UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

LIASD
LABORATOIRE INFORMATIQUE AVANCÉE DE SAINT-DENIS

Cours 5

Programmation impérative

Emna Chebbi, Revekka Kyriakoglou

Plan du cours

Définitions (1/2)

Tableaux

Le langage C permet d'utiliser des **tableaux**, c'est un ensemble d'éléments de même type désignés par un identificateur unique ; chaque élément est repéré par un **indice** précisant sa position au sein de l'ensemble.

Pointeurs

Le langage C dispose aussi de **pointeurs**, c'est-à-dire de variables destinées à contenir des adresses d'autres **objets** (variables, fonctions...).

Définitions (2/2)

Lien entre tableaux et pointeurs

A priori, ces deux notions de tableaux et de pointeurs peuvent paraître fort éloignées l'une de l'autre. Toutefois, il se trouve qu'en C un lien indirect existe entre ces deux notions, à savoir qu'un identificateur de tableau est une "constante pointeur". Cela peut se répercuter dans le traitement des tableaux, notamment lorsque ceux-ci sont transmis en argument de l'appel d'une fonction.

C'est ce qui justifie que ces deux notions soient regroupées dans un seul chapitre. Préparez-vous donc pour les pointeurs à la prochaine séance ;-).

Déclaration (1/2)

- Réserver l'emplacement pour N éléments de type int :

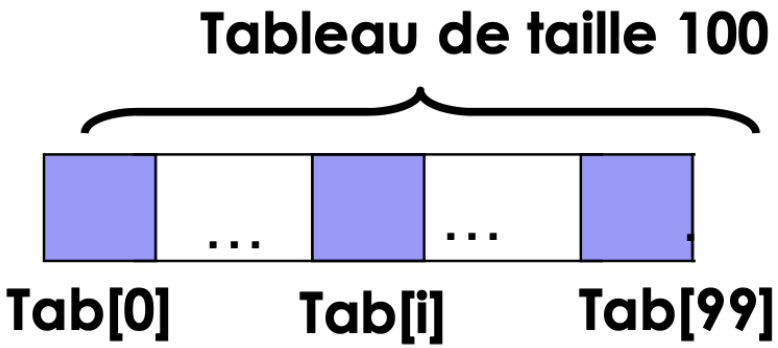
```
int t[N] ;
```

- Chaque élément est repéré par sa position dans le tableau, nommée indice.
- Conventionnellement en langage C, la première position porte le numéro 0. Ici, donc, nos indices vont de 0 à N-1.
- Désignation du premier, troisième et dernier élément :

```
t[0]
t[2]
...
t[N-1]
```

Déclaration (2/2)

- Exemple de N = 100 :



Plus généralement,

- Une notion telle que **t [i]** désigne un élément dont la position dans le tableau est fournie par la valeur de **i**.
- Elle joue le même rôle qu'une variable scalaire de type **int**.
- La notation **& t [i]** désigne l'adresse de cet élément **t [i]** de même que **&n** désignait l'adresse de **n**.

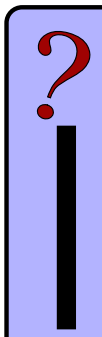
Initialisation

- 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;
}
```



Exemple d'utilisation d'un tableau

Supposons que nous souhaitions déterminer, à partir de vingt notes d'élèves (fournies en données), combien d'entre elles sont supérieures à la moyenne de la classe.

S'il ne s'agit que de calculer simplement la moyenne de ces notes, il suffirait d'en calculer la somme, en les cumulant dans une variable, au fur et à mesure de leur lecture.

Mais, comme il faut à nouveau pouvoir consulter les notes pour déterminer combien d'entre elles sont supérieures à la moyenne ainsi obtenue.
Il est donc nécessaire de pouvoir mémoriser ces vingt notes.

Solution (1/2)

 Pour ce faire, il paraît peut raisonnable de prévoir vingt variables scalaires différentes (méthode qui, de toute manière, serait difficilement transposable à un nombre important de notes).

 Le **tableau** va nous affrir une solution convenable à ce problème, comme le montre le programme suivant :

```
#include <stdio.h>
int main(void){
    int i, som, nbm ;
    float moy ;
    int t[20] ;
```

Solution (2/2)

```
for (i=0 ; i<20 ; i++){
      printf ("donnez␣la␣note␣numero␣%d␣:␣", i+1) ;
      scanf ("%d", &t[i]) ;
    }
    for (i=0, som=0 ; i<20 ; i++)
        som += t[i] ;
moy = som / 20 ;
printf ("\n␣moyenne␣de␣la␣classe␣:␣%f\n", moy) ;
for (i=0, nbm =0 ; i<20 ; i++)
    if (t[i] > moy)
        nbm++ ;
printf ("eleves␣ont␣plus␣de␣cette␣moyenne␣%d", nbm) ;
}
```

Éléments de tableau

- Un élément de tableau est une lvalue (left value). Il peut donc apparaître à gauche d'un opérateur d'affectation comme dans :

```
t[2] = 5
```

- Il peut aussi apparaître comme opérande d'un opérateur d'incrémentation, comme dans :

```
t[3]++
--t[i]
```

Attention

Il n'est pas possible, si t1 et t2 sont des tableaux d'entiers, d'écrire t1=t2 ; Le langage C n'offre aucune possibilité d'affectations globales de tableaux, comme c'était le cas, par exemple, en Pascal.

Indices

- Un indice peut prendre la forme de n'importe quelle expression arithmétique de type entier (ou caractère, compte tenu des règles de conversion systémique). Par exemple, si n,p,k et j sont de type int, ces notations sont correctes :

```
t[n-3]
t[3*p-2*k+j%1]
```

- Il en va de même, si c1 et c2 sont de type char, de :

```
t[c1+3]
t[c2-c1]
```

Dimension d'un tableau (1/2)


Le nombre d'éléments d'un tableau ne peut être qu'une constante ou une expression constante. Cette construction est correcte :

```
#define N 50
    .....
int t[N] ;
float h[2*N-1] ;
```


Mais elle ne le serait pas si N était une constante symbolique définie par :

```
const int N=50
```

Dimension d'un tableau (2/2)

? Lequel de ces exemples est juste ? C'est quoi l'"erreur" du mauvais exemple ?

- Exemple 1 :

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

- Exemple 2 :

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

Déclaration

- En C, on peut déclarer des tableaux à 2 indices (on dit aussi à plusieurs dimensions) :

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

- réserve un tableau de 15 (5 X 3) éléments. Un élément quelquonque de ce tableau se trouve alors repéré par deux indices comme dans ces notations :

```
tab[5][3]
tab[i][j]
tab[i-3][i+j]
```

Initialisation

- Voyez ces deux exemples équivalents (nous avons volontairement choisi des valeurs consécutives pour qu'il soit plus facile de comparer les deux formulations) :

```
int tab[3][4] = { {1,2,3,4},
                  {5,6,7,8},
                  {9,10,11,12} } ;
int tab[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12} ;
```

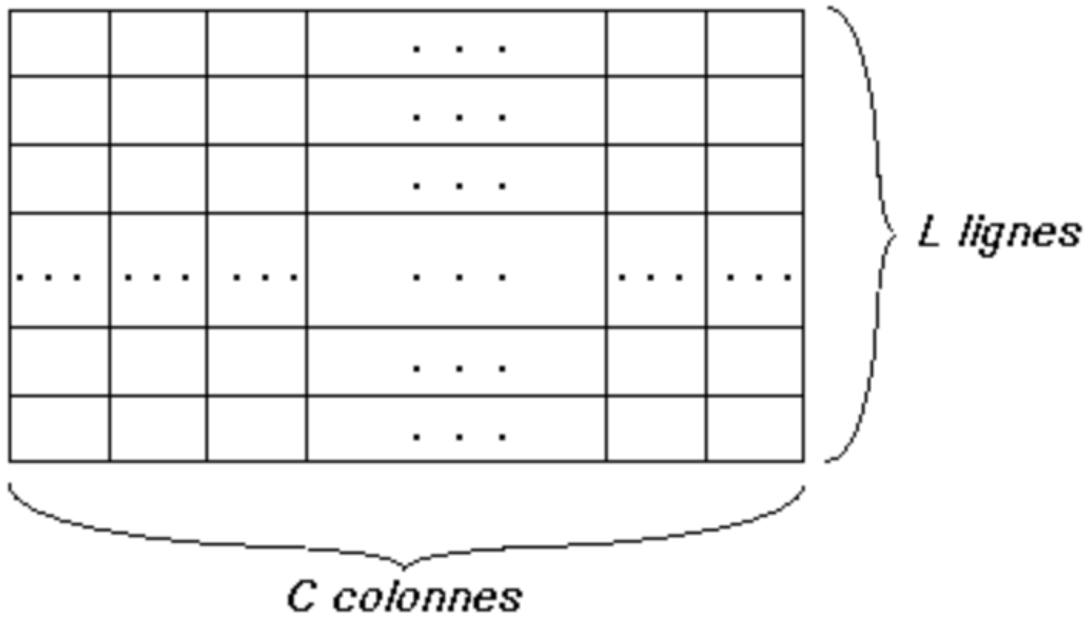
Comme les tableaux 1D

Bien entendu, les différents points évoqués, dans les précédentes diapositives, à propos des tableaux à une dimension, restent valables dans le cas des tableaux à plusieurs dimensions, avec quelques changements, comme l'initialisation dans le programme est avec deux boucles for pour les tableaux à deux dimensions ...

Arrangement en mémoire (1/3)

Les éléments d'un tableau sont rangés suivant l'ordre obtenu en faisant varier le dernier indice en premier. Ainsi le tableau tab[5][3] ses éléments ordonnés comme suit :

```
tab[0][0]
tab[0][1]
tab[0][2]
tab[1][0]
tab[1][1]
tab[1][2]
....
tab[4][0]
tab[4][1]
tab[4][2]
```



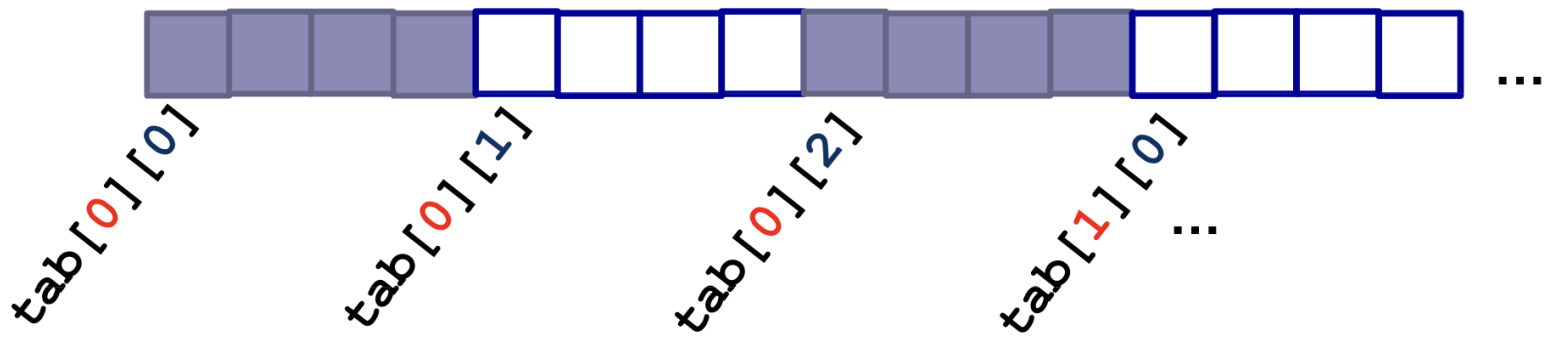
Arrangement en mémoire (2/3)

Nous verrons que cet ordre a une incidence dans au moins trois circonstances :

- lorsque l'on omet de préciser certaines dimensions d'un tableau,
- lorsque l'on souhaite accèder à l'aide d'un pointeur aux différents éléments d'un tableau,
- lorsque l'un des indices "déborde". Suivant l'indice concerné et les valeurs qu'il prend, il peut y avoir débordement d'indice sans sortie du tableau.

Arrangement en mémoire (3/3)

La mémoire du PC a une structure 1D et non 2D En mémoire, on aura pour int tab[5][3] ;



Un int occupant 4 octets donc les adresses évoluent de 4 en 4.

Autre notation

- Le symbole & pour signifier « adresse de ... »

```
Moyenne (tab, DIM) ;
Moyenne (&tab[0], DIM) ;

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)

```
Moyenne (tab, DIM) ;
int a = 8 ;
int * b = &a ;
```

Exercise 1

Quel est l'affichage du programme suivant ?

```
{
int mat[2][2],i,j;
   for (i = 0; i < 2; i++){
      for (j = 0; j < 2; j++){
         mat[i][j] = 0;
      }
   }
}
```

Exercise 2

Écrivez un code qui permet d'échanger le numéro de ligne avec le numéro de la colonne. En plus clair, les lignes deviennent les colonnes et inversement.

à l'affichage on obtient :

```
01234
56789
```

```
05
16
27
38
49
```