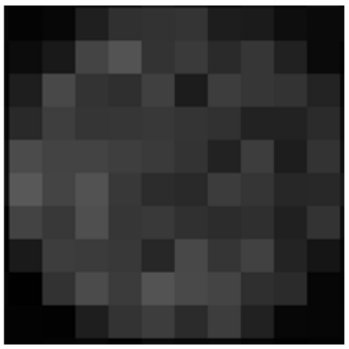
Traitement d'images

Consignes obligatoires :

- Mettre en forme votre code (tabulations, …).
- Donner des noms de variables explicites.
- Commenter votre code et surtout les fonctions !!!
- Ecrire les prototypes des fonctions.
- Gérer les erreurs en affichant des messages.

Introduction

Une **image** peut être représentée par un tableau d'entiers en deux dimensions (DIMX, DIMY). Cette image aura alors DIMX lignes et DIMY colonnes. Chaque case du tableau est alors assimilée à un **pixel**.
L'**intensité** d'un pixel est la valeur entière de la case du tableau représentant l'image. Lorsqu'on parle d'images 8 bits, les intensités sont comprises entre 0 et $2^8 - 1$, c'est-à-dire 255.



56	48	56	85	112	156	111	111	83	63
48	80	121	139	129	119	117	132	108	70
58	88	131	127	122	131	115	127	127	128
69	101	114	118	135	101	121	112	129	121
82	169	142	166	175	117	86	135	102	92
80	164	160	138	135	122	109	114	105	99
86	145	138	106	90	129	123	113	85	140
79	137	163	206	149	171	99	147	109	118
52	54	194	202	152	166	133	149	159	136
57	49	123	102	105	128	125	139	140	93

Figure 1 : Image et pixels

Réflexion 01 : De combien de pixels est constituée l'image ?

Réflexion 02 : En langage C, quels indices représentent la première et la dernière ligne, ainsi que la première et la dernière colonne?

Réflexion 03 : Entre quelles bornes sont comprises les intensités pour une image 1 bit (dite binaire) ? et pour une image de 16 bits ?

L'**histogramme** d'une image est une représentation graphique qui, pour chaque intensité, donne la fréquence d'apparition de cette intensité.

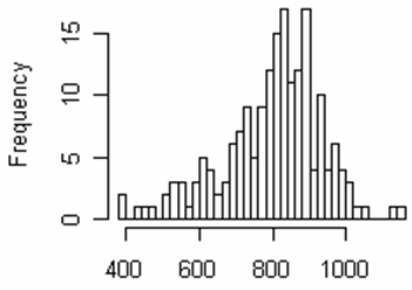


Figure 2 : Histogramme d'une image

Si on ne considère pas les pixels du bord de l'image, chaque pixel P possède **8 voisins** dans un voisinage dit de **8-connexité**.

X	X	X
X	P(i,j)	X
X	X	X

Figure 3 : Voisinage 8-connexité

Réflexion 04 : En considérant que le pixel courant est à la $i^{ème}$ ligne et $j^{ème}$ colonne, quels sont les indices des 8 voisins?

Présentation du format PGM

Le format PGM (Portable Grey Map) est l'un des formats d'image en niveaux de gris les plus simples à utiliser. Il consiste en la retranscription brute des intensités des pixels séparés par des espaces. Il n'y a pas d'algorithme de compression, ce qui rend l'ouverture et la sauvegarde des fichiers extrêmement faciles.

```
P2
31 7
9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 3 0 0 0 3 0 5 5 5 5 5 0 7 7 7 7 7 0 0 0 9 0 0 0
0 1 0 0 0 0 0 3 3 0 0 3 0 5 0 0 0 0 0 7 0 0 0 0 0 0 9 0 9 0 0
0 1 1 1 0 0 0 3 0 3 0 3 0 5 5 5 5 5 0 7 7 7 0 0 0 0 9 9 9 0 0
0 1 0 0 0 0 0 3 0 0 3 3 0 0 0 0 0 5 0 7 0 0 0 0 0 9 0 0 0 9 0
0 1 1 1 1 1 0 3 0 0 0 3 0 5 5 5 5 5 0 7 7 7 7 7 0 9 0 0 0 9 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 4 : Exemple d'image PGM

Explications sur le format

- « P2 » est le « nombre magique », c'est-à-dire ce qui permet de reconnaître ce format d'image.
- Sur la deuxième ligne, on lit deux nombres séparés par un espace représentant la largeur (31) et la hauteur de l'image (7).
- Sur la troisième ligne, on lit la valeur maximale pour coder les niveaux de gris (9). Cette valeur ne doit pas dépasser 65535.
- Les niveaux de gris des pixels (7x31) sont ensuite retranscrits. Il n'y a pas de ligne vide à la fin du fichier.

Remarque : on peut insérer des commentaires dans ce type de fichier à l'aide de #. Dans ce projet, on considérera uniquement des images sans commentaires.

Ouverture et sauvegarde d'une image PGM

Le but est de pouvoir récupérer les pixels et de les mettre dans un tableau pour qu'on puisse faire le traitement. Pour ce faire, on doit d'abord lire l'entête du fichier et récupérer la taille afin de pouvoir allouer dynamiquement la mémoire et rendre le programme le plus général possible. Les étapes développées ci-dessous vous permettront de réaliser cet objectif.

Do it 1

- Créer un répertoire dans lequel on va mettre tout le projet.
- Créer une image PGM 8 bits et la mettre dans le répertoire.
- Créer un projet dans lequel on distinguera 3 fichiers :
  - celui contenant la fonction principale « main.c »
  - celui contenant les prototypes des fonctions « fonctions.h »
  - celui contenant les fonctions elles-mêmes « fonctions.c »

Do it 2

Créer le squelette d'une structure IMAGE pour conserver des informations importantes qu'on aura lues dans le fichier. Cette structure doit comporter les champs suivants :

- char *nom_fichier
- int largeur, int hauteur, int max
- unsigned char **pixels
-

Do it 3

Créer la fonction « createImage » qui doit :

- réserver en mémoire la taille d'une structure IMAGE,
- réserver en mémoire la taille de tous ses champs internes
- renvoyer l'adresse réservée à la structure IMAGE.

Le prototype est donné ci-dessous.

```
IMAGE* createImage(int largeur, int hauteur, char* nom_fichier) ;
```

Voici un bout de code permettant d'initialiser un double pointeur :

```
int i;
unsigned char**pixels = (unsigned char**)malloc(H*sizeof(unsigned char*)) ;
for(i=0;i<H;i++){
    *(pixels+i) = (unsigned char*)malloc(L*sizeof(unsigned char));
}
```

Tester la fonction dans le « main » en mettant des arguments fictifs.

Do it 4

Créer la fonction « openImagePGM » qui doit :

- ouvrir une image PGM en lecture dont le nom est donné en argument
- charger les pixels dans le tableau interne d'une structure IMAGE.
- renvoyer l'adresse réservée à la structure IMAGE.

Le prototype est donné ci-dessous.

```
IMAGE* openImagePGM(char *nom_fichier) ;
```

Cette fonction appellera « createImage » pour définir la structure IMAGE dont elle remplira remplir les champs et qu'elle renverra en retour.

Tester la fonction dans le « main ».

Do it 5

Créer la fonction « saveImagePGM » qui, à partir des données contenues dans une structure image, doit

- sauvegarder une image au format PGM
- renvoyer un entier pour savoir si tout s'est bien passé.

Le prototype est donné ci-dessous.

```
int saveImagePGM(IMAGE *image) ;
```

Tester la fonction dans le « main ».

Fonctions de base

Do it 6

La plupart des traitements demandent une nouvelle copie de l'image chargée. Il va donc falloir une fonction permettant de dupliquer une image en mémoire.

Créer la fonction « copyImage » qui, à partir des données contenues dans une structure image, doit

- créer une nouvelle structure IMAGE, copie conforme de la structure passée en paramètre
- renvoyer l'adresse de la nouvelle structure.

Le prototype est donné ci-dessous.

```
IMAGE* copyImage(IMAGE *image) ;
```

Tester la fonction dans le « main ».

Filtrage et opérations morphologiques

Dans la suite du problème, on va utiliser une image référence qu'on va filtrer pour obtenir une nouvelle image. Pour ce faire, toutes les fonctions prendront comme arguments deux tableaux, l'un représentant l'image à filtrer, et l'autre qu'on initialisera et qui contiendra l'image filtrée.

La notation P (respectivement P') représente un pixel dans l'image à filtrer (respectivement image filtrée). Pour éviter tout problème d'effets de bord, les filtres et opérations seront appliqués à partir de l'indice 1 et ce jusqu'à DIMX (ou DIMY) – 1.

I. Filtrage

Question 14 : Créez une image de référence contenant :

- un rectangle de largeur 6, de longueur 8, d'intensité 4 et centré sur l'image
- quatre carrés de côté 4 dont les coins gauches sont respectivement : (1,1), (15,1), (1,15) et (15,15), et d'intensités respectives 1, 3, 5 et 7.

Question 15 : Affichez l'image de référence.

1. Filtre moyenneur

Ce filtre de type passe-bas permet de lisser les contours de l'image.
Pour chaque pixel P, il fait la moyenne de P et de ses 8 pixels voisins et affecte le résultat à un pixel P' d'une nouvelle image situé à la même position que P.
Par analogie on dit qu'il applique à l'image l'élément structurant suivant:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Exemple :

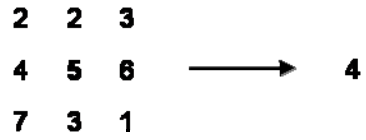


Figure 5 : Calcul d'un filtre moyenneur

Le pixel courant P est 5. Après moyennage, la valeur du pixel P' (ayant la même position que P sur l'image de départ, c'est-à-dire les mêmes indices) sur la nouvelle image est 3,66667 arrondi à 4 (une image ne comporte que des valeurs entières).

Question 16 : Créez une fonction **filtre_moyenneur** qui effectue un filtrage moyenneur.

- Objectif : filtrer l'image en utilisant un moyenneur
- Argument : l'image à filtrer, l'image filtrée
- Sortie : aucune
- Prototype de la fonction : **void filtre_moyenneur (int ref[DIMX][DIMY], int done[DIMX][DIMY]) ;**

Question 17 : Testez la fonction et ajoutez cette opération de filtrage au menu.

2. Filtre laplacien

Ce filtre de type passe-haut permet de trouver les contours des objets contenus dans une image. Pour chaque pixel P, il applique l'élément structurant suivant et affecte le résultat à P'.

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Exemple :

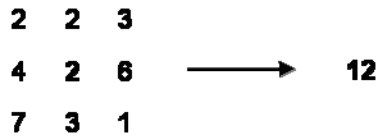


Figure 6 : Calcul par un filtre laplacien

Le pixel courant est le 2 situé au centre du carré. La valeur P' associée sera alors de : P'=2+2+3+4+6+7+3+1-8*2=12. Attention, si P'>8, alors P'=8 et si P'<0, alors P'=0.

Question 18 : Créez une fonction **filtre_laplacien** qui applique un filtre laplacien.

- Objectif : filtrer l'image en utilisant un laplacien
- Argument : l'image à filtrer, l'image filtrée
- Sortie : aucune
- Prototype de la fonction : **void filtre_laplacien (int ref[DIMX][DIMY], int done[DIMX][DIMY]) ;**

Question 19 : Appliquez-le sur l'image référence. Que constatez-vous ? Ajoutez cette opération de filtrage au menu précédent.

II. Opérations morphologiques

Question 20 : Créez une image binaire contenant un carré de côté 6 d'intensité 1 et centré sur l'image. Les deux opérations présentées dans la suite constituent la base de toute opération morphologique. A partir de compositions de ces fonctions et d'opérations ensemblistes, on arrive à créer la plupart des outils de morphologie mathématique.

1. Erosion

Cette opération consiste à prendre le minimum sur le voisinage du pixel P et à l'affecter à P'. Pour une image binaire, il suffit de tester la présence d'un 0.

Question 21 : Créez une fonction **erosion** qui applique l'érosion.

- Objectif : faire une érosion sur l'image référence.
- Argument : l'image à filtrer, l'image filtrée
- Sortie : aucune
- Prototype de la fonction : **void erosion (int ref[DIMX][DIMY], int done[DIMX][DIMY]) ;**

Question 22 : Ajoutez cette opération morphologique au menu précédent. Sur l'image binaire créée précédemment, appliquez successivement 4 érosions. Que constatez-vous?

2. Dilatation

Cette opération consiste à prendre le minimum sur le voisinage du pixel P et à l'affecter à P'. Pour une image binaire, il suffit de tester la présence d'un 0.

Question 23 : Créez une fonction **dilatation** qui applique la dilatation.

- Objectif : faire une dilatation sur l'image référence.
- Argument : l'image à filtrer, l'image filtrée
- Sortie : aucune
- Prototype de la fonction : **void dilatation (int ref[DIMX][DIMY], int done[DIMX][DIMY]) ;**

Question 24 : Ajoutez cette opération morphologique au menu précédent. Sur l'image binaire créée précédemment, appliquez successivement 4 dilatations. Que constatez-vous?

Création d'une DLL (Dynamic Link Library)

Suivre les étapes suivantes:

1. Créer un nouveau projet
2. Créer un prototype dans le fichier « dll.h »
3. Créer la fonction dans le fichier « dllmain.h »
4. Compiler.
5. Créer un projet test qui permettra d'appeler la DLL.
6. Configurer le projet test en lui indiquant le lien avec la DLL.
7. Créer une fonction principale dans ce projet test et appeler une fonction de la DLL.
8. Compiler et exécuter le projet test.