TP: tableaux à deux dimensions en C

Compilation

Dans ce qui suit, on compile avec **gcc -Wall** mais sans **-Werror=vla**. Le compilateur demande d'indiquer les dimensions des tableaux passés en paramètres (sauf la première) :

```
$ gcc test_vla.c
test_vla.c:3:39: error: array type has incomplete element type ‘double[]‘
    3 | void affiche_mat(int n, int m, double mat [][]){
      |                                       ^~~
test_vla.c:3:39: note: declaration of ‘mat‘ as multidimensional array must have bounds for all dimensions except the first
```

On comprend donc qu'il faut que, lors du passage d'un tableau bi-dimensionnel en paramètre, le nombre de colonnes soit indiqué.

En choisissant maintenant le prototype `void affiche_mat(int n, int m, double mat [][m])`, aucune erreur ne s'affiche avec **gcc -Wall**.

Mais la compilation avec l'option **-Werror=vla** produit une erreur en contradiction avec la remarque sur le nombre de colonnes :

```
 gcc -Wall -Werror=vla test_vla.c
test_vla.c:3:1: error: ISO C90 forbids variable length array ‘mat‘ [-Werror=vla]
    3 | void affiche_mat(int n, int m, double mat [][m]){
      | ^~~~
cc1: some warnings being treated as errors
```

On peut préferer bien sûr déclencher un warning plutôt qu'une erreur :

```
$ gcc -Wall -Wvla test_vla.c
test_vla.c:3:1: warning: ISO C90 forbids variable length array ‘mat‘ [-Wvla]
    3 | void affiche_mat(int n, int m, double mat [][m]){
      | ^~~~
```

Le code compile et produit un exécutable. Mais comme nous sommes contraints d'indiquer les dimensions des tableaux multidimensionnels (sauf la première), le nombre de warning risque d'exploser. C'est pourquoi nous nous contentons de `gcc -Wall`

Matrices

Exercice 1. Ecrire une fonction `void affiche_mat(int n, int m, double mat[n][m])` qui affiche le contenu d'une matrice.

Avec

```
1  double mat [2][2]={{1.,2.,},{3.,4.}} ;
2  affiche_mat(2,2,mat);
```

on obtient l'affichage

```
|1.00 2.00|
|3.00 4.00|
```

Exercice 2. Ecrire une fonction `double det(double mat[2][2])` qui prend en paramètre une matrice 2×2 et renvoie son déterminant.

Exercice 3. Écrire une procédure qui prend en paramètres deux matrices de mêmes dimensions, leurs dimensions et une matrice résultat. La fonction met dans la matrice résultat la somme des deux autres.
`void add(int n, int m, double M1[n][m], double M2[n][m], double res[n][m])` .

Exercice 4. Si $A = (a_{ij})$ est une matrice de type (m, n) et $B = (b_{ij})$ est une matrice de type (n, p), alors leur produit, noté $AB = (c_{ij})$ est une matrice de type (m, p) donnée par :

$$\forall i, j : c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}.$$

Ecrire une procédure `void prod(int n, double M1[n][n], double M2[n][n],double res[n][n])` qui prend en paramètres deux matrices carrées de même dimension, leur dimension et met dans une matrice résultat le produit matriciel $M_1 \cdot M_2$.

```
mat1=
| 1.00 -1.00  2.00|
| 2.00  3.00  1.00|
|-2.00  1.00 -1.00|

mat2=
| 0.00  1.00 -3.00|
| 1.00 -1.00  1.00|
|-1.00  0.00  1.00|

prod(3,mat1,mat2,res); res=
|-3.00  2.00 -2.00|
| 2.00 -1.00 -2.00|
| 2.00 -3.00  6.00|
```

Exercice 5. Ecrire la procédure `void transpose(int n, int m, double M[n][m], double res[m][n])` qui met dans `res` la transposée de `M`.

```
mat1=
1.00 2.00 3.00
4.00 5.00 6.00
transpose(2,3,mat1,res); res=
1.00 4.00
2.00 5.00
3.00 6.00
```

Exercice 6. Si on place une matrice carrée M de taille $n \times n$ dans un repère orthonormé $(O, \vec{i}, \vec{j})$ en considérant que le coefficient $M_{i,j}$ est sur le point de coordonnées (i, j) alors la (sous-entendu *première*) diagonale de M coïncide avec la première bissectrice du repère.

Ainsi, la transposée de M peut-être vue comme le symétrique orthogonal de M par rapport à la première bissectrice du repère (cette symétrie échange juste les coordonnées).

Considérons maintenant la seconde diagonale de la matrice.

Ecrire la fonction `void anti_transpose(int n, double mat [n][n], double res [n][n])` qui met dans `res` la transposée de `M` par rapport à la seconde diagonale.

```
mat1=
| 1.00   2.00   3.00|
| 4.00   5.00   6.00|
| 7.00   8.00   9.00|

anti_transpose(3,mat1,res); res=
| 9.00   6.00   3.00|
| 8.00   5.00   2.00|
| 7.00   4.00   1.00|
```

Géométrie

Dans cet exercice, les points et les vecteurs sont représentés par des matrices colonnes de deux doubles. Par exemple $\begin{pmatrix} 2 \\ 3 \end{pmatrix}$ est représenté par `{{2},{3}}`. La base du repère est supposée orthonormée.

Pour ne pas écrire `int [2][1]` trop souvent (c'est long !) on définit un alias de type :

```
typedef double vect [2][1];
```

De la sorte, le code suivant est licite :

```
vect v = {{2},{1}};
affiche_mat(2,1,v);
```

On obtient l'affichage :

```
|2.00|
|1.00|
```

De même, on définit un type des matrices 2×2 par

```
typedef double matrix [2][2];
```

Exercice 7.

1. Ecrire la fonction `void vects2mat(vect v1, vect v2, matrix A);` qui permet de fusionner deux vecteurs en une matrice.
2. Ecrire la fonction `void solve(matrix A, vect b, vect x);` qui résoud l'équation matricielle $AX = B$ et met le résultat dans X.

Exercice 8.

1. Écrire la procédure

`void combinaison_lineaire(double a, double b, vect V1 , vect V2, vect res)`

qui prend en paramètres deux réels a, b (représentés par `a` et `b`), deux vecteurs $\overrightarrow{V}_1 \in \mathbb{R}^2$ et $\overrightarrow{V}_2 \in \mathbb{R}^2$ (représentés par `V1` et `V2`) et met dans `res` le résultat de la combinaison linéaire $a\overrightarrow{V}_1 + b\overrightarrow{V}_2$.

2. Ecrire une fonction `void rotateV(double theta, vect a, vect b);` qui met dans `b` le résultat de la rotation vectorielle d'angle `theta` appliquée au vecteur représenté par `a`.
3. Ecrire une fonction

   `void rotateA(double theta, vect C, vect a, vect b);`

   qui met dans `b` le résultat de la rotation affine d'angle `theta` et de centre représenté par `C` appliquée au point représenté par `a`.

Exercice 9.

1. Ecrire la fonction `bool appartient(vect A, vect B, vect C);` qui indique par un bouléen si le point C est sur la droite (AB).
2. Ecrire la fonction `double ps(vect U, vect V);` qui calcule le produit scalaire des vecteurs U, V.
3. Ecrire la fonction `void pied(vect A, vect B, vect C, vect D);` qui met les coordonnées du pied de la perpendiculaire à (AB) issue de C dans D.
4. Ecrire la fonction `void sym(vect A, vect B, vect C, vect D);` qui met les coordonnées du symétrique orthogonal de C par rapport à (AB) dans D.

Dans ces 3 exercices, on suppose que $A \neq B$ sans avoir besoin de le vérifier.