Chapitre 4 : Les tableaux

Comment un algorithme peut-il mémoriser plusieurs valeurs d'un même type sans devoir déclarer une variable pour chacune d'elles ? Est-il possible de traiter une série de notes, de températures ou de valeurs saisies par l'utilisateur sans écrire plusieurs lignes répétitives ? Comment accéder facilement à la dixième ou la centième donnée saisie ?

Ce chapitre répondra à ces questions.

4.1 Introduction

Jusqu'à présent, nous avons manipulé des variables simples, chacune représentant une seule valeur. Cependant, dans de nombreux problèmes, il est nécessaire de gérer un ensemble de valeurs du même type en même temps : par exemple, les notes d'un étudiant, les températures d'une semaine, les salaires des employés d'un service, etc.

Restons sur l'exemple des notes : Pour un étudiant, nous avons besoin de dix notes afin de calculer sa moyenne générale.

À ce stade, deux solutions s'offrent à nous :

1. Utiliser une seule variable (par exemple note) et la réutiliser à chaque saisie : dans ce cas, chaque nouvelle lecture écrase la précédente, et il devient impossible de retrouver ou de réutiliser les anciennes valeurs dans le calcul.

2. Déclarer dix variables différentes : N1, N2, N3, …, N10 : cette méthode permet de conserver toutes les valeurs, mais elle rend l'algorithme long, répétitif et difficile à maintenir, car il faudrait dix lectures, dix additions, et des traitements similaires pour chaque note.

Ces deux approches montrent bien les limites des variables simples. C'est pourquoi l'algorithmique met à notre disposition un nouvel outil qui permet de regrouper plusieurs valeurs de même type sous un seul nom, tout en accédant individuellement à chacune grâce à un indice. Ce type de structure de données est appelé un tableau.

4.2 Structures de données

Une structure de données est une organisation logique et physique des données qui permet de regrouper plusieurs valeurs et de les manipuler efficacement au sein d'un programme.

Autrement dit, une structure de données permet :

- de stocker plusieurs informations sous un même nom,
- d'y accéder facilement,
- et de les traiter collectivement ou individuellement.

Il existe plusieurs types de structures de données (tableaux, listes, piles, files, arbres, etc.).

4.3 Définition du tableau

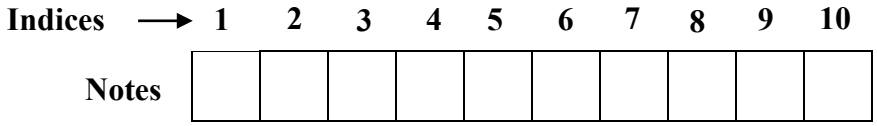
Un tableau est une structure de données homogène qui permet de stocker plusieurs valeurs du même type sous un nom unique. Il existe deux types de tableaux : les tableaux a une dimension (Vecteurs) et les tableaux a deux dimensions (Matrices).

4.4 Tableau à une dimension (Vecteur)

Un tableau à une dimension, ou vecteur, est un ensemble d'éléments de même type, repérés par un seul indice. Cet indice permet d'indiquer sa place dans le tableau, il permet d'accéder directement à la valeur située dans une position donnée.

Exemple :

Si l'on souhaite stocker les notes de dix matières d'un étudiant, on peut créer un vecteur nommé *Notes* comportant 10 cases :



Syntaxe (Déclaration)

Algorithmique :

Variables id_tab : **Tableau [1..**Taille_max**] de** type ;

En langage C :

```
Type id_tab[Taille_max];
```

Exemple :

Variables Notes : Tableau **[**1**..**10**]** de réel ;

En langage C :

```
float Notes[10];
```

4.5 Manipulation d'un vecteur

Une fois le vecteur déclaré, il est possible de manipuler ses éléments individuellement ou collectivement à l'aide de leur indice.

a. Accès à un élément

L'accès à un élément d'un vecteur se fait en utilisant son indice à travers la syntaxe suivante : id_tab[indice] , c'est la même syntaxe que nous utilisons en langage C.

b. Lecture d'un élément/Lecture d'un vecteur

La lecture d'un élément d'un vecteur consiste à lire une composante du vecteur en utilisant son indice. La lecture d'un vecteur, quant à elle, consiste à lire l'ensemble de ses éléments et ce, à l'aide d'une boucle. Comme le nombre d'éléments du vecteur est connu, la boucle *pour* est la plus adaptée pour effectuer cette opération.

c. Écriture d'un vecteur

L'écriture d'un vecteur permet d'afficher l'ensemble de ses éléments. Elle s'effectue également à l'aide d'une boucle.

d. Affectation d'un élément

L'affectation permet de modifier la valeur d'un élément du vecteur en utilisant son indice.

Syntaxe id_tab[indice] ← valeur ; ou id_tab[indice1] ← id_tab[indice2] ou
id_tab[indice]← Expression ; id_tab[indice]← Constante ;

Exemples : Soit T et V deux vecteurs d'entiers et a et b des entiers
T[1] ← 40 ;
T[2] ← T[3] ;
T[10] ← V[1] ;
T[5] ← a + b*3 ;

Algorithme de lecture et d d'écriture d'un vecteur :

```
Algorithme LE_Vect ;
Variables     T : Tableau [1..100] d'entier;
              n, i : entier ;
Début
Répéter
   Écrire("Donner le nombre d'éléments du vecteur:") ;
   Lire(n) ;
Jusqu'à (n>0 et n <= 100) ;

Pour i ← 1 à n faire
   Lire(T[i]);
Fin-pour;

Écrire(" Voici votre vecteur : T = [ ") ;
```

Pour i ← 1 **à** n **faire**
  Ecrire(T[i]);
Fin-pour;
Écrire("] ") ;

Fin.

En langage C :

```
#include <stdio.h>
int main() {
    int T[100], n, i ;

  do {
     printf("Donner le nombre de composantes du vecteur:");
     scanf("%d", &n);
  } while (n<= 0 || n >100) ;

  for (i = 0; i < n; i++) {
    scanf("%d", &T[i]);
   }
  printf("Voici votre vecteur : T = [ ");

  for (i = 0; i < n; i++) {
    printf("%d ", T[i]);
   }
  printf(" ] \n");

    return 0;}
```

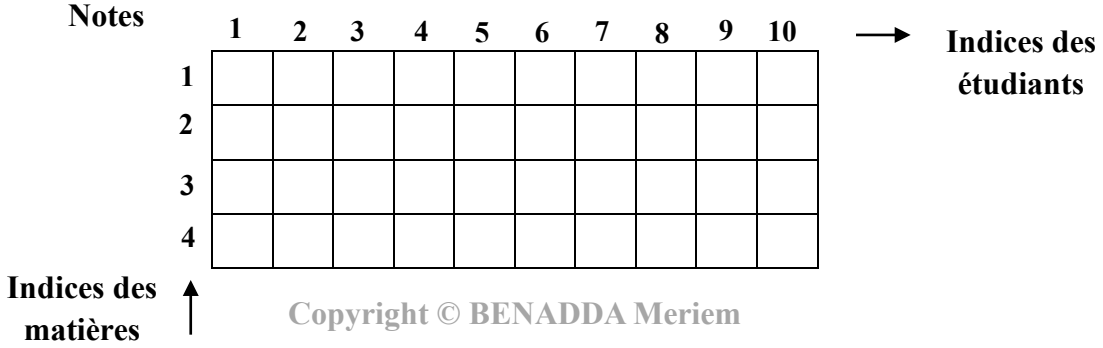
4.6 Tableau à deux dimensions (Matrice)

Un tableau à deux dimensions, ou matrice, est un ensemble d'éléments de même type, repérés par deux indices.

Ces indices permettent d'indiquer la position de chaque élément dans le tableau : le premier indice correspond généralement à la ligne et le second à la colonne. Ils permettent ainsi d'accéder directement à la valeur située à l'intersection d'une ligne et d'une colonne données.

Exemple :

Si l'on souhaite stocker les notes de dix étudiants dans cinq matières, on peut créer une matrice nommée *Notes* comportant 10 lignes (étudiants) et 4 colonnes (matières).



Syntaxe (Déclaration)

Algorithmique :

Variables id_tab : **Tableau [1..**maxLigne, **1..** maxCol**] de** type ;

En langage C :

```
Type id_tab[maxLigne][maxCol];
```

Exemple :

Variables Notes : Tableau **[**1**..**20, 1.. 10**]** de réel ;

En langage C :

```
float Notes[20][10];
```

4.7 Manipulation d'une matrice

Une fois la matrice déclarée, il est possible de manipuler ses éléments individuellement ou collectivement à l'aide de leurs indices.

a. Accès à un élément

L'accès à un élément d'une matrice se fait en utilisant **deux indices** : le premier pour la ligne et le second pour la colonne, selon la syntaxe suivante :

id_tab[indL, indC].

b. Lecture d'un élément/ Lecture de la matrice

La lecture d'un élément d'une matrice consiste à lire une composante située à l'intersection d'une ligne et d'une colonne, en utilisant ses deux indices.

La lecture d'une matrice, quant à elle, consiste à lire l'ensemble de ses éléments à l'aide de **deux boucles imbriquées**.

c. Écriture d'un vecteur

L'écriture d'une matrice permet d'afficher l'ensemble de ses éléments.
Elle s'effectue également à l'aide de deux boucles imbriquées, afin de parcourir successivement les lignes et les colonnes de la matrice.

d. Affectation d'un élément

L'affectation permet de modifier la valeur d'un élément de la matrice en utilisant ses deux indices.

Syntaxe id_tab[indL, indC] ← valeur ; ou id_tab[indL, indC] ← id_tab2[indL, indC] ou id_tab[indL, indC] ← Expression ; id_tab[indL, indC] ← Constante ;

Exemples : Soit A et B deux matrices d'entiers et a et b des entiers
A[1, 2] ← 31 ;
A[2, 7] ← B[3,7] ;
A[1,10] ← A[1,4] ;
A[4,4] ← a + b*3 ;

Algorithme de lecture et d'écriture d'une matrice :

Algorithme LE_Mat ;
Variables A : Tableau **[**1**..**100,1..100**]** d'entier;
n, m, i, j : entier ;
Début
Répéter
Écrire("Donner le nombre de lignes:") ;
Lire(n) ;
Jusqu'à (n>0 et n <= 100) ;

Répéter
Écrire("Donner le nombre de colonnes:") ;
Lire(m) ;
Jusqu'à (m>0 et m <= 100) ;

Pour i ← 1 **à** n **faire**
Pour j ← 1 **à** m **faire**
Lire(A[i, j]);
Fin-pour;
Fin-pour;

Écrire(" Voici votre matrice : ") ;
Pour i ← 1 **à** n **faire**
Pour j ← 1 **à** m **faire**
Ecrire(A[i, j]);
Fin-pour;
Fin-pour;

Fin.

En langage C :

```
#include <stdio.h>

int main() {
    int A[100][100];
    int n, m, i, j;


    do {
        printf("Donner le nombre de lignes : ");
        scanf("%d", &n);
    } while (n <= 0 || n > 100);


    do {
        printf("Donner le nombre de colonnes : ");
        scanf("%d", &m);
    } while (m <= 0 || m > 100);
```

```
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%d", &A[i][j]);
        }
    }


    printf("Voici votre matrice :\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            printf("%d ", A[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

4.8 Exemples usuels

4.8.1 Addition de deux vecteurs

Soient T1 et T2 deux vecteurs de même taille n.
L'addition de deux vecteurs consiste à additionner les éléments de même indice pour obtenir un troisième vecteur T.

Algorithme AdditionVecteurs ;
Variables
  T1, T2, T : Tableau [1..100] d'entier;
  n, i : entier ;
Début
  Répéter
    Écrire("Donner la taille des vecteurs :") ;
    Lire(n) ;
  Jusqu'à (n > 0 et n <= 100) ;

  Pour i ← 1 à n faire
    Lire(T1[i]) ;
  Fin-Pour ;

  Pour i ← 1 à n faire
    Lire(T2[i]) ;
  Fin-Pour ;

  Pour i ← 1 à n faire
    T[i] ← T1[i] + T2[i] ;
  Fin-Pour ;
  Pour i ← 1 à n faire
    Écrire(T[i]) ;

En langage C :

```
#include <stdio.h>

int main() {
  int T1[100], T2[100], T[100];
  int n, i;

  do {
    printf("Donner la taille des vecteurs :
");
      scanf("%d", &n);
  } while (n <= 0 || n > 100);

  for (i = 0; i < n; i++) {
      scanf("%d", &T1[i]);
  }

  for (i = 0; i < n; i++) {
      scanf("%d", &T2[i]);
  }

  for (i = 0; i < n; i++) {
      T[i] = T1[i] + T2[i];
  }

  for (i = 0; i < n; i++) {
      printf("%d ", T[i]);
  }
  printf("\n");

  return 0;}
```

```
   Fin-Pour ;
Fin.
```

4.8.2 Soustraction de deux matrices

Soient A1 et A2 deux matrices de même dimension n × m.
La soustraction de deux matrices consiste à soustraire les éléments de même position pour obtenir une matrice A.

```
Algorithme SoustMatrices ;
Variables
   A1, A2, A : Tableau [1..100, 1..100] de réel;
   n, m, i, j : entier ;
Début
   Répéter
      Écrire("Donner le nombre de lignes :") ;
      Lire(n) ;
   Jusqu'à (n > 0 et n <= 100) ;

   Répéter
      Écrire("Donner le nombre de colonnes :") ;
      Lire(m) ;
   Jusqu'à (m > 0 et m <= 100) ;

   Pour i ← 1 à n faire
      Pour j ← 1 à m faire
         Lire(A1[i, j]) ;
      Fin-Pour ;
   Fin-Pour ;

   Pour i ← 1 à n faire
      Pour j ← 1 à m faire
         Lire(A2[i, j]) ;
      Fin-Pour ;
   Fin-Pour ;

   Pour i ← 1 à n faire
      Pour j ← 1 à m faire
         A[i, j] ← A1[i, j] - A2[i, j] ;
      Fin-Pour ;
   Fin-Pour ;

   Pour i ← 1 à n faire
      Pour j ← 1 à m faire
         Écrire(A[i, j]) ;
      Fin-Pour ;
   Fin-Pour ;
Fin.
```

En langage C :

```c
#include <stdio.h>

int main() {
    float A1[100][100], A2[100][100],
A[100][100];
  int n, m, i, j;

    do {
    printf("Donner le nombre de lignes : ");
        scanf("%d", &n);
    } while (n <= 0 || n > 100);

    do {
   printf("Donner le nombre de colonnes :");
        scanf("%d", &m);
    } while (m <= 0 || m > 100);

    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%f", &A1[i][j]);
        }
    }

    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            scanf("%f", &A2[i][j]);
        }
    }

    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            A[i][j] = A1[i][j] - A2[i][j];
        }
    }

    for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++) {
            printf("%.2f ", A[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

4.9 Conclusion

Dans ce chapitre, nous avons introduit les tableaux comme première structure de données, en expliquant leur définition et leur manipulation. En effet, les vecteurs et les matrices constituent la base pour traiter des collections d'éléments en algorithmique et en C.