rapport

Valentin Capieu

8 janvier 2024

Contents

1 Introduction

L'objectif de ce projet était de proposer un interpréteur capable de lire un image au format .ppm et de le traduire un algorithme fonctionnel.

Une banque d'exemple était mise à disposition, pour s'assurer du bon fonctionnement du programme une fois celui-ci fini. J'ai le regret d'écrire que mon programme ne fait tourner aucun de ces exemples, malgré le fait qu'il compile sans problème et n'a pas d'erreurs de segmentation.

Je vais donc essayer d'être le plus précis possible dans ce rapport concernant les choix que j'ai pu faire durant la réalisation de ce projet.

J'ai segmentais ce projet en 4 partie, c'est pourquoi vous trouverez dans mon projet 4 headers, qui sont l'intermprétation de l'image, la définition des piles, le déplacement à l'intérieur de l'image, et les actions à effectuer sur la pile.

Concernant le lancement du programme, il contient un Makefile, il suffit donc de saisir *make* puis ./prog exemple.ppm

2 Lecture du fichier .ppm

2.1 Contenu d'un fichier .ppm

Un fichier .ppm est composé de plusieurs éléments. La première ligne est une indication concernant l'image ne elle-même, dans notre cas, "P6" désigne une image au format .ppm en couleur. La seconde information que nous pouvons lire est la taille de l'image, à savoir sa hauteur et sa largeur en terme de pixel. La suite du fichier est l'image en elle-même.

2.2 Traduction de l'image

Cette partie est un complément des fichiers *lecture.c* et *lecture.h*.

La traduction s'effectue en deux étapes, la première est la lecture du fichier .ppm et le stockage de cette lecture à l'aide d'un type image contenant des pixels (repérer par leurs valeurs rgb).

S'en suit une seconde étape, qui vise cette fois à traduire chaqun des pixels en sa "valeur" (nom de mon type) aux yeux du programme ; comme il n'y a que 18 couleurs codantes, il est possible de les identifier en fonction de leur code rgb. Autrement dit, il s'agit pour chaque pixel de référencer sa couleur et sa luminosité s'il est codant, et sa luminance sinon. Cette étape permet de faciliter les futures actions en fonction des déplacements d'un bloc à l'autre, car il suffira alors de faire la différence de couleur et du luminosité entre le bloc de d'arrivé et de départ.

3 Définition du type *pile*

Durant ce projet, j'ai opté pour une définition dynamique des piles, étant donné que je n'avais pas d'information concernant le nombre maximun d'étage possible. Ce type est défini dans les fichiers *stack.c* et *stack.h*.

J'ai implémenté en plus les fonctions classiques concernant les piles, vu en td cette année.

4 Action à effectuer sur la pile

Les fichiers *ordre.c* et *ordre.h* contiennent l'ensemble des actions qu'il est possible d'effectuer sur la pile. Chacune sont codés à partir des fonctions de bases

présentes dans *stack.h*.

Avant chaque action, un test est effectué sur la pile pour savoir si elle contient assez d'éléments pour supporter l'action.

5 Déplacements au sein de l'image

5.1 Explications générales

Cette partie porte exculsivement sur les deux fichiers *lecture.c* et *lecture.h*. Comme leurs noms l'indique, ils ont pour but de lire l'algorithme présent dans l'image.

Ces fichiers sont la clef de voûte de mon projet, car ils contiennent les fonctions permettant de se déplacer dans l'image.

Ce sont aussi ces mêmes fonctions qui n'ont pas l'effet escompté, et qui font que l'entiereté de mon programme est mis en échec. Elles sont commentés de fond en comble pour pouvoir facilité leur suivi. Je vais donc énoncer la logique que j'ai essayé de mettre en oeuvre tout au long de celle-ci.

5.2 Fonctionnement et logique mis en place

Ce fichier propose une méthode en 3 étapes pour effectuer un pas dans l'image.

La première étape et la délimitation du bloc sur lequel on se trouve. La fonction *bloc* assure ce rôle en remplissant dans une liste les coordonnées les pixels faisant parties du bloc.

La deuxième étape est de savoir sur quel pixel du bloc partir en fonction de la direction et du bord. Pour cela, j'ai opté pour une fonction de type recherche d'un éléments dans la liste des coordonnées des pixels du blocs.

Enfin, la troisième étape est le déplacement. Connaissant le pixel duquel partir, et la direction, il faut mainetenant se déplacer dans l'image tout en gardant en mémoire la couleur et la luminance de l'endroit d'où on part. Il faut prendre en compte plusieurs cas, selon si le pixel sur lequel on essaye d'avancer est codant, bloquant ou passant. Cette étape correspond à ma fonction *suivant*.

6 Complications rencontrées et conclusion

J'ai manqué de finesse lorsque j'ai défini mon type *coordonnée* car celui-ci, en réalité, ne correspond pas aux coordonnées abscisse/ordonné du pixel renseigné

mais à sa ligne et à sa colonne dans l'image.

Il faut donc être vigilant pour se déplacer dans une direction, par exemple pour se déplacer vers l'est, il ne faut pas changer x mais la colonne y.

Ce petit malentendu m'a mené à de nombreuses erreurs de segmentation, que j'ai finalement surmonté via un debugger (gdb)

En conclusion, J'ai essayé d'appliquer au mieux des choix qui me paraissait logique et optimal pour parvenir à ce qui est demandé, bien qu'au final mon programme n'est pas fonctionnel