Chapitre IV

Les Tableaux et les Matrices

1. Introduction

Pour comprendre l'utilité et le besoin des tableaux, prenons l'algorithme de calcule de moyenne du module INF I pour une classe qui contient 20 étudiants.

```
Algorithme INFI ;
Const
        CoefExam= 2 ;
        CoefEval= 1 ;
Var
        i : entier ;
        NotExam, NotEval, Moy : réel ;
Début
        Pour i allant de 1 à 20 faire
                Ecrire ('L'étudiant N° ', i) ;
                Ecrire ('Donnez la note d'examen : ') ;
                Lire (NotExam) ;
                Ecrire ('Donnez le note d'évaluation : ')
                Lire (NotEval) ;
                Moy ← (NotExam * CoefExam + NotEval * CoefEval)/ (CoefExam +CoefEval) ;
                Ecrire ('La moyenne de l'étudiant ', i, ' = ', Moy : 5 :2) ;
        Finpour ;
Fin.
```

On veut maintenant rajouter des instructions à l'algorithme pour qu'il affiche les 20 étudiants par ordre décroissant (de la plus grande moyenne jusqu'à la plus petite).

Pour faire on a besoin de toutes les moyennes calculées des 20 étudiants, or que la variable *"Moy"* ici ne mémorise qu'une seule valeur (la dernière valeur calculée).

Un premier réflexe consiste à déclarer autant de variables *"Moy"* que d'étudiants; *Moy1*, *Moy2,…* *Moy20*, dans ce cas, on peut même s'en passer de la boucle et on écrira autant de blocs d'instructions que de nombre d'étudiants. Ce qui donne un algorithme de mauvaise qualité !

La solution est qu'on doit utiliser une structure de données capable de mémoriser les moyennes dans un seul endroit, pour les retrouver ultérieurement et effectuer le traitement souhaité.
Une telle structure est appelée **Tableau**.

2. Définition

Un tableau est une structure de données permettant de regrouper sous un même nom plusieurs valeurs de même type. Il y a deux types de tableaux :

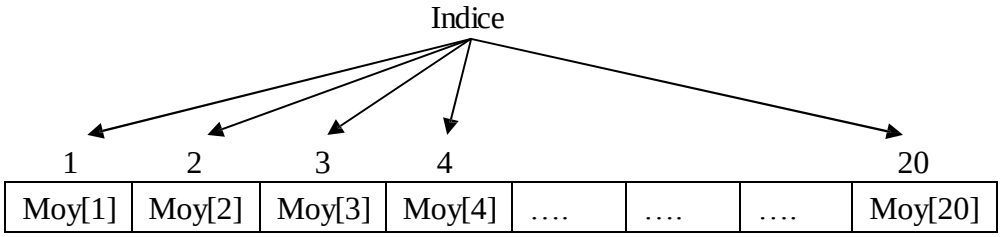
- les tableaux à une dimension appelés également Vecteurs,
- les tableaux à deux dimensions appelés également Matrices.

3. Tableau à une dimension (vecteur)

Un tableau à une dimension est caractérisé par :

- Son nom : un identificateur unique pour tous les éléments
- Sa taille : le nombre d'éléments (cases du tableau) qui est constant et doit être connu à l'avance.
- Son indice : l'accès à un élément du tableau se fait à l'aide d'un indice (N° de case).

Exemple :



Nom du tableau : Moy

Taille du tableau : 20

Indice du tableau : 1, 2, 3, ….20

3.1 Déclaration d'une variable Tableau :

Var

Nom_tableau : Tableau [Début.. Fin] de Type_élément ;

Indice : entier ;

En Pascal:

Var

Nom_tableau : array [Début.. Fin] of <Type_élément> ;

Indice : integer ;

Nom_tableau : un identificateur choisi par le programmeur.

Début et **Fin** : entiers ou caractères, Début est généralement égal à 1 et Fin à la taille du tableau.

Type_élément : le type des valeurs du tableau (elles ont toutes le même type) Réel, Entier, Caractère, Chaine de caractères, Booléen.

Exemple : on veut déclarer un tableau de moyennes

Variables Moy : Tableau [1..20] de réel

i : entier (* indice du tableau Moy *)

En mémoire, on va avoir une variable Moy comportant 20 cases et dans chacune on placera une moyenne qui est de type réel.

Soit l'ensemble des cases mémoires MOY

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
10.5	08.2	09.7	11.0	10.5															13.6

- Le premier élément de Moy contient la valeur 10.5 ; on note Moy [1] =10.5
- Le deuxième élément de Moy contient la valeur 08.2 ; on note Moy [2] =08.2
- D'une manière générale, le $i^{ème}$ élément de Moy est noté Moy[i] et contient une valeur de type réel. Moy[i] désigne également le contenu de la $i^{ème}$ case du tableau Moy.
- Moy[8] ← 15, veut dire affecter la valeur 15 au 8ème élément.
- Moy [1] ← Moy [3], veut dire affecter la valeur du 3ème élément au 1er élément, donc Moy [1] =09.7.

3.2 Opérations de base sur un tableau :

Pour représenter ces différentes opérations, on va utiliser un tableau T d'entiers.

Initialiser un Tableau

```
Algorithme Ini_Tab ;
Const
 N = 20 ;
Var
T : Tableau [1..N] de Entier ;
i : entier ;
Début
Pour i allant de 1 à N faire T[i] ← 0 ;
FinPour ;

Fin.
```

En Pascal:

```
Program
 Ini_Tab; Const  N=20;

Var
T: Array [1..N] of Integer; i: integer;
Begin

For i: =1 to N do T[i]:= 0;

End.
```

Lire (Remplir) un Tableau

```
Algorithme LecTab ;
Const N = 20 ;
Var
 T : Tableau [1..20] de Entier ;
i : entier :
Début
Pour i allant de 1 à N faire Lire(T[i]) ;
FinPour ;
Fin.
```

En Pascal:

```
Program Lec _Tab;
Const    N=20;
Var      T: Array [1..N] of Integer;
         i: integer;
Begin
For i: =1 to N do Readln (T[i]);
End.
```

Remarque : Lire(T [i]) consiste à entrer une valeur à partir du clavier et la mémoriser dans la $i^{ème}$ case du tableau T.

Afficher un Tableau

On va afficher les éléments strictement positifs du tableau T

```
Algorithme Aff_Tab ;
Const N = 20 ;
Var
 T : Tableau [1..20] de Entier ;
i : entier :
Début
Pour i allant de 1 à N faire Lire(T[i]) ;
FinPour ;
Pour i allant de 1 à N faire
Si T[i] > 0 Alors
        Ecrire (T[i]) ;
Finsi ;
FinPour ;
Fin.
```

En Pascal:

```
Program AFF_Tab;
Const   N=20;
Var     T: Array [1..N] of Integer;
        i: integer;
Begin

For i: =1 to N do Readln (T[i]);
For i: =1 to N do
    If T[i] > 0 Then Writeln (T[i]);

End.
```

Additionner les éléments de deux tableaux

On dispose de deux tableaux T1 et T2 et on veut réaliser la somme. Ça consiste à additionner les éléments de même indice et les mémoriser dans un tableau T3.

Condition nécessaire : T1 et T2 de même taille et leurs éléments de même type.

```
Algorithme Add_Tab ;
Const N = 20 ;
Var
 T1,T2,T3 : Tableau [1..20] de Entier ;
i : entier :
Début
Pour i allant de 1 à N faire Lire(T1[i]) ;
Pour i allant de 1 à N faire Lire(T2[i]) ;
FinPour ;
Pour i allant de 1 à N faire T3[i] ← T1 [i] + T2 [i] ;
FinPour ;
Pour i allant de 1 à N faire Ecrire(T3[i]) ;
Fin.
```

Exercice 1 :

Ecrire un algorithme qui :

- Lit un tableau d'entier de taille 10.
- Calcule et affiche la somme et la moyenne des éléments de ce tableau.
- Affiche le nombre d'éléments positifs et le nombre d'éléments négatifs contenus dans ce tableau.

Exemple :

	1	2	3	4	5	6	7	8	9	10
T	**13**	**-6**	**42**	**-8**	**-7**	**7**	**0**	**12**	**20**	**-30**

- La somme = (13-6+42-8-7+7+0+12+20-30) = 43
- La moyenne = 43 /10 = 4,3
- Le nombre d'éléments positif = 6
- Le nombre d'éléments négatifs = 4

Exercice 2 :

Ecrire un algorithme qui lit un tableau d'entier de taille 10 et affiche les positions (indices) des éléments inférieur au premier élément du tableau et double leurs valeurs.

Exemple :

	1	2	3	4	5	6	7	8	9	10
T=	13	-6	42	-8	-7	7	0	12	20	-30

- L'élément 2 est inférieur au premier élément
- L'élément 4 est inférieur au premier élément
- L'élément 5 est inférieur au premier élément
- L'élément 6 est inférieur au premier élément
- L'élément 7 est inférieur au premier élément
- L'élément 8 est inférieur au premier élément

Exercice 3 :

Ecrire un algorithme qui lit un tableau d'entier de taille 10 et affiche le plus grand et le plus petit élément du tableau avec leurs positions (indices).

Exemple :

	1	2	3	4	5	6	7	8	9	10
T=	13	-6	42	-8	-7	7	0	12	20	-30

- Le plus grand élément est 42 et sa position est 3
- Le plus petit élément est -30 et sa position est 10

3.3 La recherche séquentielle dans un tableau :

A) On veut rechercher une valeur lue à partir du clavier si elle existe dans un tableau et afficher sa position.

Exemple : On veut rechercher si la valeur 7 se trouve dans le tableau et afficher sa position.

	1	2	3	4	5	6	7	8	9	10
T	**13**	**-6**	**7**	**-8**	**-7**	**7**	**0**	**12**	**20**	**-30**

- Le nombre 7 existe dans le tableau à la position 3.
- Le nombre 7 existe dans le tableau à la position 6.

B) Supposant maintenant que le nombre recherché est 13, on va parcourir le tableau du début jusqu'à la fin inutilement puisque le nombre 13 se trouve juste dans la première position.

On veut réécrire l'algorithme de recherche pour qu'il s'arrête dès qu'il trouve le nombre recherché et affiche un message d'existence avec la première position trouvée. Pour le faire, on doit utiliser une variable booléenne 'Trouvé' qui sera initialisée à faux et devient vrai dès qu'on trouve le nombre recherché.

Exercice :

On dispose d'un tableau T de 20 entiers, écrire un algorithme qui permet d'afficher :

- Le nombre d'apparition d'une valeur entière X donnée dans le tableau T.
- La position de la première apparition de X,
- La dernière position d'apparition de X.

3.4 Le tri simple d'un tableau :

Il s'agit de représenter les éléments d'un tableau T dans un ordre croissant ou décroissant.

Le principe est le suivant :

1. Parcourir le tableau du premier élément jusqu'à l'avant dernier élément
2. Pour chaque élément i du tableau rechercher depuis l'élément i +1 la première valeur plus petite et les permuter, notez que pour permuter deux éléments on a besoin d'une variable temporaire :

   Temp ←T[i]
   T[i] ← T[j]
   T[j] ← Temp

3. Refaire la même procédure avec les autres éléments jusqu'à N, automatiquement la valeur la plus petite se retrouve au début du tableau
4. Passer à l'élément i + 1 et rechercher la première valeur plus petite depuis i + 2 jusqu'à N et ainsi de suite jusqu'à l'élément N-1

Exemple : On veut trier le Tableau T de taille 5 par ordre croissant. On suppose que le tableau en question est déjà rempli.

	1	2	3	4	5
T	**13**	**-6**	**7**	**-8**	**-7**

Pour i =1

13	**-6**	**7**	**-8**	**-7**

Rechercher une valeur < 13 à partir du 2ème élément et les permuter, on la trouve dans la deuxième position.

-6	**13**	**7**	**-8**	**-7**

Maintenant rechercher une valeur < -6 à partir du 3ème élément et les permuter, on la trouve dans la quatrième position.

-8	**13**	**7**	**-6**	**-7**

Maintenant rechercher une valeur < -8 à partir du 4ème élément et les permuter, on la trouve pas.

Le premier élément est trié et se trouve au début du tableau, passant maintenant au deuxième élément

Pour i =2

-8	**13**	**7**	**-6**	**-7**

Rechercher une valeur < 13 à partir du 3ème élément et les permuter, on la trouve dans la troisième position.

-8	**7**	**13**	**-6**	**-7**

Rechercher une valeur < 7 à partir du 4ème élément et les permuter, on la trouve dans la quatrième position.

-8	**-6**	**13**	**7**	**-7**

Rechercher une valeur < -6 à partir du 5ème élément et les permuter, on la trouve dans la dernière position.

-8	**-7**	**13**	**7**	**-6**

Le deuxième élément est trié et se trouve en deuxième position du tableau, passant maintenant au Troisième élément

Pour i =3

-8	**-7**	**13**	**7**	**-6**

Rechercher une valeur < 13 à partir du 4ème élément et les permuter, on la trouve dans la quatrième position.

-8	**-7**	**7**	**13**	**-6**

Rechercher une valeur < 7 à partir du 5ème élément et les permuter, on la trouve dans la dernière position.

-8	**-7**	**-6**	**13**	**7**

Le troisième élément est trié et se trouve en troisième position du tableau, passant maintenant au quatrième et dernier élément

Pour i =4

-8	**-7**	**-6**	**13**	**7**

Rechercher une valeur < 13 à partir du 5ème élément et les permuter, on la trouve dans la dernière position.

-8	**-7**	**-6**	**7**	**13**

Maintenant tous les éléments du tableau T sont triés.

```
Algorithme Tri1 ;
Const N=20 ;
Var
 T : Tableau [1..N] d'Entier ;
i, j, Temp : entier ;
Début
(* Lire les données*)
    Pour i allant de 1 à N faire
    Lire (T [i]) ;
    FinPour ;
    (* trier le tableau*)
    Pour i allant de 1 à N-1 faire
      Pour j allant de i+1 à N faire
        Si T[i]>T[j] alors
              Temp ←T[i] ;
             T[i] ←T[j] ;
             T[j] ←Temp ;
      Finsi ;
      FinPour ;
Finpour ;
(*Afficher le tableau trier *)
    Pour i allant de 1 à N faire
    Ecrire (T [i]) ;
    FinPour ;
Fin.
```

4. Tableau à deux dimensions (Les matrices) :

Un tableau à deux dimensions appelé également matrice permet d'organiser des informations de même type en lignes et en colonnes. Il est caractérisé donc par :

- Son nom : un identificateur unique pour tous les éléments
- Son nombre de lignes et son nombre de colonnes qui doivent être connus à l'avance.
- Ses deux indices : l'accès à un élément de la matrice se fait à l'aide de deux indices (N° de ligne et N° de colonne).

4.1 Déclaration d'une variable matrice :

Variables Nom_matrice : Tableau [1..Nbr_Lig, 1..Nbr_Col] de Type_élément
i, j : Entier (* i étant l'indice des lignes et j celui des colonnes *)

En Pascal:

Var Nom_matrice: array [1..Nbr_Lig, 1..Nbr_Col] of Type_élément;
i, j: integer;

Nbr_Lig: Nombre de lignes connu.
Nbr_Col: Nombre de colonnes connu.
Un élément de la matrice est identifié par son numéro de ligne et son numéro de colonne.
Si M est la matrice, M [i, j] désigne l'élément de M situé à la $i^{ème}$ Ligne et à la $j^{ème}$ Colonne.

Exemple :

Var M : Tableau [1..2, 1..3] de réel
i, j : entier

$$\mathbf{M} \begin{pmatrix} 0.1 & 1 & 2.1 \\ 5 & 3.8 & 0.9 \end{pmatrix}$$

En Pascal:

Var M: array [1..2, 1..3] of Real;
i, j: integer;

- M est une matrice de 2 lignes et 3 colonnes, en MC un espace de (2 x 3) de réels va être réservé.
- M [1 ,1] = 0.1 M [1 ,2] = 1 M [1 ,3] = 2.1 M [2 ,1] = 5 M [2 ,2] = 3.8 M [2 ,3] = 0.9
- M [2 ,1] ← 0 veut dire affecter à l'élément qui se trouve à la 2ème ligne et 1ère colonne la valeur 0.
- X ← M [2 ,3] veut dire affecter à X la valeur qui se trouve dans la 2ème ligne et la 3ème colonne (X = 0.9).

4.2 Opérations de base sur une matrice :

Pour représenter ces différentes opérations, on va utiliser une matrice M d'entiers.

Initialisation d'une Matrice

```
Algorithme Init_Mat;
Const
   NL = 10;
   NC = 20 ;
Var
T : Tableau [1..NL,1..NC] d' Entier ;
I,j : entier ;
Début
      Pour i allant de 1 à NL faire
          Pour j allant de 1 à NC faire
          M [i, j] ← 0 ;
         FinPour ;
     FinPour ;

Fin.
```

En Pascal:

```
Program Init_MAT;
Const
NL=10;
NC=20
Var
T: Array [1..NL,1..NC] of Integer;
i: integer;
Begin

For i: =1 to NL do
   For j=1 to NC do T[i,j]:=0;

End.
```

Lecture d'une Matrice

```
Algorithme Lec_Mat;
Const
   NL = 10;
   NC = 20 ;
Var
T : Tableau [1..NL,1..NC] d' Entier ;
I,j : entier ;
Début
      Pour i allant de 1 à NL faire
          Pour j allant de 1 à NC faire
          Lire(M [i, j]);
         FinPour ;
     FinPour ;

Fin.
```

En Pascal:

```
Program Lec_MAT;
Const
NL=10;
NC=20
Var
T: Array [1..NL,1..NC] of Integer;
i: integer;
Begin

For i: =1 to NL do
   For j=1 to NC do read(T[i,j]);

End.
```

Affichage d' une Matrice

On va afficher les éléments strictement positifs de la matrice M

```
Algorithme Aff_Mat;
Const
   NL = 10;
   NC = 20 ;
Var
T : Tableau [1..NL,1..NC] d' Entier ;
I,j : entier ;

Début
                    (* remplir la matrice*)
      Pour i allant de 1 à NL faire
          Pour j allant de 1 à NC faire

          Lire(M [i, j]);
         FinPour ;
     FinPour ;

            (* affichage des valeurs positives*)

Pour i allant de 1 à NL faire
  Pour j allant de 1 à NC faire
      Si M [i, j] > 0
        Alors Ecrire (M [i, j]);
      Fsi ;
  FinPour ;
 FinPour ;

Fin.
```

En Pascal:

```
Program Aff_MAT;
Const
NL=10;
NC=20
Var
T: Array [1..NL,1..NC] of Integer;
i: integer;
Begin

For i: =1 to NL do
   For j=1 to NC do read(T[i,j]);

For i: =1 to NL do
begin
   For j=1 to NC do
      If M [i, j] > 0
        Alors write (T[i,j]);
Writeln;
End;

End.
```

Additioner les éléments de deux tableaux

On dispose de deux matrices M1 et M2 et on veut réaliser la somme. Ça consiste à additionner les éléments de mêmes indices et les mémoriser dans une matrice M3.

Condition nécessaire : M1et M2 de même taille et leurs éléments de même type.

```
Algorithme ADD_Mat;
Const
   NL = 10;
  NC = 20 ;
Var
Mat1,Mat2 : Tableau [1..NL,1..NC] d' Entier ;
I,j : entier ;

Début
(* remplir la matrice Mat1*)
      Pour i allant de 1 à NL faire
          Pour j allant de 1 à NC faire

          Lire(Mat1 [i, j]);
          FinPour ;
     FinPour ;

(* remplir la matrice Mat2*)
      Pour i allant de 1 à NL faire
          Pour j allant de 1 à NC faire

          Lire(Mat2 [i, j]);
          FinPour ;
     FinPour ;

(* addition des deux matrices *)

      Pour i allant de 1 à NL faire
          Pour j allant de 1 à NC faire

          Mat3[i, j] ←Mat1[i,j]+Mat2[i,j];
          FinPour ;
     FinPour ;

(* affichage de la matrice Résultat Mat3*)

Pour i allant de 1 à NL faire
  Pour j allant de 1 à NC faire

 Ecrire (Mat3 [i, j]);

   FinPour ;
 FinPour ;

Fin.
```

Exercice 1

Ecrire un algorithme qui réalise la Multiplication de deux matrices

Exercice 2 :

Ecrire un algorithme qui :

- Lit une matrice de 5 lignes et 5 colonnes d'entiers.
- Calcule et affiche la somme des chiffres pairs et des chiffres impairs.
- Calcule et affiche le nombre des chiffres positifs et le nombre des chiffres négatifs et le nombre des 0.

Exercice 3 :

Ecrire un algorithme qui lit une matrice de 5 lignes et 4 colonnes puis cherche si une valeur entière X lue à partir du clavier existe dans la matrice et afficher sa position dans le cas contraire il affiche le message « La valeur X n'existe pas dans la matrice ».