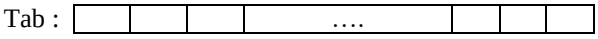
Les tableaux sont certainement les variables structurées les plus populaires. Ils sont disponibles dans tous les langages de programmation et servent à résoudre une multitude de problèmes. Dans une première approche, le traitement des tableaux en C ne diffère pas de celui des autres langages de programmation. Nous allons cependant voir plus loin, que le langage C permet un accès encore plus direct et plus rapide aux données d'un tableau.

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. Les particularités des tableaux de caractères seront traitées séparément ultérieurement.

I) Les tableaux à une dimension :

Définitions :

Un tableau (uni-dimensionnel) Tab 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].

1) Déclaration et mémorisation :

a) Déclaration

Déclaration de tableaux en langage algorithmique :

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

Déclaration de tableaux en C :

```
<TypeSimple> <NomTableau>[<Dimension>];
```

Exemples :

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]; ou bien **long A[25];** ou bien ...
float B[100]; ou bien **double B[100];** ou bien ...
int C[10];
char D[30];

b) Mémorisation :

En C, le nom d'un tableau est le représentant de ***l'adresse du premier élément*** du tableau. Les adresses des autres composantes sont calculées (automatiquement) relativement à cette adresse.

Exemple :

```
short Tab[5] = {12, 23, 34, 45, 56};
```

….	12	23	34	45	56	….
Adresse :	3006	3008	3010	3012	3014	3016

Tab

Si un tableau est formé de N composantes et si une composante a besoin de M octets en mémoire, alors le tableau occupera de N*M octets.

Exemple :

En supposant qu'une variable du type **long** occupe 4 octets (c.-à-d : **sizeof(long)=4**), pour le tableau T déclaré par : **long T[15];**

C réservera N*M = 15*4 = 60 octets en mémoire.

2) Initialisation et réservation automatique :

Initialisation :

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

Exemples :

```
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.

Réservation automatique :

Si la dimension n'est pas indiquée explicitement lors de l'initialisation, alors l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

Exemples

int A[] = {10, 20, 30, 40, 50};
==> réservation de 5*sizeof(int) octets (dans notre cas : 10 octets)
float B[] = {-1.05, 3.33, 87e-5, -12.3E4};
==> réservation de 4*sizeof(float) octets (dans notre cas : 16 octets)
int C[] = {1, 0, 0, 1, 1, 1, 0, 1, 0, 1};
==> réservation de 10*sizeof(int) octets (dans notre cas : 20 octets)

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]
```

Exemples :

```
MAX = (A[0]>A[1]) ? A[0] : A[1];
A[4] *= 2;
```

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]**

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.

a) - 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
```

```
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. (Ici : **%d** ou **%i** pour des valeurs du type **int**)

* Pour être sûr que les valeurs sont bien séparées lors de l'affichage, il faut inclure au moins un espace dans la chaîne de format. Autres possibilités :

```
printf("%d\t", A[I]); /* tabulateur */
printf("%7d", A[I]); /* format d'affichage */
```

b) Affectation :

- Affectation avec des valeurs provenant de l'extérieur :

Traduisons le programme REMPLIR du langage algorithmique en C :

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

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** ou **%i** pour lire des valeurs du type **int**).

II) Les tableaux à deux dimensions :

Définitions :

En C, un tableau à deux dimensions A est à interpréter comme un tableau (unidimensionnel) de dimension L dont chaque composante est un tableau (unidimensionnel) 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*C composantes***.

On dit qu'un tableau à deux dimensions est ***carré***, si L est égal à 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};
```

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

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

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.

1) Déclaration et mémorisation :

a) Déclarations :

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>];
```

Exemples

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

b) 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'est-à-dire 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.

2) Initialisation et réservation automatique :

a) 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.

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.

Réservation automatique

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

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]

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.

a) - 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]);
```

b) - 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);
}
```