Rapport de projet PRIM

I. introduction

Nous cherchons ici à implémenter en langage C un interpréteur permettant de convertir un fichier **.ipi**, constitué d'instructions originales, en une image sous format **.ppm** qu'il est possible de visionner plus tard par, entre autres, **ImageMagick** ou **emacs**. Plus précisément, il s'agit d'interpréter un ensemble de lettres, correspondant à un set d'instructions agissant sur une machine à états qui crée une image via la manipulation de "calques", ie. de matrices de pixels.

II. choix lors de la programmation

La décomposition en plusieurs fichiers **.h** et un seul fichier **.c** s'inspire de la gestion des fichiers d'un projet personnel précédent en **Python**, où les modules sont séparés du programme qui va être exécuté, qui lui est contenu dans un fichier **main**.

Afin de traiter le fichier **.ipi**, c'est la **libc** qui va être utilisée : après la détermination de la taille de l'image, les lignes sont traitées lettre par lettre et une par une jusqu'à la fin du fichier. L'écriture se fait aussi à l'aide de la **libc**, les données de chaque pixel étant écrites pixel par pixel. Le traitement des lettres se fait en partie grâce à des expressions régulières afin de rendre le code plus lisible : en effet, par exemple, plusieurs lettres correspondent à l'ajout d'une couleur dans le seau de couleurs, seule la couleur diffère en fonction de la lettre.

La spécification indiquait clairement le type de données à employer pour les objets basiques employés : des **unsigned int** pour les composantes ; des **struct** dotés d'attributs adéquats pour les couleurs RGB et les pixels, des **char** pour les directions (correspondant aux quatre points cardinaux). Des libertés ont été cependant prises sur les positions, qui ne sont pas implémentées via deux entiers qu'il faut passer en argument de chaque fonction agissant sur la position du curseur, mais plutôt via une nouvelle structure, **pos**, qui regroupe deux **int x** et **y** afin d'améliorer la lisibilité du code (aurait-il été plus judicieux d'utiliser des **long int** pour les grands canevas ? cf. III).

Toutes les structures utilisant une certaine forme de pile, à savoir les seaux et la pile de calques, sont implémentées en utilisant les listes chaînées. En effet, si un tableau peut suffire pour la pile de calques, de taille finie (10), ce n'est pas le cas des seaux qui eux ne peuvent a priori jamais être pleins. Cela permet en outre d'extraire les calques situés au sommet de la pile de calques afin d'effectuer des opérations avec, plutôt que d'enregistrer la position du sommet. Initialement, les piles de positions étaient également implémentées de façon similaire, mais elles ne sont pas utilisées dans le programme final (cf. III).

Le debugger n'a pas été utilisé lors de l'élaboration du programme. En revanche, une fonction intitulée **_debug** est présente au début de **main.c**, elle peut forcer l'affichage à l'écran de texte afin de déterminer à quel endroit une erreur survient. Une fonction **_spec**, précisant les caractéristiques d'un pixel d'un calque donné, a également servi à l'élaboration de la fonction de remplissage.

III. problèmes et solutions

La fonction de remplissage a été l'une des fonctions les plus difficiles à implémenter efficacement. Dans un premier temps, c'est une pile de positions qui a été utilisée afin de stocker les positions des pixels à remplir, comme le suggérait l'énoncé. Cependant, la complexité de vérifier l'unicité des positions stockées dans cette pile, associée à des erreurs de mémoire diverses (mauvaise utilisation des pointeurs vers le sommet de la pile...) a conduit à privilégier la méthode récursive. Cette méthode, bien que fonctionnelle sur de petits calques (résultat correct avec les images **ocaml** et **ducks**) a de nouveau entraîné des **stack overflow** sur des calques plus grands, comme **base** ou **virus**. Le problème a été résolu... ou plutôt contourné en augmentant la taille de la pile mémoire, via l'appel système **setrlimit**. Ainsi, la création d'images plus grandes ne pose désormais plus de problèmes liés au dépassement de la pile, bien qu'on puisse à partir d'un moment s'interroger sur la pertinence de l'utilisation d'**int**, et pas de **long int**, pour les positions...

La création d'une page **man** pour le programme, bien que peu importante (dans le cadre de ce projet de PRIM bien entendu...) a également été relativement laborieuse : par manque de maîtrise de la commande **groff**, le soin est laissé à l'utilisateur de compiler lui-même le fichier **manpage** disponible dans l'archive du projet (cf. **README**).

IV. conclusion et critique

Le programme n'a pas été testé avec de très grands canevas (taille > 1000). Or, l'augmentation de la pile mémoire via **setrlimit** est limitée par la quantité de RAM disponible sur l'ordinateur. Ainsi, l'exécution de ce programme sur une vieille machine pourrait entraîner des résultats inattendus pour des canevas de taille même modérée (environ 500). Mais même sur des ordinateurs modernes, tenter de créer une image de côté 10000 ou plus pourrait probablement faire planter l'ordinateur, si la fonction récursive de remplissage continue d'effectuer des appels récursifs, remplissant ainsi la pile.