. L'instruction write (Section, etd) permet de copier les valeurs de la variable etd dans le

fichier, autrement dit, cette instruction permet d'écrire la variable dans le fichier.

2.6. Lecture d'un fichier

On suppose qu'on veut écrire un algorithme permettant d'afficher tous les éléments d'un fichier, l'algorithme sera comme suit :

```
Algorithme Lecture_Fichier;
Type  Etudiant = enregistrement
        Num : entier ;
        Nom, Prenom : Chaîne ;
        Année_étude : entier ;
Fin ;
Var  Section : File of Etudiant ;
     etd : Etudiant ;
Début
        Assign (Section, 'C:\ fichier.dat') ;
        Reset (Section) ;
        TQ  non (eof (Section))
        Faire
                Read (Section, etd) ;
                Avec  Etd faire
                Début
                        Ecrire (num) ;
                        Ecrire (nom, prenom) ;
                        Ecrire ( année_étude) ;
                Fin ;
        Fin TQ
        Close (Section) ;
Fin.
```

- L'instruction **Reset** (section) permet d'ouvrir le fichier en mode lecture/écriture, cette instruction nous permet de manipuler des fichiers déjà existant sans supprimer leurs contenus.
- **Eof** (end of file) est une fonction prédéfinie dont le résultat est un booléen, eof est vrai

lorsqu'on atteint la fin du fichier. On utilise cette fonction car on veut afficher tous les éléments du fichier, et on ne connaît pas le nombre des éléments de ce fichier à l'avance.

- L'instruction **Read** (Section, etd) permet de copier le contenu de l'élément courant du fichier dans la variable etd, autrement dit, cette instruction permet de lire le contenu de l'enregistrement courant dans le fichier.

2.7. Remarques importantes

- Les enregistrements d'un fichier sont numérotés, et la numérotation commence à partir de 0.
- Le fichier a un pointeur qui pointe toujours sur l'élément courant dans le fichier.
- L'instruction rewrite et l'instruction reset ouvrent le fichier, dans ce cas le pointeur se trouve sur le 1er élément du fichier (l'élément numéro 0).
- L'instruction write permet d'écrire la variable dans le fichier, après l'écriture, le pointeur se déplace vers l'élément suivant.
- L'instruction read permet de lire le contenu de l'enregistrement courant dans le fichier, après la lecture, le pointeur se déplace vers l'élément suivant.

3. l'accès séquentiel

L'accès séquentiel consiste à accéder aux enregistrements du fichier élément par élément, pour accéder à un élément donné, il faut passer d'abord par tous les éléments qu'ils le précèdent.

C'est une méthode d'accès très lourde, mais elle est utile avec certains traitements qui nécessitent la lecture de tous les éléments d'un fichier.

4. L'accès direct

Pour accéder directement à un élément dans un fichier, on peut éviter l'accès séquentiel en utilisant l'instruction seek qui permet l'accès à un élément directement par son numéro.

La forme générale de cette instruction est :

```
Seek (nom du fichier, numéro de l'élément désiré) ;
```

Remarques

- Pour utiliser cette instruction, le fichier doit être déjà ouvert (en utilisant l'instruction reset).
- N'oubliez pas que la numérotation des éléments commence à partie du 0.

5. La taille d'un fichier

La taille d'un fichier représente le nombre d'élément de ce fichier, pour connaître la taille d'un fichier, on utilise la fonction prédéfinie filesize qui donne une valeur entière, et dont la forme générale est : filesize(nom du fichier)

6. exercice d'application

Ecrire un algorithme permettant d'ajouter un élément à la fin du fichier Section.

Pour ajouter un élément à la fin du fichier, on doit accéder directement à sa position, si le nombre d'éléments de ce fichier est n, le numéro du dernier élément dans ce fichier est donc n-1, on doit utiliser l'instruction seek, et on doit accéder à l'élément n (qui n'existe pas encore) pour mettre le nouvel élément. L'algorithme sera comme suit :

```
Algorithme Ajout_au_Fichier;
Type  Etudiant = enregistrement
        Num : entier ;
        Nom, Prenom : Chaîne ;
        Année_étude : entier ;
Fin ;
Var  Section : File of Etudiant ;
     etd : Etudiant ;
     taille : entier ;
Début
        Assign (Section, 'C:\ fichier.dat') ;
        Reset (Section) ;
        taille := filesize (Section) ;
        Avec  Etd faire
        Début
                Lire (num) ;
                Lire (nom, prenom) ;
                Lire ( année_étude) ;
        Fin ;
        Seek (Section, taille) ;
        Write (Section, etd) ;
        Close (Section) ;
```

Fin.

Série de TD 09

Exercice 1

3. Donner la structure de l'enregistrement « Relevé-de-notes » pour un étudiant en 1ère année.
4. Ecrire un algorithme permettant d'afficher les moyennes générales et le taux de réussite des étudiants d'une section qui contient au maximum 150 étudiants.

Exercice 2

On veut créer un annuaire téléphonique qui peut contenir 500 contacts, un contact peut être un numéro du téléphone fixe, portable ou fax.

1. Donner la structure de l'enregistrement contact.
2. Ecrire un algorithme permettant d'afficher tous les numéros des fax dans cet annuaire.
3. Ecrire un algorithme permettant de modifier le numéro du téléphone d'un contact.

Exercice 3

1. Donner la structure de l'enregistrement « Employé » qui contient toutes les informations concernant un employé dans une entreprise.
2. Ecrire un algorithme permettant d'afficher les noms des employés ayant une durée de service qui dépasse 10 ans (5000 employés).
3. Ecrire un algorithme permettant d'afficher le nom de l'employé le plus âgé dans cette entreprise.

Exercice 4

Dans une entreprise privée, les employés ont des salaires déjà fixés, et ils peuvent travailler des heures supplémentaires. L'employé bénéficie de 100 dinars de plus pour chaque heure supplémentaire, et le nombre des heures supplémentaires pendant un mois est calculé à la fin du mois pour ajouter le montant au salaire de l'employé.

1. Donner la structure de l'enregistrement **Employé.**
2. En utilisant un fichier de 1500 éléments contenant tous les employés de cette entreprise, écrire l'algorithme qui permet de calculer les salaires de tous les employés.

Conclusion

Conclusion

Au terme de cette exploration des notions de base de l'algorithmique, il devient évident que l'algorithmique ne se limite pas à un simple ensemble d'outils ou de techniques, mais qu'elle constitue un cadre fondamental pour comprendre la résolution de problèmes informatiques de manière structurée et efficace. Les éléments abordés tout au long de ce document, tels que les **structures de données simples**, les **entrées/sorties**, les **instructions conditionnelles**, les **boucles**, les **tableaux**, les **sous-programmes**, les **enregistrements** et les **structures de données dynamiques**, représentent les éléments constitutifs de tout algorithme. Chacun de ces concepts joue un rôle clé dans la formulation, l'optimisation et l'implémentation de solutions informatiques efficaces et fiables.

Les structures de données simples et les **entrées/sorties** sont les premières briques sur lesquelles repose toute application algorithmique. Les structures de données simples, telles que les entiers, les réels, ou les caractères, sont utilisées pour représenter les informations de manière élémentaire. Ces données peuvent être manipulées à travers des opérations simples mais essentielles comme l'affectation, l'addition, ou la comparaison. Les **entrées/sorties**, quant à elles, permettent d'interagir avec l'environnement extérieur de l'algorithme, en récupérant des informations de l'utilisateur ou en affichant des résultats. Bien que souvent considérées comme des aspects secondaires de l'algorithmique, la gestion des entrées et des sorties est cruciale pour toute application, car elle permet de relier l'algorithme à un contexte réel, souvent en interaction avec des utilisateurs ou des systèmes externes.

Un autre concept fondamental traité est celui des **instructions conditionnelles**. Ces instructions, comme les structures de type **if-else**, permettent de guider le flux d'exécution d'un programme en fonction des conditions rencontrées au cours de l'exécution. Elles sont essentielles pour la prise de décisions au sein de l'algorithme. Sans elles, l'algorithme serait incapable d'adapter ses actions à des situations variées. Elles permettent ainsi une grande flexibilité dans la résolution des problèmes, en ouvrant la voie à des solutions dynamiques qui s'ajustent en fonction des données d'entrée. Apprendre à maîtriser les instructions conditionnelles est donc fondamental pour construire des algorithmes capables de résoudre des problèmes plus complexes de manière optimale.

Les **boucles** et les **tableaux** sont également au cœur de l'algorithmique moderne. Les boucles, comme les boucles **for** ou **while**, permettent de répéter des opérations de manière itérative, ce qui est crucial pour traiter des ensembles de données ou pour réaliser des calculs répétitifs. Elles permettent, par exemple, d'implémenter des algorithmes de recherche, de tri, ou d'itération sur des structures de données comme les tableaux. Les tableaux, quant à eux, sont des structures de données permettant de stocker plusieurs éléments de même type dans une séquence continue en mémoire. Ils sont utilisés pour gérer des collections d'éléments de manière efficace et sont particulièrement utiles pour traiter des ensembles de données, comme dans le cas des algorithmes de tri ou de recherche. Maîtriser l'utilisation des boucles et des tableaux permet de développer des algorithmes capables de traiter rapidement et efficacement des volumes de données importants.

Les **sous-programmes** représentent également une notion centrale en algorithmique. Un sous-programme permet de diviser un problème complexe en sous-problèmes plus simples, favorisant ainsi la réutilisabilité, la modularité et la lisibilité du code. En effet, les sous-programmes permettent de décomposer les algorithmes en unités fonctionnelles qui peuvent être testées, déboguées et améliorées indépendamment. Cette approche, essentielle dans le développement logiciel, permet de créer des solutions évolutives et maintenables. De plus, la notion de sous-programme permet d'aborder des concepts avancés tels que les **paramètres** et le **passage de valeur**, qui sont indispensables pour passer des informations entre différentes parties du programme.

Les **enregistrements** et les **structures de données dynamiques** sont des notions plus avancées mais tout aussi importantes dans la construction de solutions algorithmiques efficaces. Un **enregistrement** est une structure qui permet de regrouper plusieurs éléments de types différents sous un même nom, offrant ainsi une méthode pour manipuler des objets complexes. Par exemple, dans la gestion des informations d'un étudiant, un enregistrement pourrait contenir des champs pour le nom, l'âge, la note, et l'adresse, permettant ainsi d'organiser de manière cohérente des données hétérogènes. De même, les **structures de données dynamiques**, telles que les listes chaînées, les piles ou les files d'attente, sont des structures qui permettent de gérer des ensembles de données dont la taille peut évoluer au fur et à mesure de l'exécution du programme. Ces structures sont particulièrement utiles pour les situations où l'on ne connaît pas à l'avance la taille des données à traiter, ou lorsqu'il est nécessaire d'effectuer des insertions et suppressions d'éléments de manière flexible.

À travers l'étude de ces différents concepts, nous avons vu qu'ils ne sont pas seulement des éléments théoriques, mais des outils qui permettent de résoudre une grande variété de problèmes pratiques. La capacité à choisir la structure de données appropriée et à utiliser efficacement les boucles, les conditions et les sous-programmes permet de créer des algorithmes à la fois élégants et performants. Ces connaissances permettent également d'acquérir un raisonnement algorithmique structuré, qui sera essentiel dans la résolution de problèmes plus complexes.

Enfin, il est essentiel de souligner que l'algorithmique est un domaine en constante évolution. Si les notions abordées dans ce document constituent les bases indispensables à tout informaticien, elles permettent d'élargir les horizons des étudiants, tout en leur offrant des outils puissants pour relever les défis informatiques du futur.