

Chapitre V

Les TABLEAUX

Objectifs

Général

⇒ Assimiler le concept des tableaux dans la programmation C.

Spécifiques

- ❑ Se familiariser avec les tableaux à une ou plusieurs dimensions
 - ❑ Connaître les différents traitements qu'on peut exécuter sur les tableaux ;
 - ❑ Comprendre l'importance des tableaux dans la structuration des programmes.
-

1. Introduction

Les tableaux sont certainement les variables structurées les plus populaires. Ils sont disponibles dans tous les langages de programmation et servent à résoudre une multitude de problèmes.

Dans une première approche, le traitement des tableaux en C ne diffère pas de celui des autres langages de programmation.

Les chaînes de caractères sont déclarées en C comme tableaux de caractères et permettent l'utilisation d'un certain nombre de notations et de fonctions spéciales.

2. Les tableaux à une dimension

Un tableau (uni-dimensionnel) A est une variable structurée formée d'un nombre entier N de variables simples du même type, qui sont appelées les **composantes** du tableau. Le nombre de composantes N est alors la **dimension** du tableau.



En faisant le rapprochement avec les mathématiques, on dit encore que "A est un **vecteur** de dimension N".

Exemple

La déclaration `int JOURS[12]={31,28,31,30,31,30,31,31,30,31,30,31};` définit un tableau du type **int** de dimension 12. Les 12 composantes sont initialisées par les valeurs respectives 31, 28, 31, ... , 31.

On peut accéder à la première composante du tableau par `JOURS[0]`, à la deuxième composante par `JOURS[1]`, ... , à la dernière composante par `JOURS[11]`.

2.1. Déclaration et mémorisation

→ Déclaration de tableaux en langage algorithmique

<TypeSimple> tableau <NomTableau>[<Dimension>]

→ Déclaration de tableaux en C

<TypeSimple> <NomTableau>[<Dimension>;

Exemple

Les déclarations suivantes en langage algorithmique,

*entier tableau A[25]
réel tableau B[100]
booléen tableau C[10]
caractère tableau D[30]*

se laissent traduire en C par:

*int A[25];
float B[100];
int C[10];
char D[30];*

2.2. Initialisation

Lors de la déclaration d'un tableau, on peut initialiser les composantes du tableau, en indiquant la liste des valeurs respectives entre accolades.

Exemple 1

*int A[5] = {10, 20, 30, 40, 50};
float B[4] = {-1.05, 3.33, 87e-5, -12.3E4};
int C[10] = {1, 0, 0, 1, 1, 1, 0, 1, 0, 1};*

Il faut évidemment veiller à ce que le nombre de valeurs dans la liste corresponde à la dimension du tableau. Si la liste ne contient pas assez de valeurs pour toutes les composantes, les composantes restantes sont initialisées par zéro.

Exemple 2

short A[] = {1200, 2300, 3400, 4500, 5600}; ✓

A:

1200	2300	3400	4500	5600
------	------	------	------	------

short A[5] = {1200, 2300, 3400}; ✓

A:

1200	2300	3400	0	0
------	------	------	---	---

short A[3] = {1200, 2300, 3400, 4500, 5600}; ✗

A:

1200	2300	3400	✗	✗
------	------	------	---	---

↪ ERREUR !

2.3. Accès aux composantes

En déclarant un tableau par: *int A[5];*

Nous avons défini un tableau A avec cinq composantes, auxquelles on peut accéder par: A[0], A[1], ... , A[4].

Exemple

short A[5] = {1200, 2300, 3400, 4500, 5600};					
A:	1200	2300	3400	4500	5600
	A[0]	A[1]	A[2]	A[3]	A[4]

Attention !

Considérons un tableau T de dimension N:

En C,

- ✓ L'accès au premier élément du tableau se fait par **T[0]**
- ✓ L'accès au dernier élément du tableau se fait par **T[N-1]**

En langage algorithmique,

- ✓ L'accès au premier élément du tableau se fait par **T[1]**
- ✓ L'accès au dernier élément du tableau se fait par **T[N]**

2.4. Affichage et affectation

La structure **for** se prête particulièrement bien au travail avec les tableaux. La plupart des applications se laissent implémenter par simple modification des exemples-types de l'affichage et de l'affectation.

2.4.1. Affichage du contenu d'un tableau

Traduisons le programme AFFICHER du langage algorithmique en C:

```
programme AFFICHER
| entier tableau A[5]
| entier I (* Compteur *)
| pour I variant de 1 à 5 faire
|   écrire A[I]
| fpour
fprogramme
```

se laissent traduire en C par:

```
int main( )
{
    int A[5];
    int I; /* Compteur */
    for (I=0; I<5; I++) printf("%d ", A[I]);
    return 0;
    printf("\n");
}
```

Remarques

- Avant de pouvoir afficher les composantes d'un tableau, il faut évidemment leur affecter des valeurs.
- Rappelez-vous que la deuxième condition dans la structure **for** n'est pas une condition d'arrêt, mais une condition de répétition! Ainsi la commande d'affichage sera répétée *aussi longtemps* que **I** est inférieur à 5. La boucle sera donc bien exécutée pour les indices 0,1,2,3 et 4.
- Par opposition à la commande simplifiée écrire A[i] du langage algorithmique, la commande **printf** doit être informée du type exact des données à afficher.

2.4.2. Affectation avec des valeurs provenant de l'extérieur

Traduisons le programme **REEMPLIR** du langage algorithmique en C:

```

programme REEMPLIR
|  entier tableau A[5]
|  entier I (* Compteur *)
|  pour I variant de 1 à 5 faire
|    lire A[I]
|  fpour
fprogramme

```

se laissent traduire en C par:

```

int main( )
{
    int A[5];
    int I; /* Compteur */
    for (I=0; I<5; I++) scanf("%d", &A[I]);
    return 0;
}

```

Remarques

- Comme **scanf** a besoin des adresses des différentes composantes du tableau, il faut faire précéder le terme A[I] par l'opérateur adresse '&'.
- La commande de lecture **scanf** doit être informée du type exact des données à lire. (Ici: **%d** pour lire des valeurs du type **int**).

Exercice 5.1

Ecrire un programme qui lit la dimension N d'un tableau T du type **int** (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Calculer et afficher ensuite la somme des éléments du tableau.

Solution

```

#include <stdio.h>
int main( )
{
    /* Déclarations */
    int T[50]; /* tableau donné */
    int N;     /* dimension */
    int I;     /* indice courant */
    long SOM;  /* somme des éléments - type long à cause de la grandeur
prévisible du résultat. */

    /* Saisie des données */
    printf("Dimension du tableau (max.50) : ");
    scanf("%d", &N );
    for (I=0; I<N; I++)
    {
        printf("Elément %d : ", I);
        scanf("%d", &T[I]);
    }
    /* Affichage du tableau */
    printf("Tableau donné :\n");
    for (I=0; I<N; I++) printf("%d ", T[I]);
    printf("\n");
}

```

```

    /* Calcul de la somme */
    for (SOM=0, I=0; I<N; I++) SOM += T[I];
    /* Edition du résultat */
    printf("Somme de éléments : %ld\n", SOM);
    return 0;
}

```

Exercice 5.2

Ecrire un programme qui lit la dimension N d'un tableau T du type `int` (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Effacer ensuite toutes les occurrences de la valeur 0 dans le tableau T et tasser les éléments restants. Afficher le tableau résultant.

Solution

```

#include <stdio.h>
int main( )
{
    /* Déclarations */
    int T[50]; /* tableau donné */
    int N;     /* dimension */
    int I, J;  /* indices courants */

    /* Saisie des données */
    printf("Dimension du tableau (max.50) : ");
    scanf("%d", &N );
    for (I=0; I<N; I++)
    {
        printf("Elément %d : ", I);
        scanf("%d", &T[I]);
    }
    /* Affichage du tableau */
    printf("Tableau donné : \n");
    for (I=0; I<N; I++) printf("%d ", T[I]);
    printf("\n");
    /* Effacer les zéros et comprimer : Copier tous les éléments de I vers
    J et augmenter J pour les éléments non nuls. */
    for (I=0, J=0 ; I<N ; I++)
    {
        T[J] = T[I];
        if (T[I]) J++;
    }
    /* Nouvelle dimension du tableau ! */
    N = J;
    /* Edition des résultats */
    printf("Tableau résultat : \n");
    for (I=0; I<N; I++) printf("%d ", T[I]);
    printf("\n");
    return 0;
}

```

Exercice 5.3

Ecrire un programme qui lit la dimension N d'un tableau T du type `int` (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Ranger ensuite les éléments du tableau T dans l'ordre inverse sans utiliser de tableau d'aide. Afficher le tableau résultant.

Idée: Echanger les éléments du tableau à l'aide de deux indices qui parcourent le tableau en commençant respectivement au début et à la fin du tableau et qui se rencontrent en son milieu.

Solution

```
#include <stdio.h>
main( )
{
    /* Déclarations */
    int T[50]; /* tableau donné */
    int N;      /* dimension      */
    int I,J;    /* indices courants */
    int AIDE;   /* pour l'échange   */

    /* Saisie des données */
    printf("Dimension du tableau (max.50) : ");
    scanf("%d", &N );
    for (I=0; I<N; I++)
    {
        printf("Elément %d : ", I);
        scanf("%d", &T[I]);
    }
    /* Affichage du tableau */
    printf("Tableau donné : \n");
    for (I=0; I<N; I++) printf("%d ", T[I]);
    printf("\n");
    /* Inverser le tableau */
    for (I=0, J=N-1 ; I<J ; I++,J--)
        /* Echange de T[I] et T[J] */
        {
            AIDE = T[I];
            T[I]  = T[J];
            T[J]  = AIDE;
        }
    /* Edition des résultats */
    printf("Tableau résultat : \n");
    for (I=0; I<N; I++) printf("%d ", T[I]);
    printf("\n");
    return 0;
}
```

Exercice 5.4

Ecrire un programme qui lit la dimension N d'un tableau T du type **int** (dimension maximale: 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Copiez ensuite toutes les composantes strictement positives dans un deuxième tableau TPOS et toutes les valeurs strictement négatives dans un troisième tableau TNEG. Afficher les tableaux TPOS et TNEG.

Solution

```
#include <stdio.h>
main( )
{
    /* Déclarations */
    /* Les tableaux et leurs dimensions */
    int T[50], TPOS[50], TNEG[50];
    int N,      NPOS,      NNEG;
```

```

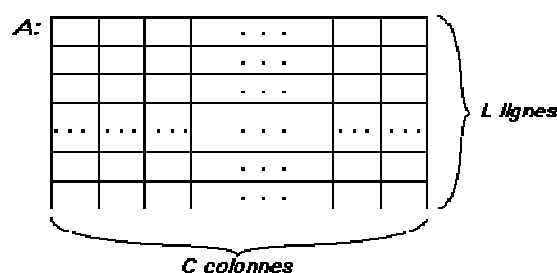
    int I; /* indice courant */

    /* Saisie des données */
    printf("Dimension du tableau (max.50) : ");
    scanf("%d", &N );
    for (I=0; I<N; I++)
    {
        printf("Elément %d : ", I);
        scanf("%d", &T[I]);
    }
    /* Affichage du tableau */
    printf("Tableau donné :\n");
    for (I=0; I<N; I++) printf("%d ", T[I]);
    printf("\n");
    /* Initialisation des dimensions de TPOS et TNEG */
    NPOS=0;
    NNEG=0;
    /* Transfer des données */
    for (I=0; I<N; I++)
    {
        if (T[I]>0)
        {
            TPOS[NPOS]=T[I];
            NPOS++;
        }
        if (T[I]<0)
        {
            TNEG[NNEG]=T[I];
            NNEG++;
        }
    }
    /* Edition du résultat */
    printf("Tableau TPOS :\n");
    for (I=0; I<NPOS; I++) printf("%d ", TPOS[I]);
    printf("\n");
    printf("Tableau TNEG :\n");
    for (I=0; I<NNEG; I++) printf("%d ", TNEG[I]);
    printf("\n");
    return 0;
}

```

3. Les tableaux à deux dimensions

En C, un tableau à deux dimensions A est à interpréter comme un tableau (uni-dimensionnel) de dimension L dont chaque composante est un tableau (uni-dimensionnel) de dimension C . On appelle L le **nombre de lignes** du tableau et C le **nombre de colonnes** du tableau. L et C sont alors les deux **dimensions** du tableau. Un tableau à deux dimensions contient donc $L \times C$



On dit qu'un tableau à deux dimensions est **carré**, si L est égal à C.

En faisant le rapprochement avec les mathématiques, on peut dire que "*A est un vecteur de L vecteurs de dimension C*", ou mieux:

*"A est une **matrice** de dimensions L et C"*.

Exemple

Considérons un tableau NOTES à une dimension pour mémoriser les notes de 20 élèves d'une classe dans un devoir:

int NOTE[20] = {45, 34, ..., 50, 48};

NOTE :	45	34	...	50	48
	A[0]	A[1]		A[18]	A[19]

Pour mémoriser les notes des élèves dans les 10 devoirs d'un trimestre, nous pouvons rassembler plusieurs de ces tableaux uni-dimensionnels dans un tableau NOTES à deux dimensions :

*int NOTE[10][20] = {{45, 34, ..., 50, 48},
 {39, 24, ..., 49, 45},
 ..., ..., ...,
 {40, 40, ..., 54, 44}};*

NOTE :	45	34	...	50	48
	39	24	...	49	45
	...	...	...	...	...
	40	40	...	54	44

} **10 lignes**

20 colonnes

Dans une ligne, nous retrouvons les notes de tous les élèves dans un devoir. Dans une colonne, nous retrouvons toutes les notes d'un élève.

3.1. Déclaration

→ Déclaration de tableaux à deux dimensions en lang. algorithmique

<TypeSimple> tableau <NomTabl>[<DimLigne>,<DimCol>]

→ Déclaration de tableaux à deux dimensions en C

<TypeSimple> <NomTabl>[<DimLigne>][<DimCol>;]

Exemple

Les déclarations suivantes en langage algorithmique,

*entier tableau A[10,10]
 réel tableau B[2,20]
 booléen tableau C[3,3]
 caractère tableau D[15,40]*

se laissent traduire en C par:

```
int A[10][10];
float B[2][20];
int C[3][3];
char D[15][40];
```

3.2. Initialisation

Lors de la déclaration d'un tableau, on peut initialiser les composantes du tableau, en indiquant la liste des valeurs respectives entre accolades. A l'intérieur de la liste, les composantes de chaque ligne du tableau sont encore une fois comprises entre accolades. Pour améliorer la lisibilité des programmes, on peut indiquer les composantes dans plusieurs lignes.

Exemple

```
int A[3][10] = {{ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90},
                {10, 11, 12, 13, 14, 15, 16, 17, 18, 19},
                { 1, 12, 23, 34, 45, 56, 67, 78, 89, 90}};
```

Lors de l'initialisation, les valeurs sont affectées ligne par ligne en passant de gauche à droite. Nous ne devons pas nécessairement indiquer toutes les valeurs: Les valeurs manquantes seront initialisées par zéro. Il est cependant défendu d'indiquer trop de valeurs pour un tableau.

3.3. Accès aux composantes

→ Accès à un tableau à deux dimensions en lang. algorithmique

<NomTableau>[<Ligne>, <Colonne>]

→ Accès à un tableau à deux dimensions en C

<NomTableau>[<Ligne>][<Colonne>]

Les éléments d'un tableau de dimensions L et C se présentent de la façon suivante:

$$\begin{array}{ccccccc} A[0][0] & A[0][1] & A[0][2] & \dots & A[0][C-1] \\ A[1][0] & A[1][1] & A[1][2] & \dots & A[1][C-1] \\ A[2][0] & A[2][1] & A[2][2] & \dots & A[2][C-1] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A[L-1][0] & A[L-1][1] & A[L-1][2] & \dots & A[L-1][C-1] \end{array}$$

Attention !

Considérons un tableau A de dimensions L et C.

En C,

- ✓ Les indices du tableau varient de 0 à L-1, respectivement de 0 à C-1.
- ✓ La composante de la N^{ième} ligne et M^{ième} colonne est notée: **A[N-1][M-1]**

En langage algorithmique,

- ✓ Les indices du tableau varient de 1 à L, respectivement de 1 à C.
- ✓ La composante de la N^{ième} ligne et M^{ième} colonne est notée: **A[N,M]**

3.4. Affichage et affectation

Lors du travail avec les tableaux à deux dimensions, nous utiliserons deux indices (p.ex: I et J), et la structure **for**, souvent imbriquée, pour parcourir les lignes et les colonnes des tableaux.

3.4.1. Affichage du contenu d'un tableau à deux dimensions

Traduisons le programme AFFICHER du langage algorithmique en C:

```
programme AFFICHER
|  entier tableau A[5,10]
|  entier I,J
|  (* Pour chaque ligne ... *)
|  pour I variant de 1 à 5 faire
|    (* ... considérer chaque composante *)
|    pour J variant de 1 à 10 faire
|      écrire A[I,J]
|    fpour
|    (* Retour à la ligne *)
|    écrire
|  fpour
fprogramme
```

se laissent traduire en C par:

```
int main( )
{
    int A[5][10];
    int I,J;
    /* Pour chaque ligne ... */
    for (I=0; I<5; I++)
    {
        /* ... considérer chaque composante */
        for (J=0; J<10; J++) printf("%7d", A[I][J]);
        /* Retour à la ligne */
        printf("\n");
    }
    return 0;
}
```

3.4.2. Affectation avec des valeurs provenant de l'extérieur

Traduisons le programme REMPLIR du langage algorithmique en C:

```
programme REMPLIR
|  entier tableau A[5,10]
|  entier I,J
|  (* Pour chaque ligne ... *)
|  pour I variant de 1 à 5 faire
|    (* ... considérer chaque composante *)
|    pour J variant de 1 à 10 faire
|      lire A[I,J]
|    fpour
|  fpour
fprogramme
```

se laissent traduire en C par:

```

int main( )
{
    int A[5][10];
    int I, J;
    /* Pour chaque ligne ... */
    for (I=0; I<5; I++)
        /* ... considérer chaque composante */
        for (J=0; J<10; J++) scanf("%d", &A[I][J]);
    return 0;
}

```

Exercice 5.5

Ecrire un programme qui lit les dimensions L et C d'un tableau T à deux dimensions du type **int** (dimensions maximales: 50 lignes et 50 colonnes). Remplir le tableau par des valeurs entrées au clavier et afficher le tableau ainsi que la somme de tous ses éléments.

Solution

```

#include <stdio.h>
int main( )
{
    /* Déclarations */
    int T[50][50]; /* tableau donné */
    int L, C; /* dimensions */
    int I, J; /* indices courants */
    long SOM; /* somme des éléments - type long à cause de la grandeur
    prévisible du résultat. */

    /* Saisie des données */
    printf("Nombre de lignes (max.50) : ");
    scanf("%d", &L );
    printf("Nombre de colonnes (max.50) : ");
    scanf("%d", &C );
    for (I=0; I<L; I++)
        for (J=0; J<C; J++)
        {
            printf("Elément[%d][%d] : ", I, J);
            scanf("%d", &T[I][J]);
        }

    /* Affichage du tableau */
    printf("Tableau donné :\n");
    for (I=0; I<L; I++)
    {
        for (J=0; J<C; J++) printf("%7d", T[I][J]);
        printf("\n");
    }

    /* Calcul de la somme */
    for (SOM=0, I=0; I<L; I++)
        for (J=0; J<C; J++) SOM += T[I][J];
    /* Edition du résultat */
    printf("Somme des éléments : %ld\n", SOM);
    return 0;
}

```

Exercice 5.6

Ecrire un programme qui lit les dimensions L et C d'un tableau T à deux dimensions du type **int** (dimensions maximales: 50 lignes et 50 colonnes). Remplir le tableau par des

valeurs entrées au clavier et afficher le tableau ainsi que la somme de chaque ligne et de chaque colonne en n'utilisant qu'une variable d'aide pour la somme.

Solution

```
#include <stdio.h>
main( )
{
    /* Déclarations */
    int T[50][50]; /* tableau donné */
    int L, C;      /* dimensions */
    int I, J;      /* indices courants */
    long SOM;      /* somme des éléments - type long à cause de la grandeur
prévisible des résultats. */

    /* Saisie des données */
    printf("Nombre de lignes (max.50) : ");
    scanf("%d", &L );
    printf("Nombre de colonnes (max.50) : ");
    scanf("%d", &C );
    for (I=0; I<L; I++)
        for (J=0; J<C; J++)
        {
            printf("Elément[%d][%d] : ",I,J);
            scanf("%d", &T[I][J]);
        }
    /* Affichage du tableau */
    printf("Tableau donné :\n");
    for (I=0; I<L; I++)
    {
        for (J=0; J<C; J++) printf("%7d", T[I][J]);
        printf("\n");
    }
    /* Calcul et affichage de la somme des lignes */
    for (I=0; I<L; I++)
    {
        for (SOM=0, J=0; J<C; J++) SOM += T[I][J];
        printf("Somme - ligne %d : %ld\n",I,SOM);
    }
    /* Calcul et affichage de la somme des colonnes */
    for (J=0; J<C; J++)
    {
        for (SOM=0, I=0; I<L; I++) SOM += T[I][J];
        printf("Somme - colonne %d : %ld\n",J,SOM);
    }
    return 0;
}
```

4. Exercices d'application

Exercice 5.7

Ecrire un programme C qui permet de lire les âges des 30 étudiants d'une classe et fournir le nombre des étudiants de chacun des âges de 18 à 22.

Exercice 5.8

Ecrire un programme C qui lit un tableau de 10 entiers et met les entiers négatifs à droite et les entiers positifs à gauche.

Exercice 5.9

Un tableau A de dimension N+1 contient N valeurs entières triées par ordre croissant; la (N+1)^{ième} valeur est indéfinie. Ecrire un programme C qui permet d'insérer une valeur VAL donnée au clavier dans le tableau A de manière à obtenir un tableau de N+1 valeurs triées.

Exercice 5.10

On dispose de deux tableaux A et B (de dimensions respectives N et M), triés par ordre croissant. Ecrire un programme C permettant de fusionner les éléments de A et B dans un troisième tableau FUS trié par ordre croissant.

Exercice 5.11

Ecrire un programme C qui permet d'effectuer la permutation circulaire des valeurs d'un tableau de N nombres entiers : chaque élément doit prendre la valeur du suivant sauf le dernier qui prend la valeur du premier.

Exercice 5.12

Lire les âges des 40 élèves d'une classe de lycée (ils ont tous au moins 14 ans, au plus 22 ans) et fournir le nombre des élèves de chacun des âges.

Exercice 5.13

On dispose de deux tableaux A et B (de dimensions respectives N et M au maximum). Ecrire un programme C qui permet de remplir les deux tableaux puis comparer leurs tailles. En cas d'égalité entre les tailles des deux tableaux, copier le tableau A dans B et en cas contraire copier le tableau ayant la taille la plus petite dans celui de taille plus grande puis terminer le remplissage des cases restantes par des zéros.

Exercice 5.14

Ecrire un programme C qui construit et affiche une matrice *carrée unitaire* U de dimension N. Une matrice unitaire est une matrice, telle que:

$$U_{ij} = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases}$$

Exercice 5.15

Ecrire un programme C qui effectue la transposition d'une matrice A de dimensions N et M en une matrice de dimensions M et N.

- a) La matrice transposée sera mémorisée dans une deuxième matrice B qui sera ensuite affichée.
- b) La matrice A sera transposée par permutation des éléments.

Exercice 5.16

Ecrire un programme qui réalise la multiplication d'une matrice A par un réel X.

- a) Le résultat de la multiplication sera mémorisé dans une deuxième matrice B qui sera ensuite affichée.
 - b) Les éléments de la matrice A seront multipliés par X.
-