

# Programmation séquentielle

## Série 9 - Tableau 2D dynamique

29.11.2022

### Les matrices

#### Buts

- Utilisation de pointeurs sur des pointeurs en C.
- Utilisation de `git`.
- Utilisation de fonctions.
- Utilisation de `make`.
- Utilisation de types énumérés.

#### Énoncé

Ce travail servira de base pour un prochain TP noté. Veuillez donc à le commencer suffisamment tôt et à avoir un code original (créé par vous).

Créer une librairie de manipulation de matrices (qui ne sont rien d'autre que des tableaux de tableaux, ou des tableaux à deux dimensions) dans les fichiers `matrix.h` et `matrix.c`. Vous devez créer un type `struct` nommé `matrix` représentant une matrice de nombres entiers (`int`). Ce type devra contenir les dimensions de la matrice, ainsi que les données contenues dans la matrice. Les données seront représentées sous la forme d'un tableau de tableaux, elles seront donc de type `int **`. Il faut également créer un exécutable qui vous permettra de tester que vos fonctions marchent comme vous le désirez.

#### Cahier des charges

Pour ce travail, en plus de la réalisation de la librairie de matrices, vous devez utiliser le logiciel de gestion de version `git`.

#### Gestion de versions

Pour tout ce travail pratique, vous devez utiliser `git` pour gérer les versions de votre programme. Vous devez utiliser votre compte sur <https://gitedu.hesge.ch>. Ensuite:

1. Créez le dépôt `matrix`.
2. Clonez les dépôt localement avec la commande `git clone`.
3. Créez un fichier `.gitignore` contenant au moins la ligne suivante `*.o`. Cela permet d'ignorer tous les fichiers `.o` que vous générerez à la compilation. Ajoutez ce fichier au dépôt avec la commande `git add .gitignore` et "commitez" le résultat avec la commande `git commit -m "ajout du .gitignore"`.<sup>1</sup>
4. Finalement, ajoutez entre autre les fichiers `matrix.h` et `matrix.c` ainsi que votre `Makefile`, puis commitez le résultat. **N'oubliez pas de compiler régulièrement votre projet et de faire des commits réguliers également.**

---

<sup>1</sup>Typiquement dans un fichier `.gitignore` on ajoute tous les fichiers binaires du dépôt pour éviter de les ajouter par erreur et de les versionner.

## Les matrices

Presque toutes les fonctions que vous allez écrire peuvent échouer. Il est donc obligatoire de créer un type énuméré contenant un code d'erreur que retourneront ces fonctions

```
typedef enum _error_code {  
    ok, err  
} error_code;
```

Pour manipuler des matrices, vous devrez implémenter les fonctions suivantes:

- création d'une nouvelle matrice de `m` lignes et `n` colonnes et allocation de la mémoire  
`error_code matrix_alloc(matrix *mat, int m, int n);`
- allocation et initialisation à une valeur, `val`, d'une nouvelle matrice de `m` lignes et `n` colonnes  
`error_code matrix_init(matrix *mat, int m, int n, int val);`
- 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`  
`error_code matrix_destroy(matrix *mat);`
- allocation d'une matrice, et initialisation de ses valeurs à partir d'un tableau de taille `m*n`  
`error_code matrix_init_from_array(matrix *mat, int m, int n,  
 int* data);`
- création du clone d'une matrice, la nouvelle matrice est une copie de la matrice d'origine (il faut réallouer la mémoire)  
`error_code matrix_clone(matrix *cloned, matrix mat);`
- création de la matrice transposée d'une matrice, la nouvelle matrice est une copie de la matrice originale ou les lignes et les colonnes sont échangées  
`error_code matrix_transpose(matrix *transposed,  
 matrix mat);`
- affichage d'une matrice (très utile pour le débogage)  
`error_code matrix_print(matrix mat);`
- test de l'égalité de deux matrices  
`bool matrix_is_equal(matrix mat1, matrix mat2);`
- récupération de l'élément `[ix][iy]` de la matrice de façon sûre (vérification des dépassements de capacité par exemple) et copie dans `elem`  
`error_code matrix_get(int *elem, matrix mat,  
 int ix, int iy);`
- modification d'un élément `[ix][iy]` de la matrice de façon sûre (vérification des dépassements de capacité par exemple)  
`error_code matrix_set(matrix mat, int ix, int iy,  
 int32_t elem);`

Le type matrice sera défini en C de la manière suivante

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

Vous devez allouer `data` comme étant un tableau de `m` pointeurs d'entier. Ensuite, la première case de `m` sera initialisé comme étant un tableau de `m*n` entiers. Puis, chaque case de `data` pointera successivement sur un élément décalé de `n` dans ce tableau d'entier. De cette façon, toutes les données de la matrice seront contigues en mémoires et accessibles avec la syntaxe `mat[i][j]`.

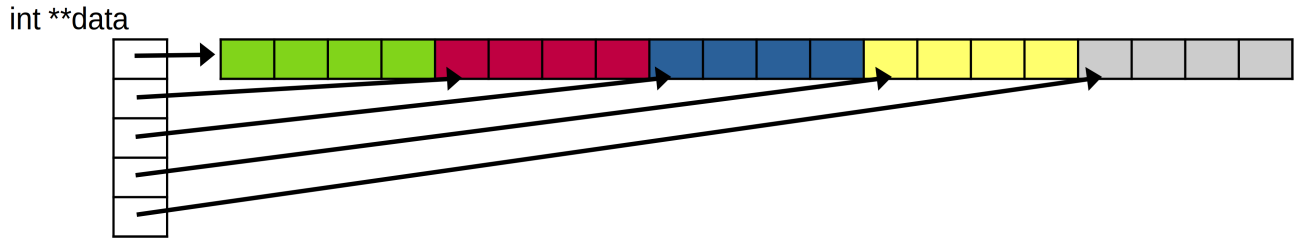


Figure 1: Exemple de la structure de donnée pour une matrice 5x4.

Écrivez un programme de test `matrix_compute.c` qui utilise votre librairie permettant de vérifier (en affichant les résultats à l'écran) que votre programme marche. Votre `Makefile` doit contenir une cible capable de compiler la librairie et le programme de test.

## Indications

- Pensez à compiler souvent: le compilateur est votre ami.
- Évitez d'écrire toutes les fonctions en une 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 -fsanitize=leak
- 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.

## 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

$$\underline{\underline{A}} = \begin{pmatrix} 2 & 1 & -1 & -2 \\ 3 & 1 & 1 & 3 \\ 1 & 4 & -1 & -1 \end{pmatrix}, \quad (1)$$

on dit donc que c'est une matrice  $3 \times 4$ .

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  $A_{14} = -2$ .

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

### Exemple (Transposée)

Pour la matrice

$$\underline{\underline{A}} = \begin{pmatrix} 2 & 1 & -1 & -2 \\ 3 & 1 & 1 & 3 \\ 1 & 4 & -1 & -1 \end{pmatrix}, \quad (2)$$

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

$$\underline{\underline{A}}^T = \begin{pmatrix} 2 & 3 & 1 \\ 1 & 1 & 4 \\ -1 & 1 & -1 \\ -2 & 3 & -1 \end{pmatrix}. \quad (3)$$

---

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