Rapport de projet IPI 2022,ENSIIE - SquIIgglE

Hugo GENEST

8 janvier 2022

Table des matières

1 Présentation du projet

1.1 Contexte du Projet

Le but de ce projet est de créer un programme en C capable de lire une entrée en format IPI et d'écrire un format PPM correspondant en sortie, c'est-à-dire un interpréteur graphique.

Un fichier au format IPI décrit la construction d'une image carrée. Il est composé de deux sections, la première est une ligne contenant la taille de l'image. La seconde est une liste de caractères représentant les instructions pour la génération de l'image. Le fichier IPI est lu par l'interpréteur pour générer l'image au format PPM.

Les instructions du fichier IPI seront interprétées par le programme pour modifier différents objets. Ces objets sont : une coordonnée marquée (X, Y), un curseur défini par une position (X, Y) et une direction (Est, Sud, Ouest, Nord), un seau de couleurs correspondant à un ensemble de couleurs (une couleur est composée de trois composantes R, G et B qui sont des entiers non signés sur 1 octet : `unsigned char`), un seau d'opacités correspondant à un ensemble d'opacités (une opacité est un `unsigned char`) , une toile contenant jusqu'à 10 calques, qui sont chacun une grille de pixels eux même définis par une couleur et une opacité. Les caractères pouvant exécuter des commandes sont '`abcefghijlmnoprstvwy`', et les commandes sont :

- `n` : Rajouter la couleur noire (0,0,0) au seau de couleurs ;
- `r` : Rajouter le rouge (255,0,0) ;
- `g` : Rajouter le vert (0,255,0) ;
- `b` : Rajouter le bleu (0,0,255) ;
- `y` : Rajouter le jaune (255,255,0) ;
- `m` : Rajouter le magenta (255,0,255) ;
- `c` : Rajouter le cyan (0,255,255) ;
- `w` : Rajouter le blanc (255,255,255) ;
- `t` : Rajouter l'opacité transparente 0 au seau d'opacités ;
- `o` : Rajouter l'opacité opaque ;
- `i` : Vider les seaux de couleurs et d'opacités ;
- `v` : Avancer le curseur un pixel dans sa direction ;
- `h` : Tourner le curseur de 90° dans le sens horraire ;
- `a` : Tourner le curseur de 90° dans le sens anti-horraire ;
- `p` : Changer la position marquée aux coordonnées du curseur ;

- `s` : Rajouter un nouveau calque noir transparent à la toile ;
- `l` : Tracer une ligne entre la position marquée et le curseur avec le pixel courant* ;
- `f` : Il s'agît de l'outil habituel 'seau', cela remplit la zone autour du curseur de la même couleur que le pixel situé aux coordonnées du curseur par le pixel courant* ;
- `e` : Fusionner les deux calques du haut de la toile ;
- `j` : Découper le deuxième calque le plus haut de la toile en utilisant celui du sommet comme masque d'opacité.

*pixel courant : Le pixel courant est un pixel avec pour opacité la moyenne des opacités du seau d'opacités, et pour couleur la moyenne du seau de couleurs multipliée par l'opacité calculée divisée par 255.

Tous les autres caractères seront ignorés, y compris les lettres majuscules. Il est donc recommandé d'agrémenter ses fichiers IPI de commentaires en lettres majuscules, afin de rendre le fichier plus lisible pour l'utilisateur. L'ensemble de ces informations sont indiquées dans le `ReadMe.txt` dans le dossier `SquIIgglE`. Un exemple y est également proposé avec les fichiers `logo.ipi` et `logo.ppm`.

1.2 Présentation des Documents

Afin de rendre le contenu du projet plus lisible, il a été divisé en plusieurs documents et dossiers. Il y tout d'abord le dossier principal `SquIIgglE` contenant l'ensemble du projet. Dans ce dossier, il y a le manuel d'utilisation `ReadMe.txt`, les fichiers d'exemple `logo.ipi` et `logo.ppm`, l'exécutable `SquIIgglE.exe`, le fichier `Makefile` et un dossier `Source`. L'ensemble des fichiers de code .c, .h et .o sont situés dans ce dernier fichier.

1.3 Présentation de l'exécutable

Le programme est implémenté dans l'exécutable `SquIIgglE.exe`, généré par `Makefile` à l'aide de la commande `make` exécutée le dossier `SquIIgglE`. Par défaut, le programme lit l'entrée standard, renvoie le contenu type d'un fichier PPM dans la sortie standard. Nous pouvons utiliser deux paramètres optionnels afin de modifier l'entrée et la sortie.

Le premier paramètre possible est `input`, qui permet de changer l'entrée pour le fichier qui est précisé après le mot input : `./SquIIgglE input file.ipi`

Le second paramètre possible est `output`, qui permet de modifier la sortie pour le fichier qui est précisé après le mot output : `./SquIIgglE output file.ppm`
Si ce dernier paramètre est précisé, le programme retournera le titre du fichier de sortie sur la sortie standard.

Le programme est capable de détecter certaines erreurs d'entrées, et les affichera dans le `stderr` :

- `"SquIIgglE Error, wrong number of arguments:%d given"`, apparaît quand le mauvais nombre d'arguments est fournis à l'exécutable ;
- `"SquIIggle Error, unknown parameter %s"`, apparaît quand un des paramètres n'est ni input, ni output ;
- `"SquIIgglE Error, redundant parameter %s"`, apparaît quand les deux paramètres optionnels sont soit tous deux `input` soit tous deux `output`.

2. Implémentation du Programme

2.1 Définition des Objets

Afin d'interpréter le fichier IPI, nous devons implémenter les différents objets cités plus haut (cf p.2) Il a donc d'abord fallu construire les types correspondants à ces objets. Cette implémentation a été réalisée avec l'idée de modularité en tête. Ainsi, chacun des types construits sont dans des modules distincts.

- Le premier type construit est le type couleur ; un enregistrement contenant un `unsigned char` pour chacune de ses composantes R,G et B. Dans ce même module est inclus la définition d'opacité, qui est un allias d'`unsigned char`. Toujours dans une idée de modularité, j'ai décider d'inclure dans ce module les fonctions `char_color` et `char_opacity` qui renvoient respectivement une couleur ou une opacité prédéfinie à partir d'un caractère d'entrée. Ces données prédéfinies pourraient ainsi être utilisés dans d'autres projets.

```c
typedef struct color{
    unsigned char R;
    unsigned char G;
    unsigned char B;
} color;

typedef unsigned char opacity;
```

- Les seaux de couleurs et d'opacités ont chacun un type distinct représenté respectivement par une liste chaînée de couleur et une liste chaînée d'opacité.

```c
typedef struct color_dose{
    color value;
    struct color_dose* next;
} cdose;

typedef cdose* cbucket;
```

```c
typedef struct opacity_dose{
    opacity value;
    struct opacity_dose* next;
} odose;

typedef odose* obucket;
```

- Un pixel est un enregistrement composé d'une couleur et d'une opacité.

```
typedef struct pixel{
    color C;
    opacity O;
} pixel;
```

- Un calque est un enregistrement composé d'un entier représentant sa taille `size`, et d'une grille de pixels de dimensions `size * size`.

```
typedef struct layer {
    int size;
    pixel **grid;
}layer;
```

- Une toile est définie par un entier représentant sa taille, un second représentant le nombre de calques qu'elle contient, puis d'un tableau de taille 10 contenant les différents calques.

```
typedef struct canvas{
    int size;
    int nb_layers;
    layer layers[10];
} canvas;
```

- Le curseur est définie par une coordonnée d'entiers (X, Y), une direction (Est, Sud, Ouest, Nord), puis un entier `lim_coord` représentant les coordonnées que le curseur ne peut pas dépasser.

```
typedef enum direction{East,South,West,North} direction;

typedef struct cursor{
    int x;
    int y;
    int lim_coord;
    direction direc;
} cursor;
```

L'ajout d'éléments aux types `obucket` et `cbucket` font intervenir des allocations dynamiques, tout comme l'initialisation des calques, il est donc important d'implanter des fonctions `free` permettant de libérer cet espace mémoire. Une fonction afin de libérer l'espace mémoire occupé par l'ensemble des calques d'une toile a également été implanté.

Afin de faciliter la correction du programme, différentes fonctions `print` ont été conçues pour l'ensemble de ces objets. De plus, la programmation a été progressive, et chaque nouvel ajout a été testé avant de passer à la suite.

2.2 Lecture du format IPI et Écriture du format PPM

Le programme avait d'abord été implémenté pour lire le contenu d'un fichier au format IPI fourni, et d'écrire dans un fichier au format PPM dont le nom est donné en sortie. Cela a été réalisé à l'aide des fonctions `fopen` et `fclose`, en utilisant les fonctions `fgets`, `fgetc`, `sscanf` pour la lecture, et `fprintf` pour l'écriture.

L'implémentation actuelle est arrivé par la suite, en comptant le nombre d'arguments `argc` et en analysant les différents arguments de `argv`, en prenant en compte les différents cas d'erreurs mentionnés précédemment (cf. p.4). Afin d'analyser les éléments d'`argv`, une fonction pour comparer deux chaînes de caractères a été implémentée.

2.3 Implémentation de la fonction de remplissage

L'implémentation de la fonction `fill()` exécutée par la commande `'f'` a été conçue en plusieurs jets. La première implémentation était une fonction récursive. Seulement, cette manière de faire ne fonctionnait que pour des petites images. Après avoir effectué des tests, cette fonction ne pouvait remplir que des images de taille inférieure à 180 pixels.

Afin de répondre à ce problème, une seconde implémentation, non récursive, a été mise en place. Dans cette implémentation, nous utilisons une liste chaînée contenant les coordonnées des pixels sur lesquels il faut encore travailler. Il a donc fallu créer ce type.

```
typedef struct cell{
    int coord_x;
    int coord_y;
    struct cell *next;
}cell;

typedef cell *list;
```

Pour éviter de faire apparaître plusieurs fois une même coordonnée dans cette liste, l'algorithme vérifie si une coordonnée a déjà été dans la liste avant de la rajouter. Seulement, le test d'appartenance à une liste peut être long. Pour faciliter ce calcul, nous utilisons une grille des coordonnées déjà ajoutées à la liste. Cette grille est de la même dimension que le calque sur lequel nous travaillons, et est initialement remplie de 0 en `char`. Dès que la coordonnée (X, Y) est insérée dans la liste, la coordonnée (X, Y) de la grille devient un 1. Il suffit alors de regarder la coordonnée (X, Y) de la grille pour savoir si cette coordonnée a déjà été incluse dans la liste.

2.4 Limites du Programme

Le programme admet ces limites. D'abord, il est incapable de vérifier la validité des fichiers d'entrée. En effet, `SquIIgglE` ne vérifie pas si le fichier indiqué par le paramètre `input` existe bien. De plus, le programme ne vérifie également pas que l'entrée suit les normes du format IPI, ce travaille est à réaliser par l'utilisateur, qui devra notamment penser à ne pas inscrire de commentaires en amont de la taille, qui doit être seule sur la première ligne.

Ensuite, il est à noter que le format IPI implique la création d'images nécessairement carrées. Le programme lisant en entrée un format IPI, il ne pourra donc renvoyer que des images carrées.

Enfin, le programme n'est compatible qu'avec Linux.

3. Conclusion

Ce projet a été mon premier projet important de C. Cela m'a permis d'approfondir mes compétences dans ce langage. Ayant eu des difficultés à écrire le `Makefile`, il m'a également formé sur l'utilisation de cet outil. Enfin, ce projet m'a permis de développé mes capacités de rédaction de programmes, en faisant attention au nommage des variables, aux commentaires, et à la documentation.