U.E. S2IN324 - CTP - Devoir

Indications : *Ce devoir est à réaliser en binôme (ou individuellement) et à rendre exclusivement par dépôt sur le site Moodle du cours en trois étapes :*

- ***avant le 20 novembre 23h55** : partie 1, Représentation - Allocation - Chargement et Sauvegarde d'une image couleur. Fichier `tpimage.h` complet et fichier `tpimage.c` contenant le code des fonctions des exercices 1 à 6.*
- ***avant le 4 décembre 23h55** : partie 2, Transformation en niveau de gris, pixelisation, chargement et sauvegarde d'une image en niveau de gris. Fichier `tpimage.c` complété avec le code des fonctions des exercices 7 à 10 (y compris la fonction luminance).*
- ***avant le 18 décembre 23h55** : le devoir complet. Fichier `tpimage.c` complété avec le code des fonctions des exercices 11 et 12, et le fichier `main.c` contenant le programme principal.*

Vous indiquerez en commentaire au début de chacun des fichiers de code : numéros étudiant, noms et prénoms des protagoniste.

Travail à réaliser

Il s'agit de fournir un module `tpimage` (fichier en-tête et fichier source) permettant de manipuler des images au travers des quelques opérations demandées dans l'énoncé. Vous devrez aussi coder un programme principal qui proposera d'utiliser les différentes opérations du module.

Votre devoir terminé sera donc constitué de 3 fichiers : `tpimage.h`, `tpimage.c` et `main.c`.

Vous déposerez sur Moodle une archive contenant l'ensemble des fichiers de votre devoir. Si vous le faites à deux, vous ne devez rendre qu'un seul devoir déposé sur un seul de vos compte Moodle.

Introduction au traitement d'image

Ce devoir propose d'aborder le traitement d'image à travers quelque opérations à effectuer sur une image au travers d'une représentation en mémoire sous forme de tableau. Le but est d'écrire un module (`tpimage.h`) contenant ces différentes opérations de traitement d'images.

Dans la première partie vous allez voir comment lire et stocker une image couleur en mémoire et la sauver dans un fichier. Dans la deuxième partie vous programmerez les opérations permettant de transformer une image couleur en niveaux de gris, de pixeliser une image en niveaux de gris et de sauver une image en niveau de gris dans le bon format de fichier. La troisième partie concerne la création d'ascii art à partir d'une image en niveau de gris.

Les types d'images numériques

Il existe trois types d'images numériques : les images noir et blanc, les images en niveau de gris et les images couleurs.
Une image numérique et un tableau rectangulaire dont les éléments sont des pixels. Selon le type d'image un pixel est codé :

- pour les images en noir et blanc, sur 1 bit : 0 pour noir et 1 pour blanc
- pour les image en niveau de gris, sur 8 bits, soit une entier compris entre 0 (noir) et 255 (blanc)
- pour les image en couleur, par un triplet d'entiers codant les trois couleurs rouge, vert et bleu. Le niveau de chaque couleur est codée par un entier généralement compris entre 0 (absence de la couleur) et 255 (niveau maximum).

Formats de fichier graphique pour les échanges.

Le portable pixmap file format (PPM), le portable graymap file format (PGM) et le portable bitmap file format (PBM) sont des formats de fichier graphique permettant de représenter les trois types d'images numériques et utilisés pour les échanges. Ces fichiers sont composés sur la même base :

- le *nombre magique* du format écrit sur deux octets qui indique le type de format et la variante sa représentation binaire ou ASCII
  - extension de fichier `.pbm` pour les images en noir est blanc, type P1 (ASCII) et P4 (binaire)
  - extension `.pgm` pour les images en niveaux de gris, type P2 (ASCII) et P5 (binaire)
  - extrension `.ppm` pour les images couleurs, type P3 (ASCII) et P6 (binaire)
- un caractère d'espacement (espace, tabulation, passage à la ligne)
- suivent éventuellement quelques lignes de commentaires qui débutent par le caractère `#`
- la largeur de l'image en nombre de pixels, écrit explicitement sous forme d'un nombre en caractères ASCII
- un caractère d'espacement,
- la hauteur de l'image, un caractère d'espacement,
- **uniquement pour les fichiers en niveau de gris et en couleur** : la valeur maximale utilisée pour coder les niveaux de gris ou les niveaux de rouge, vert et bleu. Cette valeur maximale, la plupart du temps 255, doit être inférieur à 65535. Cette valeur est suivie d'un espacement.
- les données de l'image, c'est à dire les valeurs associées à chaque pixel, écrits ligne par ligne, depuis le premier situé en haut à gauche de l'image jusqu'au dernier situé en bas à droite

Dans ce devoir, vous manipulerez des images `.ppm` de type P3 et `.pgm` de type P2.

Exemple de fichier d'image couleur `.ppm` de type P3

Voici le début du fichier `.ppm` d'une image en couleur codée en ascii, visualisé avec un éditeur de texte. Cet extrait contient le type de l'image, une ligne de commentaire indiquant le logiciel ayant servi à créer l'image, une ligne contenant la largeur et la hauteur en nombre de pixels. La ligne suivante contient la valeur maximale des niveaux : 255. Les 6 entiers qui suivent représentent les niveaux de (rouge, vert, bleu) des deux premiers pixels de l'image : (105, 145, 204) et (113, 150, 227). Chaque pixel de l'image est donc codé par 3 valeurs donnant les intensités de rouge, de vert et de bleu.

```
P3
# CREATOR: GIMP PNM Filter Version 1.1
1600 1200
255
105
145
204
113
150
227
...
```

Exemple de fichier d'image en niveau de gris `.pgm` ascii de type P2

Voici le début du fichier `.pgm` d'une image en niveaux de gris codée en ascii, visualisé avec un éditeur de texte. Cet extrait contient le type de l'image, une ligne de commentaire indiquant le logiciel ayant servi à créer l'image, une ligne contenant la largeur et la hauteur en nombre de pixels. La ligne suivante contient la valeur maximale du niveau de gris : 255. Les 4 entiers qui suivent représentent les niveaux de gris des 4 premiers pixels de l'image soit 85, 86 90 et 87. Chaque pixel est codé par une seule valeur indiquant l'intensité de gris.

```
P2
# CREATOR: GIMP PNM Filter Version 1.1
640 960
255
85
86
90
87
...
```

1 Représentation en mémoire d'une image

On donne ci-dessous (c.f. le fichier `pourCCTP.txt` qui accompagne cet énoncé) la définition des types que vous utiliserez pour représenter les images en mémoire.

```
// ***********************************************
// Définition des types

// Le type tPixel pour représenter les niveaux de (rouge, vert, bleu)
typedef struct pixel {
    int r;
    int v;
    int b;
} tPixel;

// Le type tImCouleurP3 pour représenter une image en couleur PPM dans un tableau de pixels
typedef struct imagecouleur {
    int hauteur;        // Hauteur en pixels
    int largeur;        // Largeur en pixels
    int maxval;         // Valeur maximal des niveaux de rouge, vert, bleu d'un pixel
    tPixel** tabpix;       // Le tableau des pixels (tableau de structures de type tPixel)
} tImCouleurP3;
```

Le champ `tabpix` est un tableau rectangulaire de pixels, c'est à dire un tableau de tableaux de tPixels. Le premier indice fait référence aux lignes de l'images et le second indice aux colonnes. Il y a `hauteur` lignes de `largeur` pixels. Ainsi `tabpix[i][j]` contient le pixel situé en i^{eme} ligne et j^{eme} colonne (en comptant à partir de 0).

```
// Le type tImGrisP2 pour représenter une image PGM en mémoire
typedef struct imageGris {
    int hauteur;        // Hauteur en pixels
    int largeur;        // Largeur en pixels
    int maxval;         // Valeur maximal du niveau de gris des pixels
    int** tabgris;      // Tableau des niveaux de gris des pixels
} tImGrisP2;
```

Le champ `tabgris` est un tableau rectangulaire d'entier (puisque les pixels gris sont représenté uniquement pas un niveau de gris), c'est à dire un tableau de tableaux d'entiers. Le premier indice fait référence aux lignes de l'images et le second indice aux colonnes. Il y a `hauteur` lignes de `largeur` pixels. Ainsi `tabgris[i][j]` contient le niveau de gris du pixel situé en i^{eme} ligne et j^{eme} colonne (toujours en comptant à partir de 0).

1.1 Allocation de mémoire pour stocker une image

Exercice 1

Écrivez une fonction **tImCouleurP3 initImCouleur(int haut, int larg, int vmax)** qui retourne une structure `tImCouleurP3` dans laquelle les champs `hauteur` et `largeur` sont initialisés avec les valeurs données en paramètre et qui réalise l'allocation du tableau `tabpix` pour `haut` lignes de `larg` pixels en couleur. Le champs `maxval` sera initialisé à `vmax`.

Exercice 2 — Copie d'une image couleur

Ecrivez une fonction **tImCouleurP3 copieImCouleur(tImCouleurP3 im)** qui réalise la copie de l'image donnée en paramètre dans une nouvelle image initialisée et allouée avec les mêmes caractéristiques que l'image donnée en paramètre et dans laquelle tous les pixels sont copiés. La fonction retourne la copie de l'image.

Exercice 3

Écrivez une fonction **tImGrisP2 initImGris(int haut, int larg, int vmax)** qui retourne une structure `tImGrisP2` dans laquelle les champs `hauteur` et `largeur` sont initialisés avec les valeurs données en paramètre et qui réalise l'allocation du tableau `tabgris` pour `haut` lignes de `larg` pixels gris. Le champs `maxval` sera initialisé à `vmax`.

Exercice 4 — Copie d'une image en niveau de gris

Ecrivez une fonction **tImGrisP2 copieImGris(tImGrisP2 im)** qui réalise la copie de l'image donnée en paramètre dans une nouvelle image initialisée et allouée avec les mêmes caractéristiques que l'image donnée en paramètre et dans laquelle tous les pixels sont copiés. La fonction retourne la copie de l'image.

1.2 Chargement en mémoire d'une image couleur depuis un fichier `.ppm`

Le chargement d'une image en mémoire consiste à initialiser et remplir une structure de type `tImCouleurP3` à partir des informations contenu dans un fichier `.ppm`.

Exercice 5

A partir du squelette donné dans le fichier `pourCCTP.txt` et en suivant le format d'un fichier `.ppm` décrit plus haut, écrivez la fonction **tImCouleurP3 chargePPM(char* fichier)** qui charge dans une structure `tImCouleurP3` l'image contenue dans le fichier donnée en paramètre.
Une partie du code est donnée, vous devez le compléter. Après avoir passé les lignes de commentaire, chaque donnée étant séparée de la suivante par un espace ou un passage à la ligne vous pouvez utiliser la fonction `fscanf` pour les récupérer.

1.3 Sauvegarde d'une image couleur dans un fichier `.ppm`

Exercice 6

Écrivez la fonction **void sauvePPM(char* nom, tImCouleurP3 im)** qui créer un fichier `.ppm` dont le nom est donné en paramètre et dans lequel vous sauvez le contenu de la structure `tImCouleurP3` donnée en paramètre en respectant le format d'un fichier `.ppm`. Une partie du code est donnée, vous devez le compléter.
Vous écrirez dans le fichier ppm une ligne de commentaire contenant votre nom et prénom pour renseigner le créateur du fichier.
Utilisez `fprintf` pour écrire dans le fichier.

2 Transformation en niveaux de gris, pixelisation.

Une fois chargée en mémoire les opérations qu'on peut effectuer sur une image consistent essentiellement à faire des calculs sur le tableau des pixels.

2.1 Niveaux de gris

Pour transformer une image couleur en image en niveau de gris il faut affecter un niveau de gris à chaque pixel, ce niveau de gris est appelé luminance, il est calculé à partir des niveaux de couleur r, v et b.
L'oeil humain n'étant pas sensible de la même manière à ces trois couleurs, pour obtenir un bon résultat on ne peut pas se contenter de faire la moyenne simple de ces trois valeurs.
La C.I.E (Commission Internationale de l'Éclairage) propose de caractériser l'information de luminance (la valeur de gris) d'un pixel par la formule :
Gris = 0.2125 Rouge + 0.7154 Vert + 0.0721 Bleu
On donne la fonction **float luminance(tPixel p)** (cf le fichier `pourDevoir2.txt`) qui prend en paramètre un pixel couleur et qui retourne la luminance de ce pixel en appliquant la formule ci-dessus.

```
int luminance(tPixel p){
    return p.r * 0.2125 + p.v * 0.754 + p.b * 0.0721;
```

Exercice 7

Écrivez une fonction **tImGrisP2 niveauGris(tImCouleurP3 im)** qui retourne une image en niveau de gris créée à partir de l'image couleur donnée en paramètre. Utilisez la fonction précédente pour calculer la luminance de chacun des pixels. Cette fonction retourne une structure `tImGrisP2` créée à partir de la structure `tImCouleurP3` donnée en paramètre.

2.2 Sauvegarde/chargement d'une image en niveaux de gris dans/depuis un fichier `.pgm`

Exercice 8

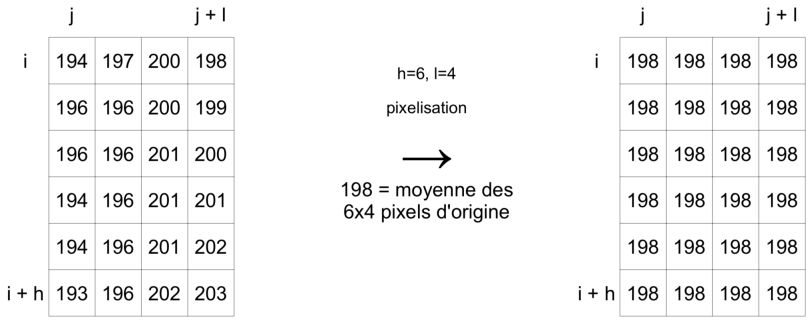
Écrivez une fonction **void sauvePGM(char* nom, tImGrisP2 im)** qui créer un fichier `.pgm` dont le nom est donné en paramètre et dans lequel vous sauvez le contenu de la structure `tImGrisP2` donnée en paramètre en respectant le format d'un fichier `.pgm`.
Inspirez-vous de la fonction de sauvegarde d'une image couleur.
N'oubliez pas la ligne de commentaire contenant votre nom et prénom pour renseigner le créateur du fichier.

Exercice 9

Écrivez la fonction **tImGrisP2 chargePGM(char* fichier)** qui charge dans une structure `tImGrisP2` l'image contenue dans le fichier donnée en paramètre.
Inspirez-vous de la fonction de chargement d'une image couleur.

2.3 Pixelisation de l'image en pavés de pixels uniformément gris

Dans cette partie vous allez pixeliser une image. Le principe est de remplacer des pavés de pixels de largeur et de hauteur données par des pavés de pixels ayant tous la même intensité de gris.
Étant données la hauteur h et la largeur l des pavés il s'agit donc de créer une image en niveaux de gris dans laquelle à chaque pavé de $h \times l$ pixels de l'image de d'origine correspond un pavé de $h \times l$ pixels tous identiques et dont la valeur est la moyenne des $h \times l$ niveaux de gris de l'image d'origine comme l'illustre le schéma ci-dessous dans lequel on considère une pavé dont le coin en haut à gauche est situé à la ligne i et la colonne j de l'image.



Attention, il est très probable que les dimensions de l'image de départ ne soient pas des multiples de la taille des pavés, dans ce cas il faut ignorer les pixels « en trop » sur la largeur et la hauteur et l'image créer n'aura donc pas exactement les mêmes dimensions que l'image d'origine.
Par exemple pour une image d'origine de hauteur 640 et de largeur 427, si on utilise des pavés de 6×4 on obtiendra comme résultat une image de hauteur 636 et de largeur 424, ces valeurs étant les multiples de 6 et 4 les plus proches de 640 et 427.

Exercice 10

Écrivez une fonction **tImGrisP2 pixelise(tImGrisP2 im, int h, int l)** qui retourne une image en niveaux de gris créée à partir de l'image donnée en paramètre en appliquant l'opération de pixelisation décrite ci-dessus.

3 Transformation en ascii art et sauvegarde

Pour créer un ascii art d'une image donnée en niveau de gris on procède de manière analogue à la pixelisation en pavés gris. Il s'agit cette fois d'associer à un pavé de $h \times l$ pixels de l'image de départ un caractère ascii qui va représenter la luminance moyenne du pavé de pixels (la moyenne des niveaux de gris du pavé).
Il n'est pas nécéssaire d'utiliser beaucoup de caractères pour obtenir un résultat satisfaisant. On utilisera les 10 caractères suivants qui représentent une échelle de niveaux de gris du plus foncé (l'espace) au plus clair (l'arobase). On stockera ces caractères dans un tableau de 10 caractères. Étant donné une intensité de gris G compris entre 0 et 255 nous lui associons le caractère dont l'indice entre 0 et 9 est obtenu par le calcul $(255 - G) * 10/256$ (division entiere).

	.	,	:	;	o	x	%	#	@

L'exemple ci-dessous illustre l'opération : pour un pavé de 6×4 pixels de l'image en niveau de gris on calcule la moyenne des valeurs des pixels, soit 198. L'indice du caractère correspondant est $(255 - 198) * 10/256 = 2$ et dans l'ascii art à créer ce pavé de pixels est représenté par le caractère virgule.

194	197	200	198
196	196	200	199
196	196	201	200
194	196	201	201
194	196	201	202
193	196	202	203

h=6, l=4

→

198 = moyenne des 6x4 pixels d'origine

(255-moyenne)*10/256 = 2

0	1	2	3	4	5	6	7	8	9
	.	,	:	;	o	x	%	#	@

On donne ci-dessous (c'est aussi dans le fichier `pourCCTP.txt`) la définition d'une structure permettant de stocker en mémoire un tableau de `nblig` lignes de `nbcol` caractères.

```
// Le type tableau ascii art
typedef struct asciiart {
    int nblig;      // nombre de lignes
    int nbcol;      // nombre de colonnes
    char** tabcar;  // le tableau des caractères
} tASCIIArt;
```

Exercice 11

Écrivez une fonction **tASCIIArt creerAsciiArt(tImGrisP2 im, int h, int l)** qui retourne un créé à partir de l'image donnée en paramètre en associant à chaque pavé de $h \times l$ pixels le caractère correspondant à la moyenne des pixels de ce pavé.
Remarques :

- cette fonction ressemble beaucoup à la fonction de pixellisation sauf qu'au lieu de créer une image contenant des pavés gris on créer un tableau de caractères ;
- les dimensions du tableau de caractères contenant l'ascii art dépendent évidement des dimensions de l'image de départ, de la hauteur et de la largeur des pavés. Plus les pavés sont petits, plus le résultat obtenu est détaillé mais plus l'ascii art généré est grand. Plus les pavés sont grands moins l'ascii art généré est grand et plus le résultat obtenu est grossier.

Exercice 12

Écrivez une fonction **void sauveASCII(char* nom, tASCIIArt art)** qui sauve dans un fichier texte dont le nom est donné en argument le tASCIIArt donné.

4 Programme principal

Exercice 13

Ecrivez la fonction `main.c` de votre programme de manière à tester toutes les opérations programmées. **N'oubliez pas d'inclure le module `tpimage.h`.**
Le programme principal doit au minimum afficher le menu suivant puis réaliser l'opération choisie :

```
Quelle opération voulez-vous effectuer ?
1. Charger une image couleur en mémoire
2. Sauver une image couleur
3. Transformer une image couleur en niveau de gris
4. Charger une image en niveau de gris en mémoire
5. Sauver une image en niveau de gris
6. Pixeliser une image en niveau de gris
7. Créer un ASCII Art à partir d'une image en niveau de gris
8. Sauver un ASCII Art
0. Quitter
Choix ==>
```

Les paramètres (nom de fichier à charger/sauver, dimension des pavés pour la pixelisation) de chacune de ces opérations seront saisis par l'utilisateur.

5 Annexes

Le dossier `CCTP` fournit pour ce devoir contient :
- L'ennoncé du devoir `S2IN324-CCTP.pdf`
- Trois images couleur de type PPM sur lesquelles tester votre programme (`ampoule.ppm`, `fleur.ppm`,`souris.ppm`) ;
- Un fichier `pourCCTP.txt` dans lequel vous trouverez :
  - la définition des types utilisés (`tPixel`, `tImCouleurP3`, `tImGrisP2`, `tAsciiArt`).
  - le squelette de la fonction `chargePPM` ;
  - le squelette de la fonction `sauvePPM` ;
  - le code de la fonction `luminance` qui calcule la luminance d'un pixel couleur ;