

Chapitre I

Type Tableau – Vecteurs & Matrices

Sommaire

Chapitre III : Variables Indicées – Le type Tableau.....	3
I.1. Introduction.....	3
I.2. Les tableaux unidimensionnels (Vecteurs).....	3
I.2.1. Accès à un élément du tableau.....	4
I.2.2. Représentation en mémoire.....	4
I.2.3. Déclaration d'un tableau unidimensionnel.....	5
I.2.4. Initialisation d'un tableau unidimensionnel.....	6
I.2.5. Manipulation de Tableaux unidimensionnels (Vecteurs).....	7
a) Somme de deux Vecteurs V1 et V2.....	8
b) Produit scalaire de deux vecteurs V1 et V2.....	8
c) Recherche d'un élément dans un tableau.....	9
d) Recherche du maximum dans un tableau unidimensionnel.....	9
e) L'algorithme de Tri par sélection (permutations).....	10
I.3. Les tableaux bidimensionnels (Matrices).....	11
I.3.1. Présentation de tableaux bidimensionnels.....	11
I.3.2. Déclaration de tableaux bidimensionnels.....	11
I.3.2. Lecture et affichage d'une matrice.....	12
I.3.3. Manipulation de matrice.....	13
a) Lire et écrire un tableau bidimensionnels.....	13
b) Produit d'une matrice par un vecteur.....	13
c) Compter le nombre d'éléments négatifs, positifs et nuls dans une matrice.....	14
d) Produit de deux matrices.....	15

Cours Elearning :

<https://elearning.univ-bejaia.dz/course/view.php?id=14316#section-6>

Page facebook :

<https://www.facebook.com/InitiationAlgoProgrammation/>

La chaîne Youtube :

<https://www.youtube.com/@algo-prog>

La playlist sur le langage C :

<https://youtube.com/playlist?list=PLwHHAvorm5F-tL9EXDEHomiOKmAj7iUTU>

Adapté par : Redouane OUZEGGANE
rouzeggane@gmail.com - redouane.ouzeggane@univ-bejaia.dz

Chapitre III : Variables Indicées – Le type Tableau

1.1. Introduction

Jusque là, nous avons vu que des variables de type simple (entier, réel, caractère, chaîne et booléen) où chaque variable est associée à un seul espace mémoire, ne pouvant loger (ou bien contenir) qu'une seule valeur à instant donné. Afin de représenter des données complexes comme des tableaux de données (Tableaux statiques, vecteurs, matrices, listes, etc.) on utilise un autre type de variables : *variables indicées* ou *tableaux*.

Exemple

Représentation d'un vecteur : soit $VECT = [c1 \quad c2 \quad c3 \quad c4 \quad c5]$

Ce vecteur sera représenté par le tableau (variable) $VECT$ comme suit :

0	1	2	3	4
c1	c2	c3	c4	c5

Avec $VECT[1] = c1$; $VECT[2] = c2$; $VECT[3] = c3$; $VECT[4] = c4$; $VECT[5] = c5$.

Les positions {1, 2, 3, 4, 5} représentent les *indices ou indexes du tableau*. Ils donnent la position d'un élément du tableau. $VECT[1]$, $VECT[2]$, $VECT[3]$, $VECT[4]$, $VECT[5]$ représentent les composantes du vecteur $VECT$, (les éléments du tableau).

$VECT[0]$ est la première composante du tableau $VECT$

$VECT[1]$ est la deuxième composante du tableau $VECT$

$VECT[2]$ est la troisième composante du tableau $VECT$

Etc.

1.2. Les tableaux unidimensionnels (Vecteurs)

Les tableaux à une seule dimension correspondent au tableaux avec une seule plage d'indices. C'est-à-dire, pour accéder à une composante on a besoin d'un seul indice (valeur entière). Autrement dit : les cases sont repérées avec un seule indice qui représente une valeur entière (valeur immédiate, constante ou variable entière, ou expression donnant un résultat entier).

I.2.1. Accès à un élément du tableau

En programmation un élément de tableau est désigné par le nom du tableau suivi de l'indice (la position) de l'élément dans le tableau. Soit :

<Nom du Tableau>[<Indice de l'élément>]

Exemple :

Soit deux Tableaux *V1* et *V2* où *V1* contient 4 composantes et *V2* contient 6 composantes :

V1 :

<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>
<i>12.50</i>	<i>25.75</i>	<i>56.00</i>	<i>60.25</i>

V2 :

<i>0</i>	<i>1</i>	<i>2</i>	<i>4</i>	<i>3</i>	<i>4</i>
<i>24.00</i>	<i>38.25</i>	<i>28.00</i>	<i>70.25</i>	<i>63.00</i>	<i>96.25</i>

On a :

$V1[0] = 12.55$; $V1[1] = 25.75$; $V1[2] = 56.00$; $V1[3] = 60.25$

$V2[0] = 24.00$; $V2[1] = 38.25$; $V2[2] = 28.00$; $V2[3] = 70.25$; $V2[4] = 63.00$; $V2[5] = 96.25$;

I.2.2. Représentation en mémoire

Dans la mémoire centrale de l'ordinateur (RAM), un tableau est représenté sous forme d'une succession de cellules mémoires (cases mémoires) contiguës (l'une à la suite de l'autre). L'adresse d'un élément de tableau est obtenue par le système en utilisant l'adresse d'implantation du tableau en mémoire auquel est ajouté la valeur de la position de l'élément par rapport au début du tableau (*l'offset de l'élément*). Pour les deux tableaux *V1* et *V2*, on aura cette représentation en mémoire :

<i>Adresses</i>	<i>Mémoire</i>
0000	
0001	
0002	
.....	
.....	

.....	
V1 → 0006	12.50
0007	25.75
0008	56.00
0009	60.25
.....

V2 → 0014	24.00
0015	38.25
0016	28.00
0017	70.25
0018	63.00
0018	96.25
.....	
.....	

I.2.3. Déclaration d'un tableau unidimensionnel

La syntaxe à suivre pour déclarer un tableau est la suivante :

Syntaxe Algorithmique

constante

MAX = <valeur_entière>

variable

<id_tab>:Tableau[0..MAX-1] de <type>;

Syntaxe en C

const int MAX = <valeur_entière>;

<type> <id_tab>[MAX];

Remarques :

- x Ce n'est pas obligatoire d'utiliser une constante pour la taille maximale de tableau ; cependant, c'est une bonne pratique dans la programmation
- x La plage d'indice 0 .. MAX-1 représente l'indice du premier élément 0 et l'indice du dernier élément MAX-1.
- x C'est pas obligatoire d'utiliser toutes les composantes du tableaux, pour cela on déclare une variable entière N qui représente la taille du tableau à utiliser. La valeur de cette variable

sera introduite au cours de l'exécution (par lecture)

Exemple :

```

constante                                const int MAX = 50;
    MAX = 50
variable
    V:Tableau[0..MAX-1] de réel;           real V[MAX];

```

V est un tableau de 50 composantes réelles (c'est comme si on a déclaré 50 variables réelles)

1.2.4. Initialisation d'un tableau unidimensionnel

Un tableau peut être initialisé en utilisant deux méthodes :

- En utilisant des affectations ;
- En utilisant la lecture à partir d'un fichier de données ou à partir de clavier.

a) *Par affectations* : On utilise pour cela l'opération d'affectation.

Exemple :

```
V[0] := 24.00; V[1] := 38.25; V[2] := 28.00; V[3] := 70.25; V[4] := 63.00 ; V[5] := 96.25;
```

b) *Par lecture* : On utilise pour cela l'opération de lecture : C'est la méthode la plus commode.

Exemple1 : Exemple d'algorithme / programme C permettant de lire et d'écrire (afficher) le tableau précédent :

```

algorithme LireEcrireTableau ;
variables
    v:tableau[0..5] de réel
    i:entier
Début
    écrire('Introduire V :');
    pour i←0 à 5 faire
        lire(v[i]);
    fin-Pour;
    écrire('Affichage de V :');
    pour i←0 à 5 faire
        écrire(v[i]);
    fin-Pour;
Fin;

#include <stdio.h>
int main()
{
    float v[6];
    int i;

    printf("Introduire V : \n");
    for (i=0; i<6; i++)
        scanf ("%f", &v[i]);
    printf ("Affichage V : \n") ;
    for (i=0; i<6; i++)
        printf ("%2f ",v[i]);
}

```

Link of program: <https://onlinegdb.com/yZireaVp2D>

À l'exécution : (RUN ou exécuter)

```

Introduire V :
12.50  12.75  56  60.25  36.75  65
Affichage V :
12.50  12.75  56  60.25  36.75  65

```

Exemple 2 : Lecture et écriture d'un tableau de N éléments.

algorithme LireEcrireTableau

variables

V:Tableau[0..99] de réel
N, i:entier

Début

écrire('Introduire nbre d'éléments N : ')
lire (N)
écrire('Introduire V : ')

pour $i \leftarrow 0$ **à** $N-1$ **faire**

lire (V[I]);

finPour;

écrire ('Affichage de V : ');

pour $i \leftarrow 0$ **à** $N-1$ **faire**

écrire (V[I]);

finPour;

Fin.

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
float V[100];
```

```
int N, i;
```

```
printf("Introduire nbre d'éléments N : \n");
```

```
scanf ("%d" , &N);
```

```
printf("Introduire V : \n");
```

```
for (i=0; i<N ; i++)
```

```
scanf ("%f", &V[i]);
```

```
printf("Affichage du V : \n") ;
```

```
for (i=0; i<N ; i++)
```

```
printf ("%f", V[i]);
```

```
}
```

Link of program: <https://onlinegdb.com/NZAEe06W->

À l'exécution : (RUN ou exécuter)

```
Introduire le nbre d'éléments N :
6
Introduire les composantes du Tableau V :
12.50 12.75 56 60.25 36.75 65
Affichage du V :
12.50 12.75 56 60.25 36.75 65
```

1.2.5. Manipulation de Tableaux unidimensionnels (Vecteurs)

Les tableaux ont une grande importance en informatique, la quasi-totalité des problèmes utilisent des structures de données sous forme de tableaux. On peut en citer le domaine du calcul matriciel, les statistiques, les traitement de gestion, *etc.* La gestion et l'analyse de données nécessitent l'organisation des données dans des tableaux pour rendre possible leur traitement informatique.

Exemples : Nous allons voir quelques exemples d'algorithmes pour les tableaux à une seule dimension (Vecteurs).

a) Somme de deux Vecteurs V1 et V2**Algorithme** Somme**Variables**V1, V2, V: **Tableau**[0..99] de réel;
N, i: entier;**Début**

Lire (N);

pour i←0 à N-1 **faire**

Lire (V1[i]);

FinPour;**pour** i←0 à N-1 **faire**

Lire (V2[i]);

FinPour;**pour** i←0 à N-1 **faire**

V[i] ← V1[i] + V2[i];

FinPour;**pour** i←0 à N-1 **faire**

Écrire (V[i]);

FinPour;**Fin.**

#include <stdio.h>

int main()

{

float V1[100], V2[100], V[100];
int N, i;printf("Introduire nbre d'éléments N : \n");
scanf ("%d" , &N);

printf("Introduire V1 : \n");

for (i=0; i<N ; i++)
scanf ("%f", &V1[i]);

printf("Introduire V2 : \n");

for (i=0; i<N ; i++)
scanf ("%f", &V2[i]);**for** (i=0; i<N ; i++)
V[i] = V1[i] + V2[i];

printf("Affichage du V : \n");

for (i=0; i<N ; i++)
printf ("%f", V[i]);

}

Link of program: <https://onlinegdb.com/LEI04pFDi>**b) Produit scalaire de deux vecteurs V1 et V2****Algorithme** ProduitScalaire;**Variables**V1, V2 : **Tableau** [0..99] de réel;
N, i: entier;
PS : réel;**Début**

Lire (N)

pour i←0 à N-1 **faire**

Lire (V1[i]);

Fin-Pour;**pour** i←0 à N-1 **faire**

Lire (V2[i]);

Fin-Pour;

PS ← 0;

pour i←0 à N-1 **faire**

PS ← PS + V1[i] * V2[i];

FinPour;

Écrire(PS);

Fin

#include <stdio.h>

int main()

{

float V1[100], V2[100], ps;
int N, i;printf("Introduire nbre d'éléments N : \n");
scanf ("%d" , &N);

printf("Introduire V1 : \n");

for (i=0; i<N ; i++)
scanf ("%f", &V1[i]);

printf("Introduire V2 : \n");

for (i=0; i<N ; i++)
scanf ("%f", &V2[i]);ps = 0;
for (i=0; i<N ; i++)
ps += V1[i]*V2[i];

printf ("Produit scalaire %.2f", ps);

}

Link of program: <https://onlinegdb.com/WhuJLL2OQ>

c) Recherche d'un élément dans un tableau

Chercher un élément val dans un tableau à une dimension (vecteur). S'il existe on récupère son rang.

```

Algorithme Recherche;
Variables
  V:Tableau[0..99] de reel;
  Val : reel;
  N, I, R:entier;
  Trouve : boolean;
Début
  Lire (N);
  pour i←0 à N-1 faire
    Lire (V1[i]);
  FinPour
  Lire (Val);

  i ← 1 ;
  Trouve ← Faux;
  TantQue (i<=N) et (Trouve=Faux) faire
    Si V[i] = Val Alors
      Trouve ← Vrai;
      R ← i;
    FinSi
    i ← i + 1;
  Fin-Tant-Que;

  Si Trouve = Vrai Alors
    Écrire('La valeur ',Val, ' exist. ');
    Écrire('Son Rang est : ', R);
  Sinon
    Écrire(Val, ' n'existe pas dans V. ');
  FinSi;
Fin;

```

```

#include <stdio.h>
#include <stdbool.h>

int main()
{
  float V[100], val;
  int n, i, pos;
  bool trouve;

  printf("Donner la taille de V : \n");
  scanf ("%d" , &n);
  printf("Introduire V1 : \n");
  for (i=0; i<n; i++)
    scanf ("%f", &V[i]);
  scanf("%f", &val);

  i=0; trouve=false;
  while ((i<n) && (!trouve)){
    if (V[i] == val){
      trouve=true; pos = i;
    }
    i++;
  }

  if (trouve)
    printf("La valeur %f existe à la position = %d", val, pos);
  else
    printf("La valeur %f n'existe pas dans V", val);
}

```

Link of program : <https://onlinegdb.com/T9RPBkDt1>

d) Recherche du maximum dans un tableau unidimensionnel

Soient deux tableaux NOMS et NOTES contenant respectivement des noms d'étudiants et leurs notes respectives, on veut sélectionner la meilleure note de l'étudiant correspondant.

```

Algorithme Maximum;
Variables
  Noms : Tableau[0..99] de chaîne;
  Notes : Tableau[0..99] de réel ;
  N, i, imax : entier;
  max : réel;
Début
  Lire (N);
  pour i←0 à N-1 faire
    Lire (Noms[i]);
    Lire (Notes[i]);
  Fin-Pour;
  max ← Notes[0]; imax ← 0;
  pour i←1 à N-1 faire
    si Notes[i] > max then
      max ← Notes[i];
      imax ← i;
    fin-si;
  Fin-Pour;

  Écrire('L'étudiant : ', Noms[imax]);
  Écrire(' a obtenu la meilleur note : ');

```

```

Écrire (max);
Fin.

#include <stdio.h>
int main()
{
  char Noms[100][100];
  float Notes[100];
  int n, i, imax;
  float max;

  scanf ("%d", &n);
  for (i=0; i<n; i++) {
    scanf ("%s", Noms[i]);
    scanf ("%f", &Notes[i]);
  }
  max=Notes[0]; imax=0;
  for (i=1; i<n; i++)
  {
    if (Notes[i] > max) {
      max = Notes[i]; imax=i;
    }
  }
}

```

Link of program: <https://onlinegdb.com/w4amw5Md4>

```
printf("L'étudiant : %s", Noms[imax]);    }
printf("a obtenu la meilleur note :%.2f", max)
```

e) L'algorithme de Tri par sélection (permutations)

– Méthode de Tri Simple (Tri croissant)

Algorithme Tri_Selection;

Variables

T:Tableau[0..99] de reel;
I, J, N:entier;
Z : real;

Début

Lire (N);

pour i←1 à N-1 **faire**

Lire (T[i]);

Fin-Pour;

pour i←1 à N-2 **faire**

pour j=(i+1) à N-1 **faire**

Si T[I] > T[J] **Alors**

Z ← T[I];

T[I] ← T[J];

T[J] ← Z;

Fin-Si;

Fin-Pour;

Fin-Pour;

pour i=1 à N **faire**

Ecrire (T[i]);

Fin-Pour;

Fin;

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
float T[100];
```

```
int i, j, n;
```

```
float Z:real;
```

```
scanf("%d", &n);
```

```
for (i=0; i<n; i++)
```

```
Read("%f", &T[i]);
```

```
for (i=0; i<(n-1); i++)
```

```
for (j=i+1; j<n; j++)
```

```
if (T[i] > T[j])
```

```
{
```

```
Z = T[i];
```

```
T[i] = T[j];
```

```
T[j] = z;
```

```
}
```

```
for (i=0; i<n; i++)
```

```
printf("%8.2f ", T[i]);
```

```
}
```

Link of program: <https://onlinegdb.com/e3H5IaefrO>

An execution of : <https://onlinegdb.com/w4amw5Md4>

Give the size of the students' list : 5

Give the names and grades of the students :

Student N° 1 :

Name : Omar

Grade : 15

Student N° 2 :

Name : Said Mohand

Grade : 13.25

Student N° 3 :

Name : Fatiha

Grade : 18.25

Student N° 4 :

Name : Mohand Said

Grade : 19.25

Student N° 5 :

Name : Samira

Grade : 11.25

The student : Mohand Said got the best grade : 19.25

An execution of : <https://onlinegdb.com/e3H5IaefrO>

Give the size of the vector T : 5

Give the values of the vector T :

65 8 7 -9 -18

The sorted vector T is :

-18.00 -9.00 7.00 8.00 65.00

1.3. Les tableaux bidimensionnels (Matrices)

1.3.1. Présentation de tableaux bidimensionnels

Un tableau est dit bidimensionnels (ou à deux dimensions), si chacun des ses éléments (composant) est repéré par un couple d'indices (i, j) où i est le numéro de la ligne et j est le numéro de la colonne où l'élément est situé. Ce type de tableau est appelé *Matrice*.

Soit la matrice A suivant :

$$A = \begin{matrix} & 14.32 & 14.50 & 14.25 & 25.84 \\ 15.26 & 15.65 & 12.23 & 15.36 \\ 12.25 & 16.23 & 10.20 & 15.63 \end{matrix}$$

peut être représentée par une tableau bidimensionnels (avec deux dimensions) de trois ligne et quatre colonnes :

$A =$	0	1	2	3
0	14.32	14.50	14.25	25.84
1	15.26	15.65	12.23	15.36
2	12.25	16.23	10.20	15.63

1.3.2. Déclaration de tableaux bidimensionnels

Pour déclarer la matrice A définie précédemment, on a deux façons :

1ère Façon :

ALGORITHME

Variables

A : Tableau[0..2,0..3] de Réel

LANGAGE C

float A[3][4];

Exemples

Une chaîne de 28 magasins, chacun comportant 4 rayons. On veut établir l'état des ventes hebdomadaires. Ces données sont ensuite entrées dans un ordinateurs en utilisant un tableau à deux dimensions où le premier indices repère le magasin et le second le rayon. Si VENTES est le nom du tableau, décrire cette représentation des données.

VENTES	00	01	02	03
00	2872	805	3211	1560
01	2196	1225	2525	1477
02	3257	1017	3687	1951
....
....
....
....
27	2618	913	2333	982

Variables

VENTES : **Tableau**[0..27,0..3] **de** Entier

I.3.2. Lecture et affichage d'une matrice

L'algorithme (respectivement, le programme) suivant permet de lire et d'écrire une matrice de n

ligne et m colonnes :

Algorithme LectureAffichageMatrice

Variables

mat:tableau [0..99, 0..99] de réel;
N,M, i,j:entier;

Début

Lire (N, M);

pour i←0 **à** N-1 **faire**

Pour j←0 **à** M-1 **faire**

 Lire (mat[i, j]);

FinPour;

FinPour;

pour i←0 **à** N-1 **faire**

Pour j←0 **à** M-1 **faire**

 Écrire (mat[i, j]);

FinPour;

FinPour;

Fin.

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    float mat[100][100];
```

```
    int i, j, n, m;
```

```
    scanf("%d %d", &n, &m);
```

```
    for (i=0; i<n; i++)
```

```
        for (j=0; j<m; j++)
```

```
            scanf("%f", &mat[i][j]);
```

```
    for (i=0; i<n; i++)
```

```
        for (j=0; j<m; j++)
```

```
            printf("%8.2f", mat[i][j]);
```

```
    }
```

```
}
```

On doit introduire le nombre de lignes n et le nombre de colonnes m . par la suite on introduit les éléments de la matrice $mat[i,j]$ / $i=1\dots n$ et $j=1..m$

I.3.3. Manipulation de matrice

a) Lire et écrire un tableau bidimensionnels

Algorithme LectureEcritureMatrice

Variables
mat: **tableau** [0..99, 0..99] **de** reel;
N, M, i, j: entier;

Début

```
Lire (N, M);
pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    Lire (mat[i, j]);
  FinPour;
FinPour;
```

```
pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    Écrire (mat[i, j]);
  FinPour;
FinPour;
```

Fin.

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
float mat[100][100];
```

```
int i, j, n, m;
```

```
printf("Donner n et m :");
```

```
scanf("%d %d", &n, &m);
```

```
for (i=0; i<n; i++)
```

```
for (j=0; j<m; j++)
```

```
scanf("%f", &mat[i][j]);
```

```
printf("Affichage de la matrice mat : \n");
```

```
for (i=0; i<n; i++) {
```

```
for (j=0; j<m; j++)
```

```
printf("%8.2f", mat[i][j]);
```

```
printf("\n");//Saut de ligne
```

```
}
```

```
}
```

Link of program: <https://onlinegdb.com/aXAoq9G3r>

b) Produit d'une matrice par un vecteur

(le nombre de colonne de la matrice = le nombre de composantes du vecteur)

Algorithme ProduitMatVect;

Variables
mat : **tableau** [0..49, 0..49] **de** reel;
vect : **tableau** [0..49] **de** reel;
p : **tableau** [0..49] **de** reel;
N, M, i, j: entier;

Début

```
Lire (N, M);
pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    Lire (mat[i, j]);
  FinPour;
FinPour;
pour i←0 à M-1 faire
  Lire(vect[i]);
finPour
pour i←0 à N-1 faire
  p[i] = 0;
  Pour j←0 à M-1 faire
    P[i] ← p[i] + mat[i, j]*vect[j];
  FinPour;
FinPour;
```

```
pour i←0 à N-1 faire
  Écrire(p[i]);
finPour;
```

Fin.

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
float mat[10][10], vect[10], p[10];
```

```
int i, j, n, m;
```

```
printf("Introduire la taille de MAT : ");
```

```
scanf("%d %d", &n, &m);
```

```
printf("Introduire les valeurs de MAT : \n");
```

```
for (i=0; i<n; i++)
```

```
for (j=0; j<m; j++)
```

```
scanf("%f", &mat[i][j]);
```

```
printf("Introduire les valeurs de VECT : \n");
```

```
for (j=0; j<m; j++)
```

```
scanf("%f", &vect[j]);
```

```
for (i=0; i<n; i++) {
```

```
p[i] = 0;
```

```
for (j=0; j<m; j++)
```

```
p[i] += mat[i][j] * vect[j];
```

```
}
```

```
printf("Produit P = MatxVect : \n");
```

```
for (i=0; i<n; i++)
```

```
printf("%8.2f", p[i]);
```

```
}
```

Link of program: <https://onlinegdb.com/ZQEc4bPVp>

c) Compter le nombre d'éléments négatifs, positifs et nuls dans une matrice**Algorithme** compteurPNN

```

Variables
mat : tableau [1..10, 1..10] de reel;
N, M, i, j: entier
nbP, nbN, nbNul : entier;
Début
Lire (N, M);
pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    Lire (mat[i, j]);
  FinPour;
FinPour;
nbP ← 0; nbN ← 0; nbNul ← 0;
pour i←0 à N-1 faire
  Pour j←0 à M-1 faire
    Si mat[i, j] > 0 Alors
      nbP ← nbP + 1;
    Sinon
      Si mat[i, j] < 0 Alors
        nbN ← nbN + 1;
      Sinon
        nbNul ← nbNul + 1;
      FinSi;
    FinSi;
  FinPour;
FinPour;
Écrire (nbP, nbN, nbNul);
Fin.

```

```

#include <stdio.h>
int main()
{
  float mat[10][10];
  int i, j, n, m;
  int nbP, nbN, nbNul;

  printf("Introduire la taille de MAT : ");
  scanf("%d %d", &n, &m);
  printf("Introduire les valeurs de MAT : \n");
  for (i=0; i<n; i++)
    for (j=0; j<m; j++)
      scanf("%f", &mat[i][j]);

  nbP = 0; nbN=0; nbNul=0;

  for (i=0; i<n; i++)
    for (j=0; j<m; j++)
      if (mat[i][j] > 0)
        nbP++;
      else if (mat[i][j] < 0)
        nbN++;
      else
        nbNul++;

  printf("Nb valeurs positives %d : \n", nbP);
  printf("Nb valeurs négatives %d : \n", nbN);
  printf("Nb valeurs nulles %d : \n", nbNul);
}

```

Link of program: <https://onlinegdb.com/qDHXKbVOI>

nbP est le compteur des nombres positifs, **nbN** est le compteur des nombre négatifs et **nbNul** est le compteur des nombre nuls.

d) Produit de deux matrices

Le nombre de colonne de la première matrice = le nombre de ligne de la deuxième matrice

Algorithme compteurPNN

Variables

$m1, m2$: **tableau** [0..49, 0..49] **de** reel {

N, M, L, i, j, k :entier

$Result$:**tableau** [0..49, 0..49] **de** reel

Début

Lire (N, M, L);

pour $i \leftarrow 0$ **à** $N-1$ **faire**

Pour $j \leftarrow 0$ **à** $M-1$ **faire**

 Lire ($m1[i, j]$);

FinPour;

FinPour;

pour $i \leftarrow 0$ **à** $M-1$ **faire**

Pour $j \leftarrow 0$ **à** $L-1$ **faire**

 Lire ($m2[i, j]$);

FinPour;

FinPour;

pour $i \leftarrow 0$ **à** $N-1$ **faire**

Pour $j \leftarrow 0$ **à** $L-1$ **faire**

$Result[i, j] \leftarrow 0$;

pour $k \leftarrow 0$ **à** $M-1$ **faire**

$Result[i, j] \leftarrow Result[i, j] +$
 $m1[i, k] * m2[k, j]$;

finPour;

FinPour;

FinPour;

pour $i \leftarrow 0$ **à** $N-1$ **faire**

Pour $j \leftarrow 0$ **à** $L-1$ **faire**

 Écrire ($Result[i, j]$);

FinPour;

FinPour;

Fin.

```
#include <stdio.h>
```

```
int main()
```

```
{
    float m1[10][10], m2[10][10];
```

```
    int N, M, L, i, j, k;
```

```
    float Result[10][10];
```

```
    printf("Donnez les dim. de m1 : ");
```

```
    scanf ("%d %d", &N, &M);
```

```
    printf('Donnez le nbre de colonnes de m2 : ');
```

```
    scanf ("%d", &L);
```

```
    for (i=0; i<N ; i++)
```

```
        for (j=0; j<M ; j++)
```

```
            scanf("%f", &m1[i][j]);
```

```
    for (i=0; i<M ; i++)
```

```
        for (j=0; j<L ; j++)
```

```
            scanf("%f", &m2[i][j]);
```

```
    for (i=0; i<M ; i++)
```

```
        for (j=0; j<L ; j++)
```

```
    {
```

```
        Result[i, j] = 0;
```

```
        for (k=0; k<M ; k++)
```

```
            Result[i, j] += m1[i, k]*m2[k, j];
```

```
    }
```

```
    printf("La matrice Resultat=M1xM2 est :\n");
```

```
    for (i=0; i<M ; i++)
```

```
    {
```

```
        for (j=0 ; j<L ; j++)
```

```
            printf("%8.2f", Result[i, j]);
```

```
            printf("\n"); //Saut de ligne
```

```
    }
```

```
}
```

Link of program: <https://onlinegdb.com/wUOoj2G5b>

Pour réaliser le produit des deux matrices $m1$ et $m2$, il faut que le nombre de colonne de $m1$ soit égale au nombre de ligne de $m2$. Le résultat est une matrice $Result$ dont le nombre de ligne est le même que celui de $m1$ et le nombre de colonne est le même que celui de $m2$.

$$Result(N, L) = m1(N, M) * m2(M, L)$$

x $Result$ est une matrice de N lignes et L colonnes

x $m1$ est une matrice de N lignes et M colonnes

x $m2$ est une matrice de M lignes et L colonnes

x L'élément $Result[i, j]$ est calculé comme suit :

$$Result[i, j] = m1[i, 1] * m2[1, j] + m1[i, 2] * m2[2, j] + m1[i, 3] * m2[3, j] + \dots +$$

$$m1[i, M] * m2[M, j] = \sum_{k=1}^M m1[i, k] * m2[k, j]$$