SÉANCE 8

Objectif

Le but de cette séance est de mettre en œuvre les techniques de programmation modulaire.

Exercices

Exercice 1 (Module de manipulation de matrices dynamiques)

Il s'agit de reprendre les fonctions que vous avez faites lors de la séance 6 afin d'en faire un module en suivant les consignes décrites dans la chapitre sur la programmation modulaire. Ce module sera constitué du fichier d'en-tête `matrice.h` et du fichier d'implémentation `matrice.c`.

1. Constituez le fichier d'en-tête `matrice.h` avec :
   - la définition du synonyme de type `tMatrice` permettant de manipuler les matrices ;
   - les déclarations des fonctions `MatAllouer`, `MatCopier`, `MatLiberer`, `MatLire` et `MatAfficher`, précédées du mot-clé `extern` ;
   - les directives permettant d'éviter les inclusions multiples.

   Constituez le fichier d'implémentation `matrice.c` avec :
   - les définitions des fonctions indiquées ci-dessus ; si vous avez utilisé des fonctions auxiliaires, faites précéder leur définition du mot-clé `static` ;
   - les inclusions qui conviennent.
2. Afin de tester le bon fonctionnement du module, copiez la fonction principale que vous avez faite lors de la séance 6 dans le fichier `tp8ex1.c`.
3. Écrivez le fichier de dépendances permettant de produire, avec l'utilitaire `make`, l'exécutable `tp8ex1`.

Exercice 2 (Module de manipulation d'images de niveaux de gris)

L'objectif de cet exercice est d'écrire un module de manipulation d'images de niveaux de gris qui utilise le module `matrice` de l'exercice précédent. Contrairement au module `matrice`, le module `image` va mettre en œuvre la technique d'encapsulation des données par la définition retardée d'une structure.

Vous devez écrire :

- le fichier d'en-tête `image.h` ;
- le fichier d'implémentation `image.c` ;
- le fichier `tp8ex2.c` qui va contenir la fonction principale permettant de tester le module ;
- le fichier de description des dépendances permettant de produire l'exécutable `tp8ex2`.

On choisit de représenter les pixels d'une image de niveaux de gris par une matrice dont les éléments sont de type `unsigned char`. Un niveau de gris est alors un entier appartenant à l'intervalle $[0, 255]$ où 0 représente le noir et 255 le blanc.

Le synonyme de type public permettant de manipuler les images est :

```
typedef struct sImage *tImage;
```

Le type privé d'implémentation correspondant est :

```
struct sImage
{
  int NbLig; // Nombre de lignes de l'image
  int NbCol; // Nombre de colonnes de l'image
  tMatrice NivGris; // Matrice des niveaux de gris de l'image
};
```

1. **Fonctions de base**
   Les fonctions publiques de base que vous devez programmer sont les suivantes :
   - `tImage ImAllouer(int NbLignes, int NbColonnes)`
     Cette fonction alloue et initialise l'espace mémoire nécessaire pour stocker une image contenant une matrice de `NbLignes` lignes et `NbColonnes` colonnes niveaux de gris. Vous n'initialiserez pas les valeurs des niveaux de gris de l'image.
   - `void ImLiberer(tImage *pIm)`
     Cette fonction libère tout l'espace mémoire occupé par l'image d'adresse `pIm` (la matrice des niveaux de gris et la structure `sImage`). Cette fonction doit aussi affecter la valeur `NULL` à l'image libérée (ceci permet d'éviter les erreurs à l'exécution si on essaie de libérer plusieurs fois la même image).
   - `int ImNbLig(tImage Im)`
     Cette fonction retourne le nombre de lignes de l'image `Im`.
   - `int ImNbCol(tImage Im)`
     Cette fonction retourne le nombre de colonnes de l'image `Im`.
   - `tMatrice ImNivGris(tImage Im)`
     Cette fonction retourne la matrice des niveaux de gris de l'image `Im`.
2. **Entrées-sorties**
   Il existe de nombreux formats de fichiers contenant des images. Nous allons ici considérer le format « PGM-ASCII ». Il s'agit de fichiers de texte contenant :
   - les deux caractères `P2` (c'est la « signature » du format PGM-ASCII) ;
   - un ou plusieurs séparateurs (espace, tabulations ou sauts de lignes) ;
   - le nombre de colonnes de l'image ;
   - un ou plusieurs séparateurs ;
   - le nombre de lignes de l'image ;
   - un ou plusieurs séparateurs ;
   - le niveau de gris maximal ;
   - un ou plusieurs séparateurs ;
   - les niveaux de gris de l'image séparés par un ou plusieurs séparateurs, disposés du coin en haut à gauche de l'image au coin en bas à droite, ligne par ligne (dans l'ordre de lecture d'un texte français).

   Dans ce format, les niveaux de gris sont des entiers dans l'intervalle $[0, M]$, où M est le niveau de gris maximal. Un niveau de gris égal à M représente le blanc. Pour respecter la représentation choisie, il est donc nécessaire de transformer chaque niveau de gris en le passant de l'intervalle $[0, M]$ à l'intervalle $[0, 255]$ en utilisant la transformation `round(255.0*val/M)`, où `val` est le niveau de gris initial.

   Rappels
   La fonction `fscanf` utilisée avec le format `%s` pour lire une chaîne de caractères s'arrête dès qu'elle rencontre un séparateur. Utilisée avec le format `%d` pour lire un `int` (pour le nombre de colonnes, le nombre de lignes, le plus grand niveau de gris et le niveau de gris de chaque pixel), la fonction `fscanf` ignore (« saute ») les éventuels séparateurs qui précèdent le nombre à lire.
   - Écrivez la fonction d'en-tête :
     `tImage ImLire(char NomFichier[])`
     qui lit l'image contenue dans le fichier de nom `NomFichier` au format PGM-ASCII et retourne cette image ou `NULL` en cas de problème.
   - Écrivez la fonction d'en-tête :
     `void ImEcrire(tImage Im, char NomFichier[])`
     qui écrit l'image `Im` dans le fichier de nom `NomFichier` au format PGM-ASCII. La valeur du

niveau de gris maximal à écrire est `255`.

3. **Test du module**
   - Dans le fichier `tp8ex2.c`, écrire une fonction principale qui :
     - lit une image contenue dans un fichier au format PGM-ASCII ;
     - crée une nouvelle image dont les niveaux de gris sont ceux de l'image initiale après avoir subi une rotation de 90° (voir l'exemple ci-dessous) ;
     - écrit cette image ayant subi la rotation dans un autre fichier au format PGM-ASCII.
   - Écrivez un fichier de description des dépendances permettant de produire le fichier exécutable `tp8ex2`.

   Exemple
   Si on effectue une rotation de 90° à partir du tableau :

1	2	3
4	5	6

   on obtient le tableau :

3	6
2	5
1	4

   Vous pouvez tester votre programme sur l'image contenue dans le fichier `feep.pgm` disponible sur Moodle. Cette image contient 7 lignes et 24 colonnes. Vous devez obtenir un fichier résultat contenant :

```
P2
7 24
255
  0   0   0   0   0   0   0
  0 255 255 255   0   0   0
  0 255   0 255   0   0   0
  0 255   0 255   0   0   0
  0 255 255 255 255 255   0
  0   0   0   0   0   0   0
  0   0   0   0   0   0   0
  0 187   0   0   0 187   0
  0 187   0 187   0 187   0
  0 187   0 187   0 187   0
  0 187 187 187 187 187   0
  0   0   0   0   0   0   0
  0   0   0   0   0   0   0
  0 119   0   0   0 119   0
  0 119   0 119   0 119   0
  0 119   0 119   0 119   0
  0 119 119 119 119 119   0
  0   0   0   0   0   0   0
  0   0   0   0   0   0   0
  0  51   0   0   0   0   0
  0  51   0  51   0   0   0
  0  51   0  51   0   0   0
  0  51  51  51  51  51   0
  0   0   0   0   0   0   0
```

Pour voir le contenu du fichier image initial, vous pouvez taper :

```
cat feep.pgm
```

Remarquez que le niveau de gris maximal dans ce fichier est `15`.

Pour visualiser le résultat sous forme d'image, si vous disposez de l'outil ImageMagick et si le résultat est dans le fichier `feep2.pgm`, alors il suffit de taper :

```
display feep2.pgm
```

qui affichera dans une petite fenêtre l'image obtenue (il suffit de taper sur la touche `q` pour quitter cet affichage).

Vous pouvez tester votre programme sur des images plus grandes en utilisant par exemple le fichier `chien.pgm` qui contient une image de 300 lignes et 400 colonnes.

Pour aller plus loin

Vous pouvez également utiliser l'image de votre choix. Pour cela, vous devez la convertir en image de niveaux de gris au format PGM-ASCII. Avec ImageMagick, pour convertir, par exemple, le fichier `mon_fichier_image.jpg` en `mon_fichier_image.pgm`, vous pouvez utiliser la commande suivante :

```
convert mon_fichier_image.jpg -compress none mon_fichier_image.pgm
```

Mais, attention, il est très probable qu'apparaissent des commentaires dans l'en-tête du fichier qui contient l'image. Il s'agit de lignes qui commencent par le caractère `#`. Vous devrez alors modifier votre programme pour qu'il ignore ces lignes.