

Chapitre 5

Les tableaux

Tableaux

- Un **tableau** est une variable structurée composée d'un nombre de variables simples de même type désignées par un seul identificateur
- Ces variables simples sont appelées *éléments ou composantes* du tableau, elles sont stockées en mémoire à des emplacements contigus (l'un après l'autre)
- Le type des éléments du tableau peut être :
 - simple : char, int, float, double, ...
 - pointeur ou structure (chapitres suivants)
- On peut définir des tableaux :
 - à une dimension (tableau unidimensionnel ou vecteur)
 - à plusieurs dimensions (tableau multidimensionnel)

Déclaration des tableaux

- La déclaration d'un tableau à une dimension s'effectue en précisant le type de ses éléments et sa dimension (le nombre de ses éléments) :
 - Syntaxe en C : **Type identificateur[dimension];**
 - Exemple : **float notes[30];**
- La déclaration d'un tableau permet de lui réserver un espace mémoire dont la taille (en octets) est égal à : dimension * taille du type
- ainsi pour :
 - **short A[100];** // on réserve 200 octets (100* 2octets)
 - **char mot[10];** // on réserve 10 octets (10* 1octet)

Initialisation à la déclaration

- On peut initialiser les éléments d'un tableau lors de la déclaration, en indiquant la liste des valeurs respectives entre accolades. Ex:
 - `int A[5] = {1, 2, 3, 4, 5};`
 - `float B[4] = {-1.5, 3.3, 7e-2, -2.5E3};`
- Si la liste ne contient pas assez de valeurs pour toutes les composantes, les composantes restantes sont initialisées par zéro
 - Ex: `short T[10] = {1, 2, 3, 4, 5};`
- la liste ne doit pas contenir plus de valeurs que la dimension du tableau. Ex: `short T[3] = {1, 2, 3, 4, 5};` → Erreur
- Il est possible de ne pas indiquer la dimension explicitement lors de l'initialisation. Dans ce cas elle est égale au nombre de valeurs de la liste. Ex: `short T[] = {1, 2, 3, 4, 5};` → tableau de 5 éléments

Accès aux composantes d'un tableau

- L'accès à un élément du tableau se fait au moyen de l'indice. Par exemple, **T[i]** donne la valeur de l'élément i du tableau T
- En langage C l'indice du premier élément du tableau est 0. L'indice du dernier élément est égal à la dimension-1

Ex: int T[5] = {9, 8, 7, 6, 5}; →

T[0]=9, T[1]=8, T[2]=7, T[3]=6, T[4]=5

Remarques:

- on ne peut pas saisir, afficher ou traiter un tableau en entier, ainsi on ne peut pas écrire `printf(" %d",T)` ou `scanf(" %d",&T)`
- On traite les tableaux élément par élément de façon répétitive en utilisant des boucles

Tableaux : saisie et affichage

- Saisie des éléments d'un tableau T d'entiers de taille n :

```
for(i=0;i<n;i++)  
    { printf ("Entrez l'élément %d \n ",i + 1);  
      scanf(" %d" , &T[i]);  
    }
```

- Affichage des éléments d'un tableau T de taille n :

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

Tableaux : exemple

- Calcul du nombre d'étudiants ayant une note supérieure à 10:

```
main ( )
{
    float notes[30];
    int nbre,i;
    for(i=0;i<30;i++)
        { printf ("Entrez notes[%d] \n ",i);
          scanf(" %f" , &notes[i]);
        }
    nbre=0;
    for (i=0; i<30; i++)
        if (notes[i]>10) nbre+=1;
    printf (" le nombre de notes > à 10 est égal à : %d", nbre);

}
```

Tableaux à plusieurs dimensions

On peut définir un tableau à n dimensions de la façon suivante:

- **Type** `Nom_du_Tableau[D1][D2]...[Dn];` où D_i est le nombre d'éléments dans la dimension i
- **Exemple** : pour stocker les notes de 20 étudiants en 5 modules dans deux examens, on peut déclarer un tableau :

`float notes[20][5][2];`

(`notes[i][j][k]` est la note de l'examen k dans le module j pour l'étudiant i)

Tableaux à deux dimensions (Matrices)

- Syntaxe : `Type nom_du_Tableau[nombre ligne][nombre colonne];`
- Ex: **`short A[2][3];`** On peut représenter le tableau A de la manière suivante :

<code>A[0][0]</code>	<code>A[0][1]</code>	<code>A[0][2]</code>
<code>A[1][0]</code>	<code>A[1][1]</code>	<code>A[1][2]</code>

- Un tableau à deux dimensions `A[n][m]` est à interpréter comme un tableau unidimensionnel de dimension `n` dont chaque composante `A[i]` est un tableau unidimensionnel de dimension `m`.
- Un tableau à deux dimensions `A[n][m]` contient `n*m` composantes. Ainsi lors de la déclaration, on lui réserve un espace mémoire dont la taille (en octets) est égal à : `n*m* taille du type`

Initialisation à la déclaration d'une Matrice

- L'initialisation lors de la déclaration se fait en indiquant la liste des valeurs respectives entre accolades ligne par ligne

- Exemple :

- `float A[3][4] = {{-1.5, 2.1, 3.4, 0}, {8e-3, 7e-5, 1, 2.7 }, {3.1, 0, 2.5E4, -1.3E2}};`

`A[0][0]=-1.5 , A[0][1]=2.1, A[0][2]=3.4, A[0][3]=0`

`A[1][0]=8e-3 , A[1][1]=7e-5, A[1][2]=1, A[1][3]=2.7`

`A[2][0]=3.1 , A[2][1]=0, A[2][2]=2.5E4, A[2][3]=-1.3E2`

- On peut ne pas indiquer toutes les valeurs: Les composantes manquantes seront initialisées par zéro
- Comme pour les tableaux unidimensionnels, Il est défendu d'indiquer trop de valeurs pour une matrice

Matrices : saisie et affichage

- Saisie des éléments d'une matrice d'entiers $A[n][m]$:

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

- Affichage des éléments d'une matrice d'entiers $A[n][m]$:

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

Représentation d'un tableau en mémoire

- La déclaration d'un tableau provoque la réservation automatique par le compilateur d'une zone contiguë de la mémoire.
- La mémoire est une succession de cases mémoires. Chaque case est une suite de 8 bits (1 octet), identifiée par un numéro appelé **adresse**.
(on peut voir la mémoire comme une armoire constituée de tiroirs numérotés. Un numéro de tiroir correspond à une adresse)
- Les adresses sont souvent exprimées en hexadécimal pour une écriture plus compacte et proche de la représentation binaire de l'adresse. Le nombre de bits d'adressage dépend des machines.
- En C, l'**opérateur &** désigne **adresse de**. Ainsi, `printf(" adresse de a=%x ", &a)` affiche l'adresse de la variable a en hexadécimal

Représentation d'un tableau à une dimension en mémoire

- En C, le nom d'un tableau est le représentant de l'adresse du premier élément du tableau (pour un tableau T: **$T = \&T[0]$**)
- Les composantes du tableau étant stockées en mémoire à des emplacements contigus, les adresses des autres composantes sont calculées (automatiquement) relativement à cette adresse :
 $\&T[i] = \&T[0] + \text{sizeof}(\text{type}) * i$
- Exemple : **$\text{short } T[5] = \{100, 200, 300, 400, 500\};$**
et supposons que **$T = \&T[0] = 1E06$**
- On peut afficher et vérifier les adresses du tableau:
 **$\text{for}(i=0; i<5; i++)$
 $\text{printf("adresse de } T[\%d] = \%x\\n", i, \&T[i]);$**

1E05	
T → 1E06	100
1E08	200
1E0A	300
1E0C	400
1E0E	500
1E0F	

Représentation d'un tableau à deux dimensions en mémoire

- Les éléments d'un tableau sont stockés en mémoire à des emplacements contigus ligne après ligne
- Comme pour les tableaux unidimensionnels, le nom d'un tableau A à deux dimensions est le représentant de l'adresse du premier élément : **$A = \&A[0][0]$**
- Rappelons qu'une matrice $A[n][m]$ est à interpréter comme un tableau de dimension n dont chaque composante $A[i]$ est un tableau de dimension m.

$A[i]$ et $\&A[i][0]$ représentent l'adresse du 1^{er} élément de la ligne i (pour i de 0 à n-1)

- Exemple : **$\text{char } A[3][4];$** $A = \&A[0][0] = 0118$

