Cours de programmation séquentielle

Tableaux unidimensionnels

1 Les matrices

1.1 Buts

Utilisation de pointeurs sur des pointeurs en `C`.

1.2 Énoncé

Écrire des fichiers `matrix.h` et `matrix.c` pour manipuler des matrices, représentées par un type `matrix` de nombres à virgule flottante. Le contenu de la matrice devra être alloué dans une zone contiguë de la mémoire, tout en restant accessible par l'opérateur `[]`.

1.3 Cahier des charges

Pour manipuler des matrices, vous devrez implémenter les fonctions suivantes

1.3.1 Fonctions pour la création de nouvelles matrices et leur destruction

— création d'une nouvelle matrice de `m` lignes et `n` colonnes et allocation de la mémoire
```
matrix matrix_create(int m, int n);
```

— libération de la mémoire de la matrice en argument, le pointeur vers les données est mis à `NULL`, le nombre de lignes et de colonnes sont mis à `-1`
```
void matrix_destroy(matrix *mat);
```

— allocation d'une matrice de taille $m \times n$, et initialisation de ses valeurs à partir d'un tableau de taille `m * n`
```
matrix matrix_create_from_array(int m, int n, double data[]);
```

— création du clone de d'une matrice, la nouvelle matrice est une copie de la matrice d'origine
```
matrix matrix_clone(matrix mat);
```

— affichage d'une matrice (très utile pour le débogage)
```
void matrix_print(matrix mat);
```

— changement de la taille d'une matrice, si la nouvelle matrice est plus grande, les nouvelles cases sont initialisées à `0`, sinon les cases en trop disparaissent simplement (la fonction `realloc()` pourrait vous être utile); cette fonction retourne `1`, si tout s'est bien passé, `0` sinon

```
int matrix_resize(matrix *mat, int m, int n);
```

— égalité parfaite ou approximative de deux matrices

```
bool matrix_is_approx_equal(matrix mat1, matrix mat2, double epsilon);
bool matrix_is_equal(matrix mat1, matrix mat2);
```

1.3.2 Fonctions pour les manipulations de matrices en place

Les fonctions modifient les matrices passées en argument. A vous de déterminer les signatures des fonctions

— addition de deux matrices, la première matrice est modifiée par la fonction
`... matrix_add_in_place(...);`

— soustraction de deux matrices, la première matrice est modifiée par la fonction
`... matrix_sub_in_place(...);`

— multiplication de deux matrices, la première matrice est modifiée par la fonction
`... matrix_mult_in_place(...);`

— addition d'une matrice avec un scalaire, la première matrice est modifiée par la fonction
`... matrix_add_scalar_in_place(...);`

— multiplication d'une matrice avec un scalaire, la matrice est modifiée par la fonction
`... matrix_mult_scalar_in_place(...);`

— calcul de la transposée d'une matrice, la matrice est modifiée par la fonction
`... matrix_transpose_in_place(...);`

1.3.3 Fonctions pour les manipulations de matrices

Ces opérations nécessitent une nouvelle allocation (il faut réutiliser les fonctions définies en 1 et 2).

— addition de deux matrices, et stockage du résultat dans une troisième matrice
`... matrix_add(...);`

— soustraction de deux matrices, et stockage du résultat dans une troisième matrice
`... matrix_sub(...);`

— multiplication de deux matrices, et stockage du résultat dans une troisième matrice
`... matrix_mult(...);`

— addition d'un scalaire avec une matrice, et stockage du résultat dans une troisième matrice
`... matrix_add_scalar(...);`

— multiplication d'un scalaire avec une matrice, et stockage du résultat dans une autre matrice

```
... matrix_mult_scalar(...);
```

— calcul de la transposée d'une matrice, et stocke le résultat dans une nouvelle matrice
```
... matrix_transpose(...);
```

Le type matrice sera défini en `C` de la manière suivante

```
typedef struct matrix {
    int m, n;
    double** data;
} matrix;
```

Pour des raisons de performance il est important que les données soient allouées de façon contiguë en mémoire. Voici une illustration de la façon dont cela doit être réalisé avec une matrice où `m=5` et `n=4`. Il faut d'abord allouer la zone mémoire des éléments et le tableau de pointeurs data, puis faire pointer chaque pointeur de data au bon endroit dans la zone mémoire des éléments (c'est-à-dire sur chaque élément de début de ligne, voir fig. 1).

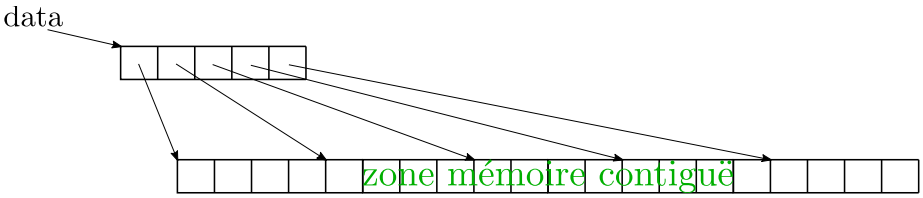

FIGURE 1: Les éléments de `data` (des pointeurs de pointeurs) pointent sur l'élément de début des lignes de la matrice.

Écrire un programme de `matrix_compute.c` utilisant vos fichiers de manipulation de matrices. Les matrices avec lesquelles on effectuera les opérations doivent être affichées à l'écran pour vérifier si les résultats sont corrects. Essayez également d'écrire des petits tests pour vérifier automatiquement si les résultats que vous obtenez sont corrects.

1.4 Indications

— Pensez à compiler souvent: le compilateur est votre ami.
— Évitez d'écrire toutes les fonctions en un fois sans tester si les fonctionnalités marchent comme vous le souhaitez.
— Pensez à utiliser les warnings et les sanitizers, cela peut vous sauver d'erreurs terribles et très difficiles à découvrir
```
-Wall -Wextra -pedantic -fsanitize=address
```
— Lorsque vous voyez apparaître des warnings corrigez-les immédiatement et pas "quand vous aurez fini". Il arrive régulièrement qu'un warning vous indique une erreur de logique dans votre code.

1.5 Rapide introduction/rappel aux matrices

Une matrice est un **tableau de nombres**, a un nombre de lignes noté, m, et un nombre de colonnes noté n. Pour simplifier, on dit que c'est une matrice $m \times n$.

La matrice $\underline{\underline{A}}$ ci-dessous, a 3 lignes et 4 colonnes

$$\underline{\underline{A}} = \begin{pmatrix} 2.2 & 1.3 & -1.2 & -2.2 \\ 3.0 & 1.2 & 1.3 & 3.3 \\ 1.0 & 4.0 & -1.3 & -1.0 \end{pmatrix}, \tag{1}$$

on dit donc que c'est une matrice 3×4.

Chaque élément d'une matrice peut être accédé par une paire d'indices, i, j (i étant le numéro de la ligne, j le numéro de la colonne), et est noté par A_{ij}. Dans le cas ci-dessus, l'élément $A_{14} = -2.2$.

Si on considère deux matrices, $\underline{\underline{A}}$, $\underline{\underline{B}}$ de tailles identiques, $m \times n$. Ces matrices peuvent s'additionner et se soustraire élément par élément. Dans le cas de l'addition (la soustraction se fait de façon similaire), on a

$$\underline{\underline{C}} = \underline{\underline{A}} + \underline{\underline{B}}, \tag{2}$$

où

$$C_{ij} = A_{ij} + B_{ij}, \quad 1 \leq i \leq m,\ 1 \leq j \leq n. \tag{3}$$

Exemple 1.1 *(Addition)*

Pour les matrices $\underline{\underline{A}}$, $\underline{\underline{B}}$

$$\underline{\underline{A}} = \begin{pmatrix} 2.2 & 1.3 & -1.2 & -2.2 \\ 3.0 & 1.2 & 1.3 & 3.3 \\ 1.0 & 4.0 & -1.3 & -1.0 \end{pmatrix} \text{ et } \underline{\underline{B}} = \begin{pmatrix} 1.2 & 2.3 & 3.2 & -1.2 \\ 2.0 & 0.2 & 2.3 & 3.0 \\ 1.0 & 3.2 & 1.3 & -1.0 \end{pmatrix}. \tag{4}$$

on aura comme résultat

$$\underline{\underline{A}} + \underline{\underline{B}} = \begin{pmatrix} 3.4 & 3.6 & 2.0 & -3.4 \\ 5.0 & 1.4 & 3.6 & 6.3 \\ 2.0 & 7.2 & 0.0 & -2.0 \end{pmatrix}. \tag{5}$$

De façon similaire, on peut définir multiplication (ou l'addition) par un scalaire, α

$$\underline{\underline{B}} = \alpha \cdot \underline{\underline{A}}, \tag{6}$$

où

$$B_{ij} = \alpha \cdot A_{ij}, \quad 1 \leq i \leq m,\ 1 \leq j \leq n. \tag{7}$$

On peut procéder de façon similaire pour l'addition, où on multiplie tous les éléments de la matrice par α.

Exemple 1.2 *(Multiplication par un scalaire)*

Pour la matrice $\underline{\underline{A}}$

$$\underline{\underline{A}} = \begin{pmatrix} 2.2 & 1.3 & -1.2 & -2.2 \\ 3.0 & 1.2 & 1.3 & 3.3 \\ 1.0 & 4.0 & -1.3 & -1.0 \end{pmatrix}, \tag{8}$$

et $\alpha = 2$, on a

$$\underline{\underline{B}} = 2 \cdot \begin{pmatrix} 2.2 & 1.3 & -1.2 & -2.2 \\ 3.0 & 1.2 & 1.3 & 3.3 \\ 1.0 & 4.0 & -1.3 & -1.0 \end{pmatrix} = \begin{pmatrix} 4.4 & 2.6 & -2.4 & -4.4 \\ 6.0 & 2.4 & 2.6 & 6.6 \\ 2.0 & 8.0 & -2.6 & -2.0 \end{pmatrix} \tag{9}$$

Pour la multiplication de deux matrices, cela est un peu plus compliqué. Supposons que la matrice $\underline{\underline{A}}$ soit de taille $m \times l$, et la matrice $\underline{\underline{B}}$ de taille $l \times n$, la multiplication

$$\underline{\underline{C}} = \underline{\underline{A}} \cdot \underline{\underline{B}}, \tag{10}$$

se définit comme

$$C_{ij} = \sum_{k=1}^{l} A_{ik} B_{kj}, \quad 1 \leq i \leq m,\ 1 \leq j \leq n, \tag{11}$$

et la matrice $\underline{\underline{C}}$ est de taille $m \times n$.

Exemple 1.3 *(Multiplication)*

Pour les matrices

$$\underline{\underline{A}} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad \underline{\underline{B}} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \tag{12}$$

On a

$$\underline{\underline{A}} \cdot \underline{\underline{B}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{13}$$

Finalement, on définit également la matrice *transposée* de la matrice $\underline{\underline{A}}$, notée $\underline{\underline{A}}^{\mathrm{T}}$, comme la matrice obtenue en inversant tous les indices de $\underline{\underline{A}}$. On a que $A^{\mathrm{T}}_{ij} = A_{ji}$. Si $\underline{\underline{A}}$ est une matrice $m \times n$, alors $\underline{\underline{A}}^{\mathrm{T}}$ est une matrice de taille $n \times m$.

Exemple 1.4 *(Multiplication par un scalaire)*

Pour la matrice

$$\underline{\underline{A}} = \begin{pmatrix} 2.2 & 1.3 & -1.2 & -2.2 \\ 3.0 & 1.2 & 1.3 & 3.3 \\ 1.0 & 4.0 & -1.3 & -1.0 \end{pmatrix}, \tag{14}$$

la matrice transposée $\underline{\underline{A}}^{\mathrm{T}}$ sera

$$\underline{\underline{A}}^{\mathrm{T}} = \begin{pmatrix} 2.2 & 3.0 & 1.0 \\ 1.3 & 1.2 & 4.0 \\ -1.2 & 1.3 & -1.3 \\ -2.2 & 3.3 & -1.0 \end{pmatrix}. \tag{15}$$

Finalement, pour que deux matrices soient égales, il faut que tous leurs éléments soient égaux et que leurs tailles soient les mêmes évidemment

$$\underline{\underline{A}} = \underline{\underline{B}} \Leftrightarrow A_{ij} = B_{ij}, \quad 1 \leq i \leq m,\ 1 \leq j \leq n. \tag{16}$$