Université Mohammed V-Agdal
Faculté des sciences de Rabat
Département de Mathématique et Informatique

Langage C

Préparé et présenté par
Pr. M. Benchrifa

Année Universitaire 2006/2007

- ***CHAPITRE 6*** **: Tableaux**
  - Définition
  - Tableaux à une dimension (Vecteurs)
    - Déclaration ; Mémorisation ; …
  - Tableaux à plusieurs dimensions
    - Déclaration
    - Tableaux à deux dimensions (matrices) :
      - *Déclaration ; Mémorisation ; …*

Chap. 6 : Tableaux - Introduction

- Les variables, telles que nous les avons vues, ne permettent de stocker qu'une seule donnée à la fois.
- Pour mémoriser et manipuler de nombreuses données (100, 1000, …), des variables distinctes seraient beaucoup trop lourdes à gérer.
- Pour résoudre ce problème, le langage C (ainsi que les autres langages de programmations) propose une structure de données permettant de stocker l'ensemble de ces données dans une "variable commune" appelée :

Tableau

Chap. 6 : Tableaux

1. Définition

- On appelle tableau une variable composée de données de même type, stockée de manière contiguë en mémoire (les unes à la suite des autres).
- Un tableau est une suite des éléments de même taille. Par conséquent, la taille (en octet) du tableau est :

  taille (en octet) du type de donnée * le nombre d'élément
- Le type des éléments du tableau peut être :
  - simple : char, int, float, double, ...
    → tableau à une dimension ou tableau unidimensionnel
  - tableau
    → tableau à plusieurs dimensions ou tableau multidimensionnel
  - Autres : Pointeurs (clef chapitre 7) et Structures (clef chapitre 8)

Chap. 6 : Tableaux

2. Tableaux à une dimension

2.1 Déclaration

La déclaration d'un tableau à une dimension se fait de la façon suivante :

<Type Simple> Nom_du_Tableau [Nombre_Elements];

Type Simple : définit le type d'élément que contient le tableau
Nom_du_Tableau : est le nom que l'on décide de donner au tableau, le nom du tableau suit les mêmes règles qu'un nom de variable.
Nombre_Elements : est une expression constante entière positive.

Exemples :

```
char     caractere[12] ;        //Taille en octet : 1 octet  * 12 = 12 octets
int      entier[10] ;           //Taille en octet : 2 octets * 10 = 20 octets
float    reel_SP[8] ;           //Taille en octet : 4 octets * 8 = 32 octets
double   reel_DP[15] ;          //Taille en octet : 8 octets * 15 = 120 octets
```

Chap. 6 : Tableaux 2. Tableaux à une dimension

2.3 Initialisation à la déclaration

Il est possible d'initialiser le tableau à la définition :

<Type> Tableau [Nombre_Elements] = {C1, C2, … , Cn};

Où C1, C2, .., Cn sont des constantes dont le nombre ne doit pas dépasser le Nombre_Elements.

Si la liste de constantes ne contient pas assez de valeurs pour tous les éléments, les éléments restantes sont initialisées à zéro.

Exemples :

```
char   voyelles[6]            = { 'a' , 'e' , 'i' , 'o' , 'u' , 'y' } ;
int    Tableau_entier1[10]    = {10 , 5 , 9 , -2 , 011 , 0xaf , 0XBDE};
float  Tableau_reel[8]        = { 1.5 , 1.5e3 ,  0.7E4 };
Short A[3]                    = {12 , 23 , 34 , 45 , 56};          //Erreur !
int Tableau_entier2[] = { 15 , -8 , 027 , 0XABDE }   //Réservation automatique
                                                     //Tableau de 4 éléments
```

Chap. 6 : Tableaux 2. Tableaux à une dimension (Vecteurs)

2.4 Accès aux composantes d'un tableau

Pour accéder à un élément du tableau, il suffit de donner le nom du tableau, suivi de l'indice de l'élément entre crochets :

Nom_du_Tableau [indice]

Où indice est une expression entière positive ou nulle.

Un indice est toujours positif ou nul ;
L'indice du premier élément du tableau est 0 ;
L'indice du dernier élément du tableau est égal au nombre d'éléments – 1.

Exemples :

```
short A[5] = {12 , 23 , 34 , 45 , 56}; // A[0] = 12 ; A[1] = 23 ; A[2] = 34 ;
                                       //  A[3] = 45 ; A[4] = 56 ;

for( i = 0 ; i< 5 ; i++ )              //Affichage des éléments
  printf(''A[%d] = %d \t'',i,A[i]);    // du tableau A
```

Chap. 6 : Tableaux 2. Tableaux à une dimension (Vecteurs)

2.5 Remarques

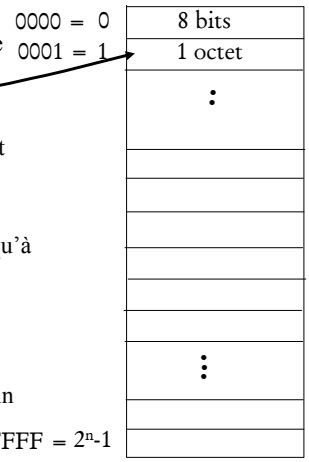
- Chaque élément (TAB[i]) d'un tableau (int TAB[20]) est manipulé comme une simple variable, on peut :
  - la lire : scanf("%d", &TAB[i]); // TAB[i] sera initialisé par un entier saisi //depuis la clavier
  - l'écrire : printf("TAB[%d] = %d", i , TAB[i]); //Le contenu de TAB[i] //sera affiché sur écran
  - la passer en argument à une fonction : TAB[i] = pow(TAB[i],2); // = TAB[i] ^{2}
- Pour initialiser un tableau (TAB1) par les éléments d'un autre tableau (TAB2) :
  - évitez d'écrire **TAB1 = TAB2** (**incorrect**)
  - On peut par exemple écrire :

```
for( i = 0 ; i < taille_tableau ; i++ )
    TAB1[i] = TAB2[i];
```

Chap. 6 : Tableaux 2. Tableaux à une dimension 2.1 Déclaration

2.6 Déclaration et représentation en mémoire d'un tableau:

- La déclaration d'une variable au sein d'un programme provoque la réservation automatique, par le compilateur, d'un espace de la mémoire. Ce qui permettra, par conséquent, de conserver les valeurs assignées à cette variable le long de l'exécution du programme.
- Rappelons que la mémoire (RAM) est une superposition de cases mémoires ou rangées.
- Chaque case mémoire est une suite de 8 bits (1 octets). La lecture ou l'écriture en mémoire se fait par octet ou par mot machine (2 octets ou 4 octets).
- Chaque case mémoire est identifiée par un numéro appelé **adresse**. Par convention, la première case mémoire est identifiée par l'adresse 0, la seconde par adresse 1, …, jusqu'à 2^n-1 avec n le nombre de bits pour l'adressage.
- Souvent on est ramené à exprimer cette adresse en hexadécimal. Une écriture plus compact proche de la représentation binaire de l'adresse et donne une idée sur le nombre des bits d'adressage. Dans le cas ou n = 16, on dit un adressage sur 16 bits. Dans des machines, on peut avoir un adressage sur n = 32 bits ou n = 64 bits.

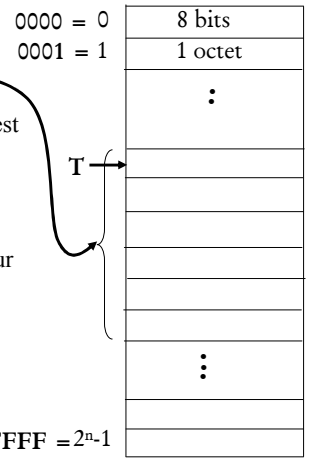


Chap. 6 : Tableaux 2. Tableaux à une dimension 2.1 Déclaration

2.2 Déclaration et représentation en mémoire d'un tableau (suite):

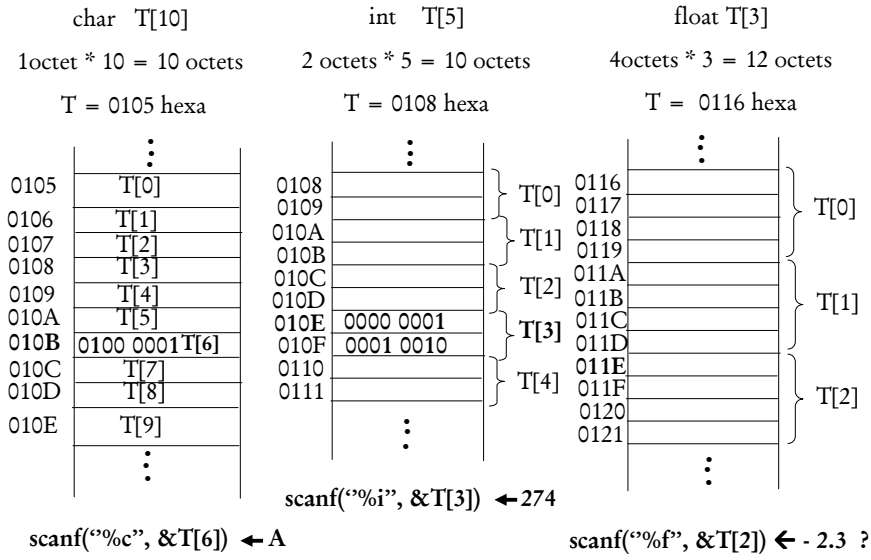
- En C, la déclaration d'un tableau T induit une réservation automatique d'une zone mémoire contiguë (les cases mémoire sont successives).
- Dés le déclaration, on connaît La taille d'un tableau T qui est conditionnée par le nombre N des éléments et leur type :
  Taille de T en octet = N * sizeof (type)
- En C, le nom d'un tableau **T** (l'identificateur) est un pointeur constant qui pointe sur le premier élément du tableau. Il contient l'adresse du premier élément du tableau.
- Etant donnée l'adresse du premier élément du tableau (matérialisée par le nom du tableau T), l'adresse du ième élément de T est donnée par :

  &T[i] = &T[0] + sizeof(type) * i ou
  &T[i] = T + sizeof(type) * i



Chap. 6 : Tableaux 2. Tableaux à une dimension 2.1 Déclaration

Exemples :



Chap. 6 : Tableaux 2. Tableaux à une dimension (Vecteurs)

2.5 Exemples

- Saisie et affichage des données d'un tableaux d'entiers de 20 éléments aux maximum.
- Déterminer la plus petite valeur du tableau d'entiers A. afficher ensuite la valeur et la position du minimum. Si le tableau contient plusieurs minimum, retenir la position du premier minimum rencontré.
- Recherche d'une valeur dans un tableau **:** Etant donnés un tableau et une valeur. On recherche cette valeur dans le tableau. S'il existe, on affiche sa position sinon on affiche un message d'erreur

Chap. 6 : Tableaux

3. Tableaux à plusieurs dimensions

3.1 Déclaration

De manière similaire, on peut déclarer un tableau à plusieurs dimensions :

<Type Simple> Nom_du_Tableau [Nbre_E_1] [Nbre_E_2]…[Nbre_E_N];

- **Chaque élément entre crochets désigne le nombre d'éléments dans chaque dimension ;**
- **Le nombre de dimension n'est pas limité.**

3.2 Tableaux à deux dimensions (matrices)

3.2.1Déclaration

<Type Simple> Nom_du_Tableau [Nombre_ligne] [Nombre_colonne];

Exemple :

```
int     T[3][4] ;          //Taille en octet : 2 octets * 3 * 4 = 24 octets
```

On peut représenter un tel tableau de la manière suivante :

T[0][0]	T[0][1]	T[0][2]	T[0][3]
T[1][0]	T[1][1]	T[1][2]	T[1][3]
T[2][0]	T[2][1]	T[2][2]	T[2][3]

Chap. 6 : Tableaux 3.2 Tableaux à deux dimensions (matrices)

3.2.2 Initialisation à la déclaration et accès aux éléments :

Les valeurs sont affectées ligne par ligne lors de l'initialisation à la déclaration;

Accès aux composantes se fait par : Nom_tableau[ligne][colonne].

Exemples :

```
float   A[3][2] = { {-1.05,-1.10} , {86e-5, 87e-5} ,  {-12.5E4} };
int B[4][4] = { {-1 , 10 , 013 , Oxfe} ,  {+8 , -077} , {} , {011,-14,0XAD} };
```

A ➡

-1.05	-1.10
86e-5	87e-5
-12.5E4	0.0

B ➡

-1	**10**	**013**	**0xFE**
+8	**-077**	**0**	**0**
0	**0**	**0**	**0**
011	**-14**	**0XAD**	**0**

Chap. 6 : Tableaux 3.2 Tableaux à deux dimensions (matrices)

3.2.3 Déclaration et représentation en mémoire d'un tableaux à 2 dimensions :

La déclaration d'un tableau T à deux dimensions (matrice) induit la réservation de l'espace mémoire nécessaire pour accueillir tous les éléments du tableau.
Les éléments de la matrice sont répartis en mémoires ligne par ligne.

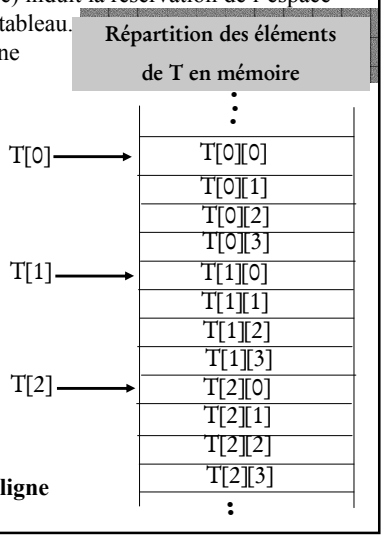
Par exemple, soit T un tableau défini par : char T[3][4]

Les écritures suivantes sont équivalentes :
T[0] ≡ &T[0][0] : adresse du 1er élément

T[1] ≡ &T[1][0] : adresse du 1er élément de la 2ème ligne

T[2] ≡ &T[2][0] : adresse du 1er élément de la 3ème ligne

T[i] ≡ &T[i][0] : adresse du 1er élément de la ième ligne



Chap. 6 : Tableaux 3.2 Tableaux à deux dimensions (matrices)

3.2.3 Exemples

- Saisie et affichage des données entières d'une matrice de M ligne et N colonnes. 20 éléments aux maximum. M et N sont entrées au clavier.

- Produit de deux matrices : En multipliant une matrice A de M lignes et N colonnes avec une matrice B de N lignes et P colonnes, on obtient une matrice C de M lignes et P colonnes :
  La composante cij de la matrice C, placée à la ième ligne et jème colonne, se calcule de la façon suivante :

$$c_{ij} = \sum_{k=1}^{M} a_{ik}\, b_{kj} \qquad \text{où } 1 \le i \le N \text{ et } 1 \le j \le P$$