Université de Nice-Sophia Antipolis
Polytech/Xidian

Vendredi 6 novembre 2015
Durée: 1h30

tableaux
Travaux Pratiques – Séance n° 6b

---

1 Les tableaux à plusieurs dimensions

En C, mais aussi dans la plupart des langages de programmation, les tableaux à plusieurs dimensions sont des tableaux de tableaux, c'est-à-dire que ce sont des tableaux dont les éléments sont eux-mêmes des tableaux, dont les éléments peuvent être à leur tour des tableaux, *etc.*

1.1 Déclaration

Ci-dessous des exemples de déclaration de tableaux à plusieurs dimensions :

```
int t1[6][10];
char t2[3][4][2];
```

La déclaration de `t1` définit un tableau à 2 dimensions, *i.e.* un tableau à 6 éléments de type tableau à 10 éléments de type **int**. En fait, ce tableau représente une matrice à 6 lignes et 10 colonnes d'entiers.

Celle de `t2` définit un tableau à 3 dimensions, *i.e.* un tableau à 3 éléments de type tableau à 4 éléments de type tableau à 2 éléments de type **char**.

Notez que le nombre de dimensions n'est pas limité.

1.2 Accéder aux éléments

L'accès à un élément d'un tableau se fait par l'intermédiaire du nom du tableau et d'*indices*, un par dimension. La notation `t1[i][j]` désigne l'élément d'indice `ij` du tableau `t1`. La notation `t1[i][j][k]` désigne l'élément d'indice `ijk` du tableau `t2`.

Les indices pour chacune des dimensions est une expression entière à valeurs prises sur l'intervalle $0..n-1$ où n est le nombre d'éléments de la dimension.

1.3 Manipuler les éléments

Comme pour un élément de tableau à une dimension, un élément de tableau à plusieurs dimensions peut être manipulé exactement comme une variable. On peut donc effectuer des opérations avec des éléments de tableau comme avec n'importe quelles variables simples.

Le fragment de code suivant initialise à 1 tous les éléments d'une matrice $m \times n$, c'est-à-dire à m lignes et n colonnes.

```
#define M 5     /* nombre de lignes */
#define N 15    /* nombre de colonnes */

int m[M][N];
int i, j;
for (i=0; i<M; i++)
   for (j=0; j<N; j++)
       m[i][j] = 1;
```

2 Paramètres tableaux d'une fonction

Dans le cas d'un tableau à plusieurs dimensions, seul le nombre d'éléments de la première dimension peut être omis [1]. Par exemple, on écrira l'initialisation d'une matrice comme suit :

```
#define M 5
#define N 15

/* Rôle : initialise les éléments de mat tels que mat[i][j] = i*j */
/* Antécédent : nbLg nombre de lignes de la matrice mat */
void initTab(int mat[][N], int nbLg) {
   for (i=0; i<nbLg; i++)
     for (j=0; j<N; j++)
         mat[i][j] = i*j;
}
```

---

3 Exercices

1) Écrivez et testez deux fonctions. La première, `identite`, initialise une matrice $n \times n$ à la matrice identité. La seconde, `ecrireMat`, écrit sur la sortie standard une matrice $n \times n$. Le programme a la forme suivante :

```
#define N 10

int main(void) {
  int mat[N][N];
  identite(mat, N);
  ecrireMatrice(mat, N);
  return EXIT_SUCCESS;
}
```

1. Nous verrons également plus tard comment ne faire apparaître aucun nombre d'éléments.

2) Écrivez et testez une fonction qui retourne initialise la matrice d'entiers précédente de façon aléatoire.

3) Écrivez et testez une fonction qui retourne le minimum de la matrice d'entiers précédente.

---

4) Écrivez la procédure *triangleDePascal* qui écrit sur la sortie standard, sous forme d'un triangle à $n+1$ lignes, les coefficients binomiaux tels que la ligne $l \in [0; n]$ donne tous les coefficients binomiaux $\binom{n}{k}, \forall k \in [0; n]$. Par exemple, pour $n = 8$, on obtient :

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5  10  10   5   1
1   6  15  20  15   6   1
1   7  21  35  35  21   7   1
1   8  28  56  70  56  28   8   1
```

Notez qu'il est inutile de calculer les factorielles. On tient compte du fait que $\forall i, j$ tels $0 < j < i$, on a $\binom{i}{j} = \binom{i-1}{j-1} + \binom{i-1}{j}$. Les valeurs des coefficients binomiaux seront conservés dans une matrice, avec $m(0)(0) = 1$.

5) Modifiez votre programme, et utilisez la bibliothèque ncurses, pour faire apparaître *progressivement* les coefficients du triangle de Pascal.

---

6) On appelle *carré magique* une matrice carrée d'ordre n, contenant les entiers compris entre 1 et n^2, telle que les sommes des entiers de chaque ligne, de chaque colonne et des deux diagonales sont identiques. Vous allez voir maintenant comment produire des carrés magiques d'ordre impair.

Exemples : carrés magiques d'ordre 3 et 5

4	9	2
3	5	7
8	1	6

11	24	7	20	3
4	12	25	8	16
17	5	13	21	9
10	18	1	14	22
23	6	19	2	15

L'algorithme pour calculer un carré magique d'ordre impair est le suivant :

— On place 1 dans la case située une ligne en dessous de la case centrale (n est impair).
— Lorsqu'on a placé l'entier x dans la case (i, j), on place $x + 1$ dans la case $(i + 1, j + 1)$, mais ...
  — si un indice devient égal à n, il revient à 0, et
  — si on tombe sur une case déjà occupée, disons (l, k), on essaie de placer le nombre en $(l + 1, k - 1)$ ou si k était égal à 0, on prend $n - 1$ (et non -1) !

Notez que cet algorithme suppose que le carré est représenté par un tableau C à deux dimensions et donc que les indices commencent à 0. Le carré magique est représenté par la déclaration suivante :

```
#define N   ??      /* taille du carré magique */

int CarreMagique[N][N];
```

Écrivez la procédure `initCarreMagique` qui initialise un carré magique à 0.

```
void initCarreMagique(int cmagique[][N])
/* Rôle: Initialise un carré magique de dimension N à 0 */
```

7) Écrivez la procédure `afficherCarreMagique` qui affiche sur la sortie standard un carré magique. Vous afficherez le carré ligne par ligne, et chaque élément d'une ligne sera séparé par un espace.

```
void afficherCarreMagique(int cmagique[][N])
/* Rôle: Afficher un carré magique de dimension N sur la sortie standard */
```

8) Écrivez la procédure `calculerCarreMagique` qui calcule les valeurs du carré magique selon la méthode donnée précédemment.

```
void calculerCarreMagique(int cmagique[][N])
/* Rôle: calcule un carré magique de dimension N */
```

9) Écrivez la fonction `verifierCarre` qui vérifie si un carré est magique ou pas.

```
int verifierCarre(int cmagique[][N])
/* Rôle: vérifie si le carré magique de dimension N est magique */
```

Le corps du programme principal aura l'allure suivante :

```
int main(void) {
  int CarreMagique[N][N];
  if ((N & 1) == 0) {
    fprintf(stderr, "le carré doit être d'ordre impair\n");
    return 1;
  }
  calculerCarreMagique(CarreMagique);
  if (verifierCarre(CarreMagique))
    afficherCarreMagique(CarreMagique);
  return EXIT_SUCCESS;
}
```