IN 101 - TD 06 **Énoncé**

1 Images en noir et blanc

≪Écrivez un programme permettant transformer une image initialement en niveaux de gris en une image en noir et blanc selon un seuil. ≫

Chaque point d'une image en niveau de gris est représenté par une valeur **entière** $\in [0; max]$ pour un max donné (explicitement dans le format de représentation de l'image). La valeur d'un point représente sa luminosité ($0 \rightarrow$ noir, $max \rightarrow$ blanc, entre les deux $\rightarrow$ un niveau de gris). Pour représenter une image en noir et blanc, il suffira de n'utiliser que les deux valeurs 0 et 1 (donc max vaudra 1 pour l'image résultat).

Votre programme devra traiter des images au format `pgm`. C'est une représentation sous forme **textuelle** d'une image. Un fichier dans ce format **commence toujours** par la chaîne "P2", suivie de la **largeur** de l'image, de sa **hauteur**, puis de la valeur max dont il est question ci-dessus (séparées par des ≪blancs ≫ – retours à la ligne, espaces). Ensuite viennent, séparées par des ≪blancs ≫, les valeurs des points de l'image.

Ainsi, le fichier `ensta.pgm` contenant l'image suivante (réduite pour rentrer dans la page) :

a le contenu suivant, où l'on reconnaît la chaîne fixe "P2" suivie de la largeur de l'image (870 pixels), sa hauteur (438 pixels), la valeur maximale de niveau de gris (255, qui correspond donc à du ≪blanc ≫), puis la suite de pixels (les 3 premiers valant 194, le $4^{\text{ème}}$ 195, etc.).

```
P2
870
438
255
194
194
194
195
...
```

De nombreux fichiers au format `pgm` vous sont donnés pour tester votre futur programme. Pour visualiser une image dans votre environnement Linux, vous pouvez par exemple utiliser les commandes `eog` ou `display`. Sinon, l'explorateur de fichiers vous le permettra à la souris.

Q 1.1. Quelles sont les grandes étapes du programme ?

Q 1.2. Quelle structure de données allez-vous utiliser pour représenter l'image ?

Q 1.3. Identifiez tous les cas où le contenu d'un fichier est incorrect vis-à-vis à vis du format `pgm`.

Souvenez-vous des fonctions de manipulation de fichiers et de la fonction `fscanf` que nous avons vues au TD 3.

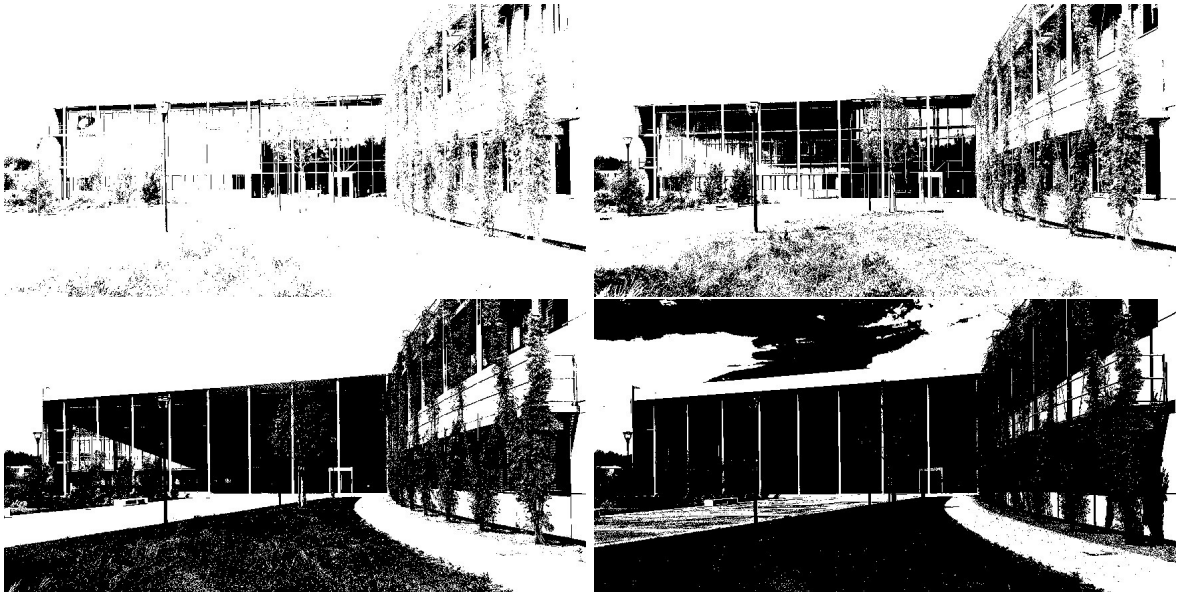
Q 1.4. Décrivez l'algorithme de la fonction permettant de charger le contenu d'un fichier, d'en créer une image et de la retourner. Quels sont ses domaines d'entrée et de sortie ? Comment peut-on signaler une erreur ?

Q 1.5. Décrivez l'algorithme de la fonction permettant de sauvegarder dans un fichier une image passée en argument. Quels sont ses domaines d'entrée et de sortie ?

Comme introduit au début de l'exercice, transformer une image de niveaux de gris en noir et blanc (donc **2** ≪couleurs≫) se fait en transformant chaque pixel en l'une de ces deux couleurs. Le choix est effectué en fonction d'un **seuil**. Si un point a un niveau de gris inférieur au seuil, alors il sera transformé en ≪noir≫ dans l'image résultat, sinon en ≪blanc≫.

Par convention, vu du côté d'un utilisateur, une image en niveau de gris est **toujours** en **256 niveaux de gris**. Aussi, le seuil spécifié par l'utilisateur sera naturellement et **obligatoirement** $\in [0; 255]$.

Les images ci-dessous représentent le résultat du seuillage avec les seuils 50, 100, 150 et 200.



Q 1.6. Décrivez l'algorithme qui permet de transformer une image en noir et blanc en fonction d'un **seuil** passé en **paramètre**.

Du point de vue de l'utilisateur, une image comportera toujours **256** niveaux de gris ≪logi-

ques ≫. Il devra donc fournira donc son seuil entre 0 et 255. En effet, il n'est pas censé connaître le contenu ≪technique ≫ de l'image, donc le nombre réel de niveaux de gris qu'elle comporte.

Q 1.7. Comment allez-vous ramener le seuil donné par l'utilisateur dans l'intervalle propre à l'image ?

Q 1.8. Écrivez la fonction `load_pgm` dont vous avez donné l'algorithme en Q1.4.

Q 1.9. Pour vérifier la robustesse de votre fonction de chargement, testez-la avec les trois fichiers **corrompus** :

- `mangled_extra_pixels.pgm` : trop de pixels (ce cas peut ou non lever une erreur, on peut considérer que l'on ignore les pixels en trop),
- `mangled_missing_pixels.pgm` : pas assez de pixels,
- `early_bad.pgm` : entête incomplet.

Q 1.10. Écrivez la fonction `save_pgm` qui permet de sauvegarder dans un fichier une image passée en argument et dont vous avez esquissé l'algorithme en Q1.5.

Q 1.11. Écrivez la fonction `binarize` qui permet de transformer une image en noir et blanc en fonction d'un **seuil** passé en **paramètre** dont vous avez esquissé l'algorithme en Q1.6.

Q 1.12. Écrivez le `main` qui orchestre la transformation de l'image, depuis son chargement jusqu'à sa sauvegarde. Cette fonction récupérera sur la **ligne de commande** :

1. le nom du fichier image à charger,
2. le nom de fichier sous lequel sauvegarder l'image transformée,
3. le seuil de transformation.

Q 1.13. Testez votre programme avec des images qui vous sont données : `ensta.pgm`, `minions.pgm`, `flower.pgm`, `trooper.pgm`, `road.pgm` ou `marvin.pgm`.

N'essayez pas avec `rabbid.pgm` qui sert pour la partie ≪Pour aller plus loin ≫ (son format fera échouer le chargement).

S'il vous reste du temps ou pour continuer après la séance.

2 S'il vous reste du temps : le vrai format `pgm`

≪Écrivez une fonction permettant de charger une image au véritable format `pgm`. ≫

Dans la réalité, un fichier `pgm` peut contenir des lignes de commentaire commençant par le caractère `#`. Le fichier `rabbid.pgm` illustre cette structure.

```
P2
# CREATOR: GIMP PNM Filter Version 1.1
300 437
255
255
255
      # Comment in the middle not starting in column 0.
255
```

```
255  # Comment after a value.
255
255 255 255 255
255 255 255 255 255 255 255
255
255
(...)
```

Q 2.1. Que change la présence de commentaires dans la manière dont vous allez lire le contenu du fichier ?

La bibliothèque standard de `C` fournit la fonction :

int fgetc(FILE *stream)

qui permet de lire 1 caractère dans le fichier `stream`. Elle renvoie la valeur `EOF` lorsqu'elle ne peut plus lire car la fin du fichier a été atteinte.

Q 2.2. Nous souhaitons trouver l'algorithme de notre fonction `my_scanner` qui va remplacer `fscanf` utilisé jusqu'à présent et retourner une chaîne de caractères. Ainsi, `my_scanner` devra trouver des caractères consécutifs qui ne sont pas des « blancs » ni des commentaires.

Réfléchissez aux différents cas de détection de début et de fin de chaîne. Esquissez l'algorithme de `my_scanner`.

Petite indication : si vous détectez un problème lors de la détermination de fin de chaîne à cause d'un début de commentaire, il existe la fonction `ungetc` qui permet de réinjecter un caractère dans un fichier à la position actuelle. Utilisez la commande `man` pour obtenir des détails sur cette fonction (`man 3 ungetc`).

Q 2.3. Pensez à comment vous aller écrire cette fonction en `C`. Quel vont être les domaines des entrées et des sorties de cette fonction ?

Q 2.4. Quel va être l'impact de l'utilisation de `my_scanner` dans votre fonction `load_pgm` ?

Q 2.5. Implémentez la fonction `my_scanner` et modifiez votre `main` en conséquence. Testez ensuite votre programme avec le fichier `marvin.pgm`.