

Cours de programmation séquentielle

Matrices et outil de gestion d'images

Les matrices

Buts

- Utilisation de pointeurs sur des pointeurs en C.
- Utilisation de `git`.
- Utilisation de fonctions.
- Utilisation de `make`.
- Utilisation de types énumérés.
- Lecture et écriture de fichiers

Énoncé

Créer une librairie de manipulation de matrices (qui ne sont rien d'autre que des tableaux de tableaux) 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 **`. Une fois vos fonctions de manipulation de matrices fonctionnelles, vous les utiliserez pour implémenter un outil de traitement d'images simple.

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 `traitement_images`.
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"`.

4. Finalement, ajoutez entre autres les fichiers `matrix.h` et `matrix.c`, ainsi que votre `Makefile`, puis committez le résultat. **N'oubliez pas de compiler régulièrement votre projet et de faire des commits réguliers également.**

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 au minimum 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 `s = 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);
```
- affichage d'une matrice (très utile pour le débogage)

```
error_code matrix_print(matrix mat);
```
- 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,  
                     int elem);
```

Le type matrice sera défini en C de la manière suivante

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

Bien que cela ne soit pas optimal d'un point de vue de la performance, vous devez allouer `data` comme étant d'abord un tableau de `m` pointeur d'entier, puis chaque case de `data`, contiendra un tableau de `n` entiers.

Traitement d'images

Créer une librairie de manipulation d'images en *niveaux de gris*. Cette librairie devra permettre d'effectuer les opérations suivantes :

1. Lecture d'une image au format PGM.
2. Sauvegarde d'une image dans le format PGM.
3. Effectuer des opérations sur les images :
 1. Symétrie *horizontale* d'une image.
 2. *Rogner* une image.
 3. Calculer le *négatif* d'une image.

Représentation informatique des images

Dans ce travail pratique, une image n'est rien d'autre qu'une matrice de nombres entiers, au sens de celles que vous venez d'implémenter. Pour simplifier les images seront uniquement représentées en *niveaux de gris*. Ainsi chaque élément de la matrice représente un pixel. Les valeurs des pixels devront être limitées entre 0 et une valeur maximale `max`.

Pour simplifier encore, les images seront supposées être données au format PGM. Le format PGM (portable graymap file format) est un format de fichier très simple permettant de stocker des images en niveau de gris. Nous utiliserons le format PGM **textuel** pour commencer. Vous trouverez une définition du format sur la page https://fr.wikipedia.org/wiki/Portable_pixmap.

Une image PGM peut être représentée à l'aide de la structure de données suivante :

```
typedef struct _pgm {  
    int max;  
    matrix pixels;  
} pgm;
```

Le format PGM implique textuel la lecture dans un premier temps d'une entête contenant le texte `P2` sur la première ligne, puis les dimensions de l'image sur la deuxième ligne. Sur la troisième ligne se trouve le niveau de gris maximal. Ces trois lignes sont en format `ASCII`. Finalement, les pixels de l'image sont stockés dans les lignes restantes.

Nous supposons dans ce travail, pour simplifier, qu'il n'existe pas de commentaires dans les fichiers (pas de lignes commençant par `#`).

Lecture et écriture d'une image PGM

Afin de lire et écrire un fichier, il faut utiliser le type pointeur de fichier :

```
FILE *f;
```

et les fonctions permettant de manipuler les fichiers :

- `fopen()` avec les option `r` et `w` pour ouvrir un fichier en mode lecture ou écriture.
- `fclose()` pour fermer un fichier.
- `fprintf()` pour écrire du texte formaté.
- `fgets()` pour lire une ligne d'un fichier au format ASCII.
- `fwrite()` et `fread()` pour lire et écrire des données contiguës en mémoire dans un fichier (si vous souhaitez gérer le format binaire).

A l'aide de ces fonctions vous devriez être capables de lire et écrire des fichiers PGM.

Afin de tester vos fonctions vous pouvez utiliser l'image du mandrill.

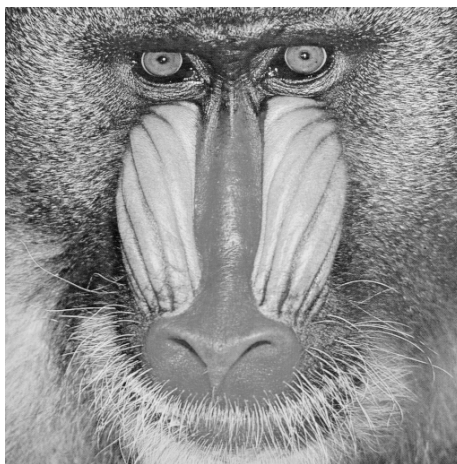


FIG. 1 : Image d'un mandrill.

Fonctions à implémenter

Il faut implémenter **au minimum** les fonctions suivantes :

- la fonction

```
error_code pgm_read_from_file(pgm *p, char *filename);
```

lisant le fichier `filename` et écrivant son contenu dans la variable `p`. Cette fonction retourne un `error_code`.

- la fonction

```
error_code pgm_write_to_file(pgm *p, char *filename);
```

écrivant le contenu de l'image `p` dans le fichier `filename`. Cette fonction retourne un `error_code`.

Les transformations d'images

Le négatif

Le négatif d'une image consiste à *inverser* la valeur des pixels de l'image par rapport à la valeur maximale permise. Ainsi si on représente **max** niveaux de gris, le négatif d'un pixel de niveau de gris, **p**, est donné par **max-p**. Un exemple de négatif de l'image du mandrill se trouve sur la figure suivante.

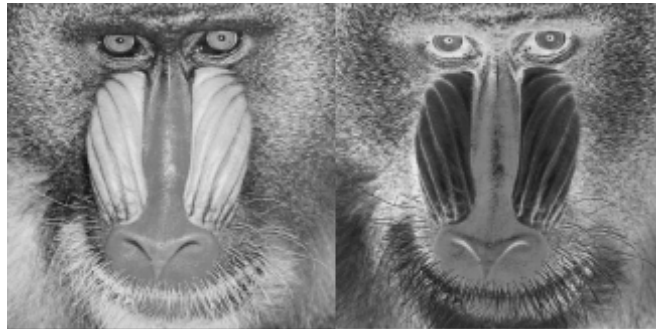


FIG. 2 : L'original (image de gauche) et le négatif (image de droite) de la photo du mandrill.

Fonctions à implémenter

Il faut implémenter **au minimum** la fonction suivante :

- la fonction

```
error_code pgm_negative(pgm *neg, pgm *orig);
```

calculant le négatif de l'image **orig** et la stockant dans **neg** (qui est également allouée dans cette fonction). L'image **orig** n'est pas modifiée. Cette fonction retourne un **error_code**.

Les symétries

Une symétrie verticale ou horizontale consiste à inverser l'ordre des pixels verticalement ou horizontalement respectivement. La symétrie centrale consiste à échanger les lignes et les colonnes d'une image. Vous pouvez voir un exemple de ces trois symétries sur les figures suivantes.

Fonctions à implémenter

Il faut implémenter **au minimum** la fonction suivante :

- la fonction

```
error_code pgm_symmetry_hori(pgm *sym, pgm *orig);
```

calculant la symétrie horizontale de l'image **orig** et la stockant dans **sym** (qui est également allouée dans cette fonction). L'image **orig** n'est pas modifiée. Cette fonction retourne un **error_code**.

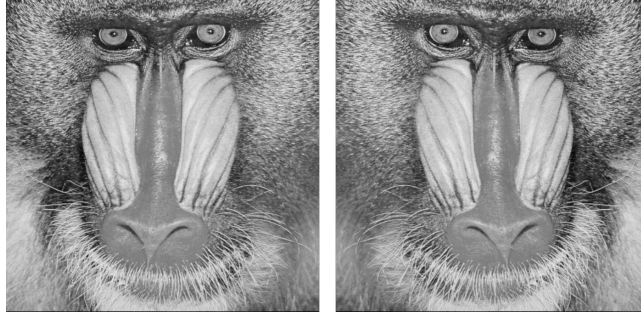


FIG. 3 : Exemple de symétrie horizontale.

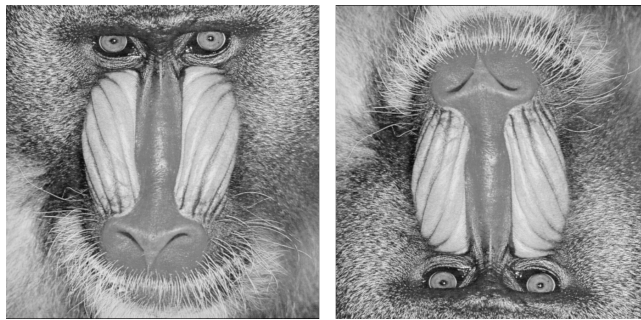


FIG. 4 : Exemple de symétrie verticale.

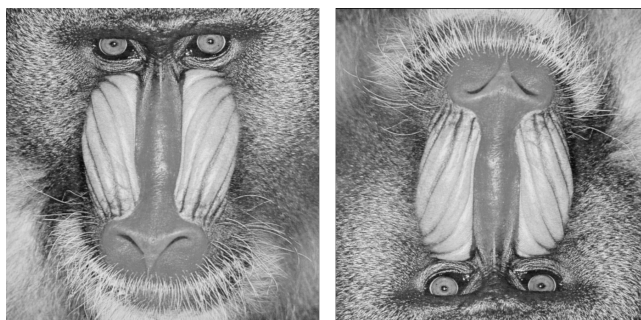


FIG. 5 : Exemple de symétrie centrale.

Rogner

Le rognage d'une image est une opération assez simple. Elle consiste à extraire une sous partie rectangulaire des pixels de l'image d'origine. Un exemple peut se trouver sur la figure suivante.

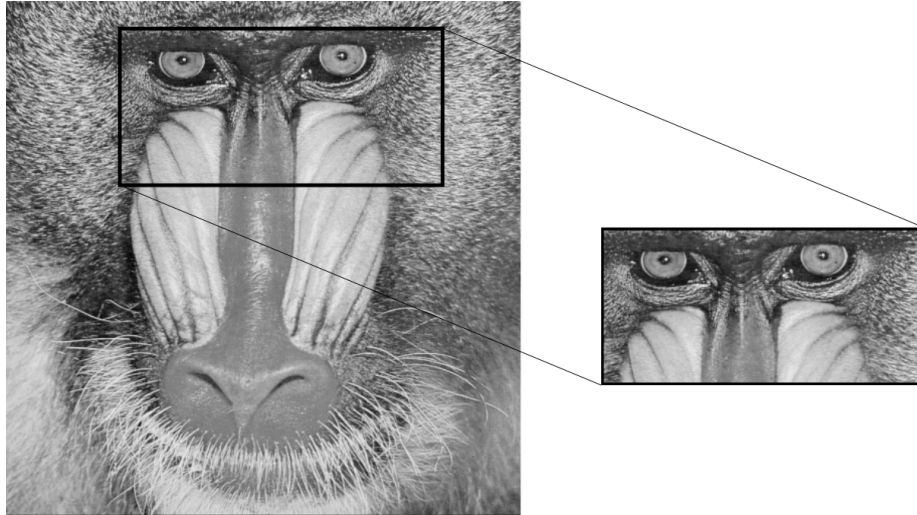


FIG. 6 : Rognage de la photo de mandrill.

Fonctions à implémenter

Il faut implémenter **au minimum** la fonction suivante :

- la fonction

```
error_code pgm_crop(pgm *crop, pgm *orig,  
                    int32_t x0, int32_t x1,  
                    int32_t y0, int32_t y1);
```

calculant la sous-image de `orig` aux coordonnées `x0` à `x1` (non-inclu), et `y0` à `y1` (non-inclu). Le résultat est stocké dans `crop` (qui est également allouée dans cette fonction). L'image `orig` n'est pas modifiée. Cette fonction retourne un `error_code`.

Programme principal

Une fois que toutes les fonctions demandées sont implémentées, créez un exécutable qui :

- Demande à l'utilisateur quel fichier PGM lire,
- quel traitement appliquer (symétrie, rognage, négatif),
- dans quel fichier stocker le résultat,

et effectue l'opération demandée.