TP9 - Tableaux multidimensionnels

Langage C (LC4)

En C un tableau à deux dimensions de taille 3×4 peut être déclaré de la façon suivante :

```
int t[3][4];
```

Question 1. Déclarez un tableau d'entiers à deux dimensions de taille fixe, puis initialisez toutes ses valeurs par des entiers pris aléatoirement entre -5 et 5 (utilisez la fonction `rand`).

Question 2. Affichez l'adresse et le contenu de chaque case du tableau. Comment ce tableau est-il disposé en mémoire ?

1 Matrices

On utilisera la structure de donnée suivante pour représenter des matrices de taille arbitraire :

```
typedef struct {
  int lines, columns;
  int *data;
} matrix;
```

Question 3. Créez un fichier `matrix.c` ainsi qu'une interface `matrix.h` qui permettent de manipuler des matrices en fournissant les fonctions suivantes :

- `matrix *matrix_make(int lines, int columns)` qui crée une matrice de taille `lines` × `columns`.
- `void matrix_free(matrix *m)` qui libère l'espace alloué pour la matrice `m`.
- `int matrix_get(matrix *m, int i, int j)` qui retourne la valeur contenue dans la case d'indices donnés.
- `void matrix_set(matrix *m, int i, int j, int v)` qui écrit la valeur `v` dans la case d'indices donnés.
- `void matrix_random(matrix *m)` qui remplie `m` avec des valeurs aléatoires (comme dans la première question).
- `void matrix_print(matrix *m)` qui affiche la matrice sur `stdout`.

Question 4. Créez un fichier `main.c` pour tester toutes les fonctions définies dans `matrix.c`.

Question 5. Ajoutez dans `matrix.c` l'opération qui fait la somme de deux matrices :

- `matrix *sum(matrix *m1, matrix *m2)`

On vérifiera en faisant appel à la fonction `assert` que les deux matrices `m1` et `m2` ont les mêmes nombres de lignes et de colonnes.

On rappelle que, par exemple pour des matrices de taille 2×3 :

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{13}+b_{23} \end{pmatrix}$$

Question 6. Ajoutez dans `matrix.c` l'opération qui calcule l'opposée d'une matrice :
– `matrix *neg(matrix *m)`

Question 7. Ajoutez dans `matrix.c` l'opération qui calcule le produit de deux matrices :
– `matrix *product(matrix *m1, matrix *m2)`

On rappelle que la multiplication de deux matrices M et N est possible lorsque M est de taille $I \times K$, et N de taille $K \times J$. La valeur de la matrice produit MN est donnée par :

$$(MN)_{ij} = \sum_{k=1}^{K} M_{ik} N_{kj}$$

pour tout $i \in [1, I]$ et $j \in [1, J]$. Par exemple, lorsque $I = 4$, $K = 3$ et $J = 2$:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32} \\ a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31} & a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32} \end{pmatrix}$$

Attention : en C on utilisera on numérotera plutôt des indices à partir de 0.

2 Tableaux de lignes

On souhaite à présent classer des lignes et les trier en fonction de la valeur maximale qu'elle contiennent.

Question 8. Peut on utiliser la représentation mémoire précédente (soit avec taille fixe, soit avec allocation dynamique) ?

On définit la structure de donnée suivante :

```
typedef struct {
  int lines, columns;
  int **data;
} multiline;
```

Question 9. Pourquoi cette structure de donnée est-elle plus adaptée à notre nouveau problème ?

Question 10. Écrivez une fonction qui trie un `multiline` de la façon évoquée plus haut, ainsi qu'un exemple d'utilisation.