# TP 2 bis

Dans cet exercice vous devez écrire le programme `convert_pgm` qui convertit un fichier image au format PGM en mode binaire (en anglais *raw* ) en un fichier image au format PGM en mode texte ou ASCII (en anglais *plain*). Le programme prend deux arguments en entrée :

```
./convert_pgm <file_in.pgm> <file_out.pgm>
```

où `file_in.pgm` est le nom du fichier d'entrée et `file_out.pgm` est le nom du fichier de sortie qui sera écrit sur le disque.

## Description du format PGM

Un fichier image PGM permet de stocker une image en niveaux de gris. Il est composé d'une entête contenant les informations suivantes :

- le *magic number* du format (deux octets) : il indique le type de format (PGM) et la variante (binaire : "P5" ou ASCII : "P2")
- un caractère d'espacement (espace, tabulation, nouvelle ligne)
- la largeur de l'image (nombre de pixels, écrit explicitement sous forme d'un nombre en caractères ASCII)
- un caractère d'espacement
- la hauteur de l'image (nombre de pixels, écrit explicitement sous forme d'un nombre en caractères ASCII))
- un caractère d'espacement
- la valeur maximale des pixels représentés dans l'image (écrite explicitement sous la forme d'un nombre en caractères ASCII)

puis des données de l'image, sous la forme d'une suite de valeurs associées à chaque pixel :

- l'image est codée ligne par ligne en partant du haut
- chaque ligne est codée de gauche à droite

Dans le cas du format PGM binaire (*raw*), correspondant au *magic number* `P5`, les données de l'image sont stockées sous la forme d'une suite de valeurs encodées au format binaire.

Dans le cas du format PGM texte (*raw*), correspondant au *magic number* `P2`, les données de l'image sont stockées sous la forme d'une suite de valeurs écrites sous la forme de nombres en caractères ASCII, séparées par un espace.

Toutes les lignes commençant par le caractère croisillon `#` sont ignorées (lignes de commentaires).

## Travail à effectuer

L'objectif est d'écrire un programme qui lit un fichier PGM au format binaire (*magic number* `P5`) et le convertit en un fichier PGM au format texte (*magic number* `P2`).

Le programme ne traitera que les fichiers PGM pour lesquels la valeur maximale est inférieure à 255 (c'est à dire contenant des données codées sur 1 octet). Dans le cas contraire, le format sera considéré invalide. On fait également l'hypothèse que le fichier d'entrée ne contient *pas* de commentaires (lignes commençant par le caractère `#`). Votre programme n'aura donc pas à tester la présence de commentaires.

Le programme :

- doit vérifier que le *magic number* du fichier d'entrée est `P5`
- doit vérifier que le nombre de champs écrits dans l'entête est valide
- doit vérifier que le champ `MAXVAL` de l'entête est inférieur ou égal à 255
- doit vérifier que le nombre de valeurs associées aux pixels est bien égal à la taille de l'image DIMX $*$ DIMY, où DIMX et DIMY représentent respectivement la largeur et la hauteur lues dans l'entête de l'image d'entrée.

Le programme renvoit le code 0 s'il n'y a eu aucune erreur, le code 1 en cas d'erreur sur un appel système, et le code 2 si le fichier d'entrée ne correspond pas à un fichier PGM valide.

Le programme doit écrire l'entête du fichier de sortie de la manière suivante :

```
P2
DIMX DIMY
MAXVAL
```

avec DIMX, DIMY, MAXVAL respectivement la largeur, hauteur, valeur maximale de l'image d'entrée.

Les données du fichier de sortie sont stockées au format texte sous la forme de DIMY lignes de DIMX valeurs avec un espace avant et après, et un caractère retour chariot `\n` à la fin de chaque ligne.

Exemple de fichier de sortie représentant une image de largeur 5 pixels et de hauteur 3 pixels :

```
P2
5 3
255
0 255 0 255 0
255 0 255 0 255
0 255 0 255 0
```

Votre fichier de sortie doit être identique à celui obtenu avec la commande (disponible sur turing) :

```
convert file_in.pgm -compress none file_out.pgm
```

Vous pourrez visualiser les images au format PGM avec le programme `imagej` installé sur `turing`.

Vous écrirez le programme `convert_pgm` en langage C en utilisant uniquement les primitives système. L'utilisation d'un buffer pour la lecture/écriture des données donnera lieu à un bonus.

Les seules fonctions de bibliothèque autorisées sont pour l'affichage (par exemple `fprintf`, `perror`).

Pour la conversion d'une chaîne de caractères en un entier vous pourrez coder votre propre fonction ou utiliser la fonction `atoi`. De même, pour convertir un entier en chaîne de caractères, vous pourrez coder votre propre fonction ou utiliser la fonction `sprintf`.

Votre programme doit vérifier les valeurs de retour de toutes les primitives système utilisées. Votre programme ne devra pas comporter d'allocation dynamique de mémoire.

Vous apporterez un soin particulier à la mise en forme de façon à rendre un code lisible et commenté à bon escient. Référez-vous au document « Conseils de programmation pour réussir vos projets et TP » (mis à votre disposition sur Moodle) et, si besoin, utilisez l'utilitaire `indent` avec la configuration donnée en fin du document.

Votre programme doit compiler avec la commande suivante (disponible dans le `Makefile` mis à disposition sur Moodle) : `cc -Wall -Wextra -Werror`
Les programmes qui ne compilent pas sur **turing** avec cette commande **ne seront pas examinés**.

Un script de test est mis à votre disposition. Celui-ci exécute votre programme sur des jeux de tests qui serviront de base à l'évaluation de votre rendu. La commande suivante permet de lancer les tests :

```
make test
```

Vous devrez rendre sur Moodle un *unique* fichier nommé `convert_pgm.c` et il est interdit de rendre un fichier d'un autre nom ou une archive.

Ce TP à rendre est **individuel**. On rappelle que la copie ou le plagiat sont sévèrement sanctionnés.