Plan du Chapitre 3

Chapitre 3 : Variables Indicées – Le type Tableau

3.1. Introduction

Jusque là, on a 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 de données. 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 c2 c3 c4 c5]

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

1	2	3	4	5
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[1] est la première composante du tableau *VECT*

VECT[2] est la deuxième composante du tableau *VECT*

VECT[3] est la troisième composante du tableau *VECT*

Etc.

3.2. 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).

3.2.1. 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 :

1	*2*	*3*	*4*
12.50	*25.75*	*56.00*	*60.25*

V2 :

1	*2*	*3*	*4*	*5*	*6*
24.00	*38.25*	*28.00*	*70.25*	*63.00*	*96.25*

On a :

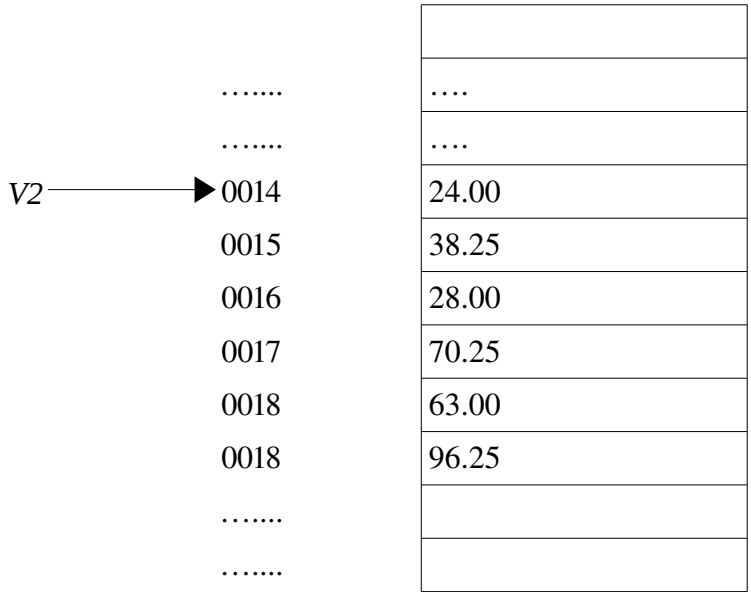
V1[1] = 12.55 ; *V1[2] = 25.75* ; *V1[3] = 56.00* ; *V1[4] = 60.25*

V2[1] = 24.00; *V2[2] = 38.25*; *V2[3] = 28.00*; *V2[4] = 70.25*; *V2[5] = 63.00* ; *V2[6] = 96.25* ;

3.2.2. 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 :

	Adresses	***Mémoire***
	0000	
	0001	
	0002	
	…….	
	…….	
	…….	
V1 →	0006	12.50
	0007	25.75
	0008	56.00
	0009	60.25



3.2.3. 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[1..MAX] de <type>;
```

Syntaxe en PASCAL

```
const
   MAX = <valeur_entière>;
var
   <id_tab>:Array [1..MAX] of <type>;
```

Remarques :

- ✗ C'est pas obligatoire d'utiliser une constante pour la taille maximale de tableau ; cependant, c'est une bonne pratique dans la programmation
- ✗ La plage d'indice *1* .. *MAX* représente l'indice du premier élément *1* et l'indice du dernier élément *MAX*.
- ✗ 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
   MAX = 50
variable
   V:Tableau[1..MAX] de réel;
```

```
const
   MAX = 50;
var
   V:Array [1..MAX] of real;
```

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

3.2.4. 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[1] := 24.00; *V[2] := 38.25*; *V[3] := 28.00*; *V[4] := 70.25*; *V[5] := 63.00* ; *V[6] := 96.25*;

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

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

```
algorithme LireEcrireTableau
variables
   v:tableau[1..6] de réel
   i:entier
Début
      écrire('Introduire V :')
      pour i:=1 à 6 faire
            lire(v[i])
      finPour
      écrire('Affichage V :')
      pour i:=1 à 6 faire
            écrire(v[i])
      finPour
Fin
```

```
program LireEcrireTableau;
      uses wincrt;
      var
            V:Array[1..6] of real;
            i:integer;
Begin
      writeln('Introduire V : ');
      for i:=1 to 6 do
            Read (V[I]);
      writeln ('Affichage V : ') ;
      for i:=1 to 6 do
            Write (V[I]);
End.
```

À l'exécution : (RUN ou exécuter)

```
Introduire les composantes du Tableau V :
12.50  12.75  56  60.25  36.75  65
Affichage des composantes du Tableau 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[1..100] de réel
            N, i:entier

Début
  écrire('Introduire nbre d'éléments N : ')
  lire (N)
  écrire('Introduire V : ')
  pour i:=1 à N faire
     lire (V[I])
  finPour
  écrire ('Affichage de V : ') ;
  pour i:=1 à N faire
    écrire (V[I])
  finPour
  Fin
```

```
program LireEcrireTableau;
      uses wincrt;
      var
            V:Array[1..100] of real;
            N, i:integer;
Begin
   write('Introduire nbre d'éléments N:');
   read (N);

   writeln('Introduire V : ');
   for i:=1 to N do
      Read (V[I]);
   writeln ('Affichage du V : ') ;
   for i:=1 to N do
      Write (V[I]);
End.
```

À l'exécution : (RUN ou exécuter)

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

3.2.5. 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[1..100] de réel;
   N, i:entier;
Début
   Lire (N)
   pour i=1 à N faire
      Lire (V1[i])
   FinPour

   pour i=1 à N faire
      Lire (V2[i])
   FinPour

   pour i=1 à N faire
      V[i] ← V1[i] + V2[i]
   FinPour

   pour i=1 à N faire
      Écrire (V[i])
   FinPour
Fin
```

```
Program Somme;
Uses Wincrt;
Var
   V1, V2, V : Array[1..100] of real;
   N, i : integer;
Begin
   Read (N);
   for i=1 to N do
     readln (V1[i]);

   for i=1 to N do
     read (V2[i]);


   for i=1 to N do
     V[i]:=V1[i]+V2[i] ;

   for i=1 to N do
     Write(V[i]) ;
End.
```

b) Produit scalaire de deux vecteurs V1 et V2

```
Algorithme ProduitScalaire

Variables
   V1, V2 : Tableau [1..100] de réel
   N, i:entier
   PS : réel
Début
   Lire (N)
   pour i=1 à N faire
     Lire (V1[i])
   FinPour
   pour i=1 à N faire
     Lire (V2[i])
   FinPour
   PS  ← 0
   pour i=1 à N faire
     PS ← PS + V1[i] * V2[i]
   FinPour

   Écrire (PS)
Fin
```

```
Program ProduitScalaire;
Uses Wincrt;
Var
   V1, V2 : Array[1..100] of real;
   N, i : integer;
   PS : real;
Begin
   Read (N);
   for i=1 to N do
     readln (V1[i]);

   for i=1 to N do
     readln (V2[i]);

   PS:=0;
   for i=1 to N do
     PS:= PS + V1[i]*V2[i] ;



   Write('Le produit scalaire = ', PS);
End.
```

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[1..100] de reel;
   Val : reel;
   N, I, R:entier;
   Trouve : boolean;
Début
   Lire (N)
   pour i=1 à N 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
   FinTantQue

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

```
Program Recherche;
Uses Wincrt;
Var
   V : Array[1..100] of real;
   Val : real;
   N, I, R : integer;
   Trouve : boolean ;
Begin
   Read (N);
   for i=1 to N do
     readln (V[i]);
   read(val);

   i:=1 ;
   while (i<=N) and (Trouve=false) do
    begin
        if V[i] = Val then
         begin
            Trouve := true;
            R:=i;
         end ;
         i:=i+1;
    end;

   if Trouve = true then
    begin
       write('La valeur ', val, ' existe dans V');
       write('Son rang est : ', R);
    end
   else
       write(Val,' n''existe pas dans V.');
End.
```

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 meilleurs note de l'étudiant correspondant.

```
Algorithme Maximum
 Variables
   Noms : Tableau[1..100] de chaîne
   Notes : Tableau[1..100] de réel
   N, i, imax : entier
   max : réel
Début
   Lire (N)
   pour i=1 à N faire
     Lire (Noms[i])
     Lire (Notes[i])
   FinPour

   max ← Notes[1]   imax ← 1;

   pour i←2 à N faire
     si Notes[i] > max then
        max ← Notes[i]
        imax ← i
     finsi
   FinPour

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

```
Program Recherche;
Uses Wincrt;
Var
   Noms : Array[1..100] of String;
   Notes : Array[1..100] of real;
   n, i, imax : integer;
   max : real;
Begin
   Read (n);
   for i=1 to N do
    begin
      readln (Noms[i]);
      readln (Notes[i]);
    end;

   max:=Notes[1] ;  imax:=1;
   for i:=2 to n do
    begin
        if Notes[i] > max then
          begin
              max := Notes[i];
              imax := i;
          end;
    end;

   write('L''étudiant : ', Noms[imax]);
   write('a obtenu la meilleur note : ');
   write(max)
End.
```

e) L'algorithme de Tri

- Méthode de Tri Simple (Tri croissant)

```
Algorithme TriSimple
 Variables
   T:Tableau[1..100] de reel
   I, J, N:entier
   Z : real
Début
   Lire (N)
   pour i=1 à N faire
      Lire (T[i])
   FinPour

   pour i=1 à N-1 faire
      pour j=(i+1) à N faire
           Si T[I] > T[J] Alors
                Z ← T[I]
                T[I] ← T[J]
                T[J] ← Z
           FinSi
      FinPour
   FinPour

   pour i=1 à N faire
      Ecrire (T[I])
   FinPour
Fin
```

```
Program TriSimple;
  uses wincrt;
  var
      T : Array[1..100] of real;
      I, J, N : integer;
      Z:real;
  Begin
      Readln(N);
      for i:=1 to n do
            Read(T[i]);

      for i:=1 to (N-1) do
        for j:=(i+1) to N do
            if T[I] > T[J] then
            begin
                  Z:=T[I];
                  T[I]:=T[J];
                  T[J]:=Z;
            end;

      for i:=1 to N do
            Write(T[I], '  ');
  End.
```

Réaliser le déroulement (Trace d'exécution) Pour N = 4) : Le tableau avec et après l'opération de tri.)

3.3. Les tableaux bidimensionnels (Matrices)

3.3.1. 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 =	***1***	***2***	***3***	***4***
1	14.32	14.50	14.25	25.84
2	15.26	15.65	12.23	15.36
3	12.25	16.23	10.20	15.63

3.3.2. Déclaration de tableaux bidimensionnels

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

1ère Façon :

ALGORITHME

```
Variables
      A:Tableau[1..3,1..4] de Réel
```

PASCAL

```
Var
      A:Array[1..3,1..4] of real;
```

2ème Façon :

ALGORITHME

```
Type Ligne = Tableau[1..4] de reel
Variables
      A:Tableau[1..3] de Ligne
```

PASCAL

```
Type Ligne = Array[1..4] of real;
Var
      A:Array[1..3] of Ligne;
```

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	***01***	***02***	***03***	***04***
01	2872	805	3211	1560
02	2196	1225	2525	1477
03	3257	1017	3687	1951
….	….	….	….	….
….	….	….	….	….
….	….	….	….	….
….	….	….	….	….
28	2618	913	2333	982

```
Variables
      VENTES : Tableau[1..28,1..4] de Entier
```

3.3.2. 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 [1..10, 1..10] de réel
   N,M, i,j:entier
Début
   Lire (N, M);
   pour i=1 à N faire
      Pour j=1 à M faire
        Lire (mat[i, j])
      FinPour
   FinPour

   pour i=1 à N faire
      Pour j=1 à M faire
        Écrire (mat[i, j])
      FinPour
   FinPour
Fin
```

```
Program LectureAffichageMatrice;
  uses wincrt;
  var
    mat:array [1..10, 1..10] of real;
    N,M, i,j : integer;
Begin
      Read(N, M);
      For i:=1 To N Do
            For j:=1 To M Do
                  Read(mat[i, j]);

      For i:=1 To N Do
      begin
            For j:=1 To M Do
            begin
                  Write(mat[i,j],'   ');
            end;
            writeln;
      end;
End.
```

On doit introduire le nombre de lignes *n* et le nombre de colonnes *m*.

3.3.3. Manipulation de matrice

a) Lire et écrire un tableau bidimensionnels

```
Algorithme LectureEcritureMatrice
 Variables
   mat:tableau [1..10, 1..10] de reel
   N,M, i,j:entier
Début
   Lire (N, M);
   pour i=1 à N faire
      Pour j=1 à M faire
        Lire (mat[i, j])
      FinPour
   FinPour

   pour i=1 à N faire
      Pour j=1 à M faire
        Écrire (mat[i, j])
      FinPour
   FinPour
Fin
```

```
Program LectureEcritureMatrice;
  uses wincrt;
  var
    mat:array [1..10, 1..10] of real;
    N,M, i,j : integer;
Begin
      Read(N, M);
      For i:=1 To N Do
            For j:=1 To M Do
                  Read(mat[i, j]);

      For i:=1 To N Do
      begin
            For j:=1 To M Do
            begin
                  Write(mat[i,j],'   ');
            end;
            writeln; {saut de ligne}
      end;
End.
```

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 [1..10, 1..10] de reel
   vect : tableau [1..10] de reel
   p : tableau [1..10] de reel
   N,M, i,j:entier
Début
   Lire (N, M);
   pour i=1 à N faire
      Pour j=1 à M faire
        Lire (mat[i, j])
      FinPour
   FinPour
   pour i=1 à M faire
      Lire(vect[i])
   finPour
   pour i=1 à N faire
      p[i] = 0
      Pour j=1 à M faire
        P[i] ← p[i] + mat[i, j]*vect[j]
      FinPour
   FinPour
   pour i=1 à N faire
      Écrire(p[i])
   finPour
Fin
```

```
Program ProduitMatVect;
  uses wincrt;
  var
    mat:array [1..10, 1..10] of real;
    vect, p:array[1..10] of real;
    N,M, i,j : integer;
Begin
      Read(N, M);
      For i:=1 To N Do
            For j:=1 To M Do
                  Read(mat[i, j]);
      For i:=1 To M Do
            Read(vect[i]);

      For i:=1 To N Do
      begin
        p[i]=0;
        For j:=1 To M Do
          p[i]:= p[i]+ mat[i,j]*vect[j];
      end;

      For i:=1 To N Do
        write(p[i]);
End.
```

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=1 à N faire
      Pour j=1 à M faire
        Lire (mat[i, j])
      FinPour
   FinPour

   nbP ← 0; nbN ← 0; nbNul ← 0;
   pour i=1 à N faire
      Pour j=1 à M 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
```

```
Program compteurPNN;
  uses wincrt;
  var
    mat:array [1..10, 1..10] of real;
    N,M, i,j : integer;
    nbP, nbN, nbNul, integer;
Begin
      Read(N, M);
      For i:=1 To N Do
            For j:=1 To M Do
                  Read(mat[i, j]);
      nbP:=0; nbN:=0;
      nbNul:=0;

      For i:=1 To N Do
            For j:=1 To M Do
              if mat[i,j]>0 then
                nbP:=nbP+1
              else
                if mat[i,j]<0 then
                   nbN:=nbN+1
                else
                   nbNul:=nbNul+1;

      Write(nbP, nbN, nbNul);
End.
```

***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 [1..10, 1..10] de reel
  N,M, L, i,j, k:entier
  Result:tableau [1..10, 1..10] de reel
Début
   Lire (N, M, L);
   pour i=1 à N faire
      Pour j=1 à M faire
         Lire (m1[i, j])
      FinPour
   FinPour

   pour i=1 à M faire
      Pour j=1 à L faire
         Lire (m2[i, j])
      FinPour
   FinPour

   pour i=1 à N faire
      Pour j=1 à L faire
         Result[i, j] ← 0
         pour k=1 à M faire
            Result[i,j]← Result[i,j]+
                         m1[i,k]*m2[k,j]
         finPour
      FinPour
   FinPour

   pour i=1 à N faire
      Pour j=1 à L faire
         Écrire (Result[i, j])
      FinPour
   FinPour
Fin
```

```
Program produitMatriceMatrice;
  uses wincrt;
  var
    m1,m2:array [1..10, 1..10] of real;
    N,M, L, i,j, k : integer;
    Result:array [1..10, 1..10] of real;
Begin
  Write('Donnez les dim. de m1 : ');
  read (N, M);
  write('Donnez le nbre de colonnes de m2 : ');
  read(L);

  For i:=1 To N Do
    For j:=1 To M Do
      Read(m1[i, j]);

  For i:=1 To M Do
    For j:=1 To L Do
      Read(m2[i, j]);

  For i:=1 To N Do
    For j:=1 To L Do
    begin
      Result[i, j]:=0;
        For k:=1 To M Do
          Result[i,j] := Result[i,j]
                         +m1[i,k]*m2[k,j];
    end;

  For i:=1 To N Do
  begin
    For j:=1 To L Do
      write(Result[i,j], '  ');
    writeln; {Saut de ligne}
  end;
End.
```

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 qui celui de *m1* et le nombre de colonne est le même qui celui de *m2*.

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

- ✗ *Result* est une matrice de *N* lignes et *L* colonnes
- ✗ *m1* est une matrice de *N* lignes et *M* colonnes
- ✗ *m2* est une matrice de *N* lignes et *L* colonnes
- ✗ L'élément *Result[i, j]* est calculé comme suit :

$$Result[i, j] = m1[i, 1] * m2[1, j] + m1[i, 2] * m2[2, j] + m1[i, 3] * m2[3, j] + \ldots\ldots\ldots\ldots + m1[i, M] * m2[M, j] = \sum_{k=1}^{M} m1[i, k] * m2[k, j]$$