Chapitre 6 :
Les tableaux à deux dimensions

Définition

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.

5.1. Déclaration et mémorisation

Déclarations

Déclaration de tableaux à deux dimensions en lang. algorithmique

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

Déclaration de tableaux à deux dimensions en C

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

Exemples

Les déclarations suivantes en langage algorithmique,

entier tableau A[10,10]
réel tableau B[2,20]
caractère tableau D[15,40]

se laissent traduire en C par:

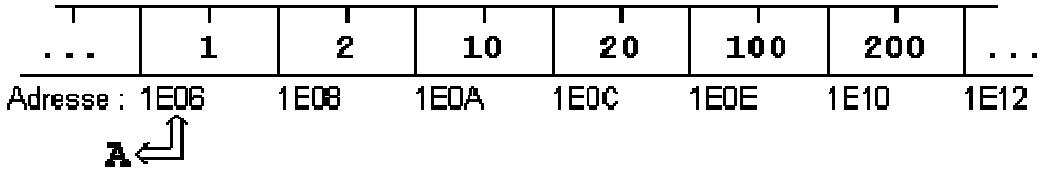
int A[10][10];	**ou bien**	**long A[10][10];**
float B[2][20];	**ou bien**	**double B[2][20];**
int C[3][3];		
char D[15][40];		

Mémorisation

Comme pour les tableaux à une dimension, le nom d'un tableau est le représentant de ***l'adresse du premier élément*** du tableau (c.-à-d. l'adresse de la première ***ligne*** du tableau). Les composantes d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire

Exemple: Mémorisation d'un tableau à deux dimensions

```
short A[3][2] = {{1,   2  },
                 {10,  20 },
                 {100, 200}};
```



Un tableau de dimensions L et C, formé de composantes dont chacune a besoin de M octets, occupera L*C*M octets en mémoire

Exemple

En supposant qu'une variable du type **double** occupe 8 octets (c.à d: **sizeof(double)=8**), pour le tableau T déclaré par: **double T[10][15];** C réservera L*C*M = 10*15*8 = 1200 octets en mémoire.

5. 2. Initialisation et réservation automatique

Exemples

```
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}};
float B[3][2] = {{-1.05,   -1.10  },
                 {86e-5,    87e-5 },
                 {-12.5E4, -12.3E4}};
```

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.

Exemples

```
int C[4][4] = {{1, 0, 0, 0}
               {1, 1, 0, 0}
               {1, 1, 1, 0}
               {1, 1, 1, 1}};
```

✓

C:

1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1

```
int C[4][4] ={{1, 1, 1, 1}};
```

C:

1	1	1	1
0	0	0	0
0	0	0	0
0	0	0	0

```
int C[4][4] ={{1}, {1}, {1}, {1}};
```

C:

1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0

```
int C[4][4] ={{1, 1, 1, 1, 1}};
```

ERREUR !

Si le nombre de **lignes L** n'est pas indiqué explicitement lors de l'initialisation, l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

```
int A[][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}};
```

réservation de 3*10*2 = 60 octets

```
float B[][2] = {{-1.05,   -1.10  },
                {86e-5,    87e-5 },
                {-12.5E4, -12.3E4}};
```

réservation de 3*2*4 = 24 octets

Exemple

```
int C[][4] = {{1, 0, 0, 0}
              {1, 1, 0, 0}
              {1, 1, 1, 0}
              {1, 1, 1, 1}};
```

C:

1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1

⇒ réservation de 4*4*2 = 32 octets

5. 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:

```
/                                          \
|  A[0][0]   A[0][1]   A[0][2]   . . .   A[0][C-1]  |
|  A[1][0]   A[1][1]   A[1][2]   . . .   A[1][C-1]  |
|  A[2][0]   A[2][1]   A[2][2]   . . .   A[2][C-1]  |
|  . . .     . . .     . . .     . . .     . . .    |
| A[L-1][0]  A[L-1][1]  A[L-1][2]  . . .  A[L-1][C-1] |
\                                          /
```

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]

5.4. 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.

- 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
```

```
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;
}
```

Remarques

* Avant de pouvoir afficher les composantes d'un tableau, il faut leur affecter des valeurs.

* Pour obtenir des colonnes bien alignées lors de l'affichage, il est pratique d'indiquer la largeur minimale de l'affichage dans la chaîne de format. Pour afficher des matrices du type **int** (valeur la plus 'longue': -32768), nous pouvons utiliser la chaîne de format **"%7d"** :

printf("%7d", A[I][J]);

- 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
```

```
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;
}
```