Cours de Langage C
Les tableaux

Objectifs de la séance

- Les tableaux 1D en langage C
- Tableaux et fonctions
- Emplacement du début d'un tableau
- Les tableaux 2D
- Variante sur les notations
- Notion d'adresse

Tableaux à une dimension

(« Tableaux 1D »)

- On va se préoccuper des tableaux à 3 occasions :
  - **Déclaration** : Préciser le type des éléments d'un tableau

    1 seule fois

    tous les éléments sont du même type
  - **Initialisation**: Remplir un tableau avec des valeurs

    Peut être fait en une seule fois avec la déclaration

    Sinon, dans le code, les cases seront remplies une par une
  - **Utilisation** du tableau

    Le plus souvent case par case

    Ex : tableau[i] = 45 ;

    Parfois en un seul bloc , dans des fonctions *(voir fin de ce cours)*

Tableaux à une dimension

- **Déclaration d'un tableau**

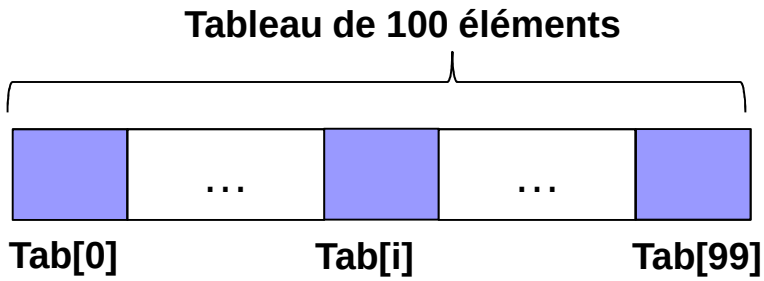
  int Tab[100] ;

  Déclare un tableau d'entiers de 100 éléments

  Allant de Tab[0] à Tab[99].

  → **Le 1er élément a l'indice 0** (et non 1 comme en Matlab)

  Le nom du tableau est Tab, ses éléments sont tous du même type et contigus en mémoire (une mémoire informatique a une structure 1D)



Attention aux problèmes de débordement : l'élément Tab[100] n'existe pas

Initialisation d'un tableau 1D

- **Initialisation à la déclaration**

  int tab[5] = {1, 2, 3, 4, 5};

- **Initialisation dans le programme**

  Par exemple avec une boucle

```
{…
int tab[5];
int i;
for (i = 0; i < 5; i++) tab[i] = i + 1;
}
```

- **Exemples de codes interdits**

  tab[5] ={1, 2, 3, 4, 5}; → interdit dans le code

  tab2 = tab; → interdit

Taille d'un tableau 1D

- La taille d'un tableau est connue à la compilation : c'est une constante symbolique ou bien un nombre.

oui

```
#define TAILLE 100
main( )
{
double tab1[TAILLE] ;
int tab2[5];
short tab3[2*TAILLE];
…
}
```

non

```
main( )
{
int n = 100 ;
double tab[n] ;
…
}
```

Erreur à la compilation
La taille d'un tableau
ne peut pas être une variable.

Fonctions et tableaux

- On est souvent amené à parcourir les éléments d'un tableau pour faire des opérations *(Moyenne, classement, …)*

  les fonctions sont indispensables pour ne pas avoir à réécrire un nouveau code pour chaque tableau.

- Un tableau peut donc être passé comme argument d'une fonction.
- On transmet à la fonction l'emplacement mémoire du début du tableau
- On parlera de l'adresse du tableau qui est égale à l'adresse du 1er élément du tableau : il s'agit du nom du tableau

  Ex : **int tab[10] ; tab** est l'adresse du tableau (et de tab[0])

Fonctions et tableaux :
déclaration, utilisation, définition

```
#define DIM 100
//Prototype
void initialise_tab(int tab[ ], int n, int valeur);
main()
{
int T[DIM]; //Création du tableau à initialiser
initialise_tab(T, DIM, 0); //Appel
}
//Définition
void initialise_tab(int tab[ ], int n, int valeur)
{
int i;
for (i = 0; i < n; i++) tab[i] = valeur;
}
```

Nb d'éléments sur lesquels on travaille

On ne spécifie pas la taille du tableau entre [ ]

qu'on veut initialiser

Fonctions et tableaux : *utilisation*

- Syntaxe pour transmettre à la fonction l'emplacement mémoire du début du tableau :

Rq : Le plus souvent, il faudra aussi transmettre la taille du tableau comme autre argument de la fonction.

```
#define DIM 100 /* DIM est la taille du tableau */
#define DIM2 200
…
double m1, m2 ;
m1=Moyenne( tab, DIM ); /* on calcule la moyenne de deux
                           tableaux de tailles différentes */
m2=Moyenne ( tab2, DIM2 ) ;
```

Tableaux à 2 dimensions (« 2D »)

- En C, on peut déclarer des tableaux à 2 indices

  int tab[5][3] ; // 5 lignes de 3 colonnes

- Plusieurs interprétations :
  - **tab[i][j]** pour i = 0,1,2,3,4 et j=0,1,2 est un entier
  - **tab[i]** pour i=0,1,2,3,4 est un tableau de 3 entiers
  - **tab** est un tableau dont chacune des 5 composantes est un tableau de 3 entiers

Tableaux à 2 dimensions (« 2D »)

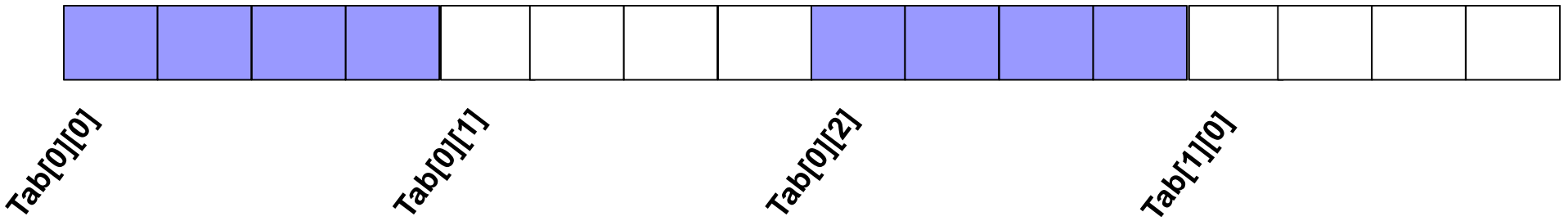
- Initialisation :
  - En bloc :

    int tab[2][3] = {{1,2,3}; {3,2,1}} ;

    int tab[2][3] = {1, 2, 3, 3, 2, 1} ;
  - Dans le programme avec deux boucles for.
- Utilisation des tableaux 2D
  - Comme pour les tableaux 1D

Tableaux à 2 dimensions (« 2D »)

- Positionnement en mémoire
  - La mémoire du PC a une structure 1D et non 2D
  - En mémoire, on aura pur **int tab[5][3] ;**



Nb : un int occupant 4 octets ⇨ les adresses évoluent de 4 en 4

Autre notation

- Le symbole **&** pour signifier **« adresse de … »**

```
moyenne (tab, DIM) ;
moyenne (&tab[0], DIM) ; Notations équivalentes
classerTableau (tab, DIM2) ;
classerTableau (&tab[0], DIM2) ;
```

- Ce symbole **&** pourra aussi servir à accéder à l'adresse d'une variable (en dehors de la notion de tableau)

```
int a = 8 ;
int b;
b = &a ;
```

En-tête de fonction

- Une fonction ne retourne qu'une valeur (au plus) mais peut maintenant changer N variables : par exemple avec la fonction « classerTableau »
- L'en-tête de fonction devra alors préciser le(s) tableau(x) modifiés par cette fonction

```
/*
Role : La fonction ClasserTableau() range dans
   l'ordre croissant les N elements tab[i]
Entrees : le tableau tab d'int, sa taille n
Sorties : tab
 */
void classerTableau(int tab[], int n) ;
```