Projet C pour le système

PixmapConvertor

Réalisé par *Damien Wykland* et *Ancelin Serre*

Objectifs du projet

Ce projet a pour objectif de proposer un programme simple, utilisable via un terminal et permettant de lire des images au format *.ppm* (pour *Portable Pix Map*) dans le but des les convertir, selon le choix de l'utilisateur, soit en image en niveaux de gris avec le format *.pgm* ou en représentation binaire avec le format *.pbm* donc en noir et blanc.

Du point de vue scolaire, ce projet nous a permis de mettre en application les notions avancées du langage C abordées lors du cours de **C pour le système** comme les opérations sur les bits d'une variable via *l'utilisation d'opérateurs binaires*, *l'utilisation avancée de pointeurs* ou encore la *lecture et l'écriture dans un fichier*.

Organisation générale du code

Pour mener à bien ce projet, nous avons respecté l'architecture imposée dans le sujet. On retrouve donc, à la racine du projet, l'arborescence suivante :

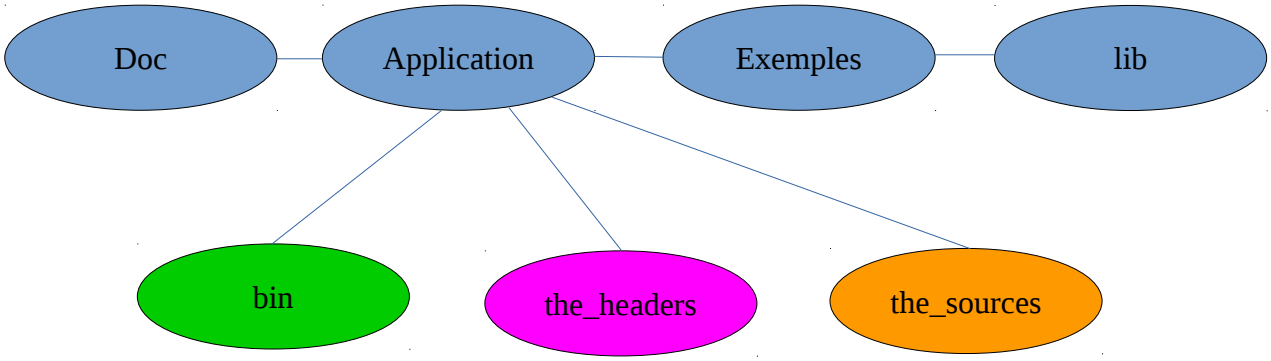

Figure 1. Schéma de l'arborescence de l'application

C'est dans le répertoire Application que l'on va retrouver la quasi totalité du projet avec le Makefile. L'ensemble des sources et des fichiers d'en-têtes sont respectivement placés dans les dossiers **the_sources** (Makefile compris) et **the_headers**. Le Makefile se charge de générer l'exécutable du projet dans le répertoire **bin**, les bibliothèques statiques et dynamiques se trouvent pour leur part dans le dossier **lib**.

Des fichiers aux formats *.pgm*, *.pbm* et *.ppm* sont finalement mis à disposition dans le dossier **Exemples** afin de tester les fonctionnalités du programme.

Choix de conception

Déroulement principal du programme

Pour utiliser ce programme, l'utilisateur doit écrire dans son terminal une des commandes suivantes :

```
./main -g image.ppm
./main -b image.ppm
./main -b (ou -g)
```

On remarque ici les deux arguments passables au programme qui sont **-g** et **-b** qui servent respectivement à convertir l'image passée en argument au format *.pgm* ou au format *.ppm*. L'image donnée par l'utilisateur doit bien évidemment être au format *.ppm* sans quoi le programme ne fonctionnera pas et renverra un message d'erreur. On peut toutefois se passer de renseigner une image en écrivant directement toute la structure de son image sur l'**entrée standard**.

Une fois la commande validée par la fonction **main** du programme, l'application va procéder tout d'abord à la lecture du fichier image (ou de l'entrée standard) puis à sa conversion dans le format souhaité avant de finalement écrire la nouvelle image dans un fichier avec la bonne extension dans le dossier **Exemples** du projet.

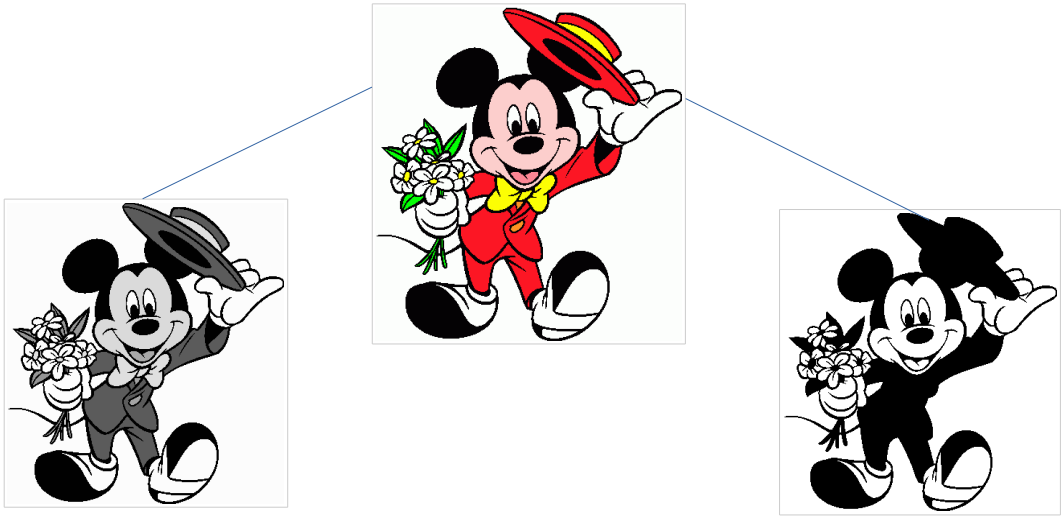

Figure 2. Exemple de résultat avec à gauche l'image en *.pgm* et à droite en *.pbm*

N.B. Des messages d'erreurs seront affichés avant l'interruption du programme si l'utilisateur renseigne une image dans un autre format que le .ppm, si l'image est mal codée, si l'argument est invalide et si le nom du fichier donné ne correspond à aucun fichier.

Représentation d'une image

Avant d'initier la lecture d'une image, il est nécessaire de connaître la façon dont sa structure est représentée. Dans notre cas, on travaille uniquement avec des fichiers au format *.ppm*, ces derniers se présentent sous la forme suivante :

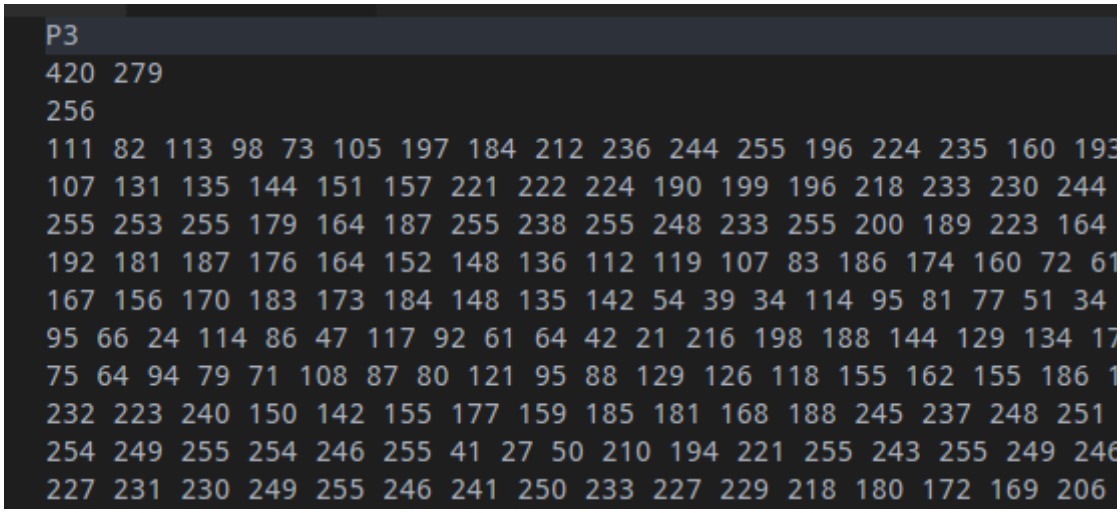

Figure 3. Structure d'une image au format *.ppm*

On trouve sur la première ligne ce que l'on appelle **le nombre magique** dont la mission est de donner le type du fichier. Sur la seconde ligne, on trouve les dimensions de l'image avec dans l'ordre **la largeur** et **la hauteur**. On a ensuite **la valeur maximum** (ici 256) que peut prendre une composante et finalement l'ensemble des **pixels** de l'image.

Nous avons donc choisi de représenter une image avec une structure de donnée comprenant les champs suivants (voir fichier **the_headers/struct.h**) :

```c
/* Définition de la structure de l'image */
typedef struct
{
  image_type type;
  uint32_t width;
  uint32_t height;
  uint16_t maxValue;
  uint64_t *data;
} image;
```

Figure 4. Structure de donnée d'une image au format *.ppm*

Le champ type est un type énuméré correspondant au type de l'image. Ce type peut prendre dans notre cas les valeurs suivantes : *P1*, *P2*, *P3*. La largeur et la hauteur sont stockés dans les champs **width** et **height**, la valeur maximum de l'image est stockée dans le champ **maxValue** et l'ensemble des pixels est rangé dans un tableau d'entier non signé de 64 bits **data**.

N.B. Les types de ces champs sont fixés dans le sujet du projet.

On peut remarquer qu'un pixel tient sur un entier non signé de 64 bits ce qui nous a forcé à découper chacun de ces entiers en 3 pour y ranger leurs valeurs des composantes rouge, verte et bleue. Le calcul pour cette répartition des composantes est décrit dans la fonction **fillPixel** présente dans le fichier **the_sources/struct.c.** Son code est le suivant :

```c
uint64_t fillPixel(uint16_t red, uint16_t green, uint16_t blue)
{
  uint64_t red64 = 0;
  uint64_t green64 = 0;
  uint64_t blue64 = 0;

  /* La composante rouge est écrite à partir du 32ème bit */
  red64 = red;
  red64 = red64 << 32;
  /* La composante verte est écrite du 16ème au 31ème bit */
  green64 = green;
  green64 = green64 << 16;
  /* La composante bleue est écrite du bit 0 au bit 15 */
  blue64 = blue;

  /* On fait un OU binaire pour ajouter chaque composante dans le pixel */
  red64 = red64 | green64 | blue64;

  return red64;
}
```

Figure 5. Fonction fillPixel chargée de coder un pixel dans un entier non signé de 64 bits

Lecture d’une image

La lecture d’une image se réalise simplement via les fonctions **readImage** ou **readSTD ,** selon le cas, présentes dans le fichier **the_sources/readimg.c.** On récupère d’abord les champs de description : (type, taille, valeur max) puis on alloue l’espace nécessaire pour cette nouvelle image et on complète son champ data avec les pixels. On a choisi de créer une nouvelle image dans une optique d’amélioration si on décide un jour d’ajouter un autre format de conversion. On a aussi décidé de procéder ainsi pour ne pas modifier l’image initiale de l’utilisateur.

Modification d’une image

Les fonctions de conversion d’image **convertPGM** et **convertPBM** sont présentes dans le fichier **the_sources/processing.c**.

- Pour **convertPGM**, on doit transformer les 3 composantes rouge, verte, et bleue en une seule composante en niveau de gris via un calcul qui vise à multiplier chaque composante par des constantes prédéfinies et à sommer ces résultats.
- Pour **convertPBM**, on procède également à un calcul afin de transformer selon l’intensité lumineuse du pixel, la couleur en noire ou en blanc d’où l’aspect binarisation.

N.B. On a eu à développer une fonction pour élever un nombre à une certaine puissance car on manipule des unsigned int.

Écriture d’une image

L’écriture d’une image reprend dans le sens inverse la procédure de lecture d’une image. Il n’y a rien de très complexe dans tout cela. Pour plus de lisibilité on fait en sorte de n’afficher que 6 pixels par ligne. Les fonctions d’écritures **writeImagePPM**, **writeImagePGM** et **writeImagePBM** se trouvent dans le fichier **the_sources/writeimg.c.**

Compléments techniques

Makefile

all : compile le fichier main et les librairies en statique et dynamique. Efface tous les fichiers secondaires inutiles (.o et .gch)

Clean_obj : efface tous les fichiers secondaires inutiles (.o et .gch)

clean : efface tous les fichiers non sources (executables, librairies, etc).

Gestion mémoire

Après avoir passé l'exécution de notre programme au crible avec l'utilitaire Valgrind, on peut affirmer que notre programme ne provoque aucune fuite mémoire dans tous ses cas d'utilisations. Nous avons pris soin de bien libérer la mémoire quand cela était nécessaire.

Notice d'utilisation

- Se positionner dans le dossier **Application/the_sources**
- Lancer la commande **make**
- L'exécutable est généré dans le dossier **Application/bin**
- Les bibliothèques sont générées dans le dossier **lib**
- Pour utiliser le programme, il est nécessaire de renseigner l'argument **-g** pour une conversion en *.pgm* ou **-b** pour une conversion au format *.pbm*. Il n'est pas nécessaire de rentrer un nom de fichier au format *.ppm* pour utiliser l'application. Si vous voulez convertir un fichier existant, il suffit de l'ajouter dans la commande après l'argument.