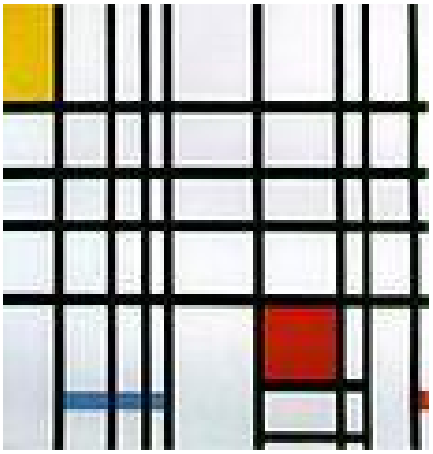
Rapport du projet de programmation impérative

Implémentation d'un interpréteur pour un langage de programmation dont les programmes sont des images

Naël GUERARD



Composition with Yellow, Blue and Red. Piet Mondrian, 1937-1942, huile sur toile, 72,7 × 69,2 cm, Londres,

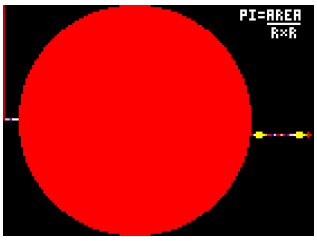
ENSIIE 2023-2024

Sommaire

1. Introduction

1.1. Le langage Cornelis

Le langage Cornelis, fortement inspiré du langage Piet utilise des images d'art abstrait colorées inspirées des tableaux du peintre Piet Mondrian. L'image est parcouru et les différences de couleurs entre les différents blocs de l'image correspondent à des actions à effectuer.



Calcul une valeur approximative de pi

1.2. Le but du programme

Le but de ce projet est d'implémenter un interpréteur pour le langage de programmation Cornelis fortement inspiré du langage Piet. Notre programme prend en argument un fichier PPM, prend en compte ses informations et parcours l'image. A chaque changement de couleur entre 2 blocs, on effectue une action sur une pile initialement vide. Il y a au total 17 actions parmis lesquelles empiler la taille du blocs précédent ou dépiler 2 éléments puis empiler leur somme

1.3 Les fichiers PPM

Le fichier PPM commence par un en-tête qui indique le type de fichier (par exemple P6 pour les images couleur en format binaire), les dimensions de l'image (largeur et hauteur) et la valeur maximale de la couleur. Ensuite, pour chaque pixel on indique le niveau de rouge, vert et bleu.
Par exemple :

```
P6
3 1
255

255  0    0
 0   255  0
 0    0   255
```

2. Le Programme

2.1. Les piles

Pour représenter notre pile d'actions, nous allons utiliser la structure des piles. Nous allons utiliser un type abstrait dans le fichier pile.h et un type concret dans le fichier pile.c a la difference, que le type concret sera déclaré dans le fichier pile.c car nous allons inclure par la suite le fichier pile.h dans le fichier main.c et nous aurons besoin de manipuler la structure struct stack.

```c
#ifndef STACK_H
#define STACK_H

#define STACK_SIZE 1000000

struct stack {          //type concret
    int content[STACK_SIZE];
    int top;
};

typedef void* stack;      //type abstrait
typedef int elem;
```

2.1. Lecture de fichier PPM

L'objectif ici est de récupérer la largeur, la hauteur la couleur maximum et un tableau avec pour chaque case, le niveau de rouge vert et bleu de des pixels de l'image. Pour cela, nous aurons besoin d'un type pixel et d'un type regroupant toutes ces informations.

```c
typedef struct pixel pixel;     //on definit le type pixel
struct pixel {
    unsigned char rouge, vert, bleu;   //valeurs de 0 a 255
};

typedef struct infoPPM infoPPM;   //structure pour les informations d'un fichier PPM
struct infoPPM {
    int hauteur;
    int largeur;
    int couleurMax;
    pixel** pixels;    //tableau 2D de pixels
};
```

2.3. Les couleurs

Ici, nous allons implémenter les fonctions relatives aux couleurs, avec un type énumérant les 18 couleurs codantes et les 2 fonctions les plus importantes pour connaitre la difference de cran de couleur et de luminosité entre 2 couleurs.

2.4. Les blocs

Après avoir défini nos types pour la direction et le bord, une des fonctions les plus importantes du projet sera implémentée ici;

```
typedef enum {    //type pour la direction
    EST,
    SUD,
    OUEST,
    NORD
}direction;

typedef enum {    //type pour le bord
    BABORD,
    TRIBORD
}bord;
```

la fonction traitement sur la base de l'exemple proposé dans le sujet qui permet d'isoler un bloc pour ensuite travailler avec.

```
traitement(i : image, init : couleur, x : colonne, y : ligne, traité : tableau)
  traité[x,y] := 1
  si init = i[x, y]
    ... traitement du pixel en x, y ...
    pour chacun des quatre voisins vx, vy de x,y
      si non traité[vx,vy]
        traitement(i, init, vx, vy, traité)
```

2.5. Module main

Comme dit dans la partie sur les piles, nous allons utiliser un pointeur vers struct stack pour après avoir initialisé notre pile pour pouvoir par exemple stocker la valeur en haut de la pile avec les composantes content et top.

```
stack s = new();
struct stack*r = s;
```

Nous avons également eu besoin des fonctions de la bibliothèque string.h memset pour initialiser notre tableau de zéros avant de remplir avec des 1 la zone de notre

bloc dans la fonction traitement et memcpy pour transférer notre tableau de coordonnées d'un pixel vers un autre de manière simple.

```
int** traite = malloc(largeur*hauteur*sizeof(int));
memset(traite, 0, largeur*hauteur*sizeof(int));

memcpy(p_prev, p_cur, 2*sizeof(int));
```

3. Conclusion

Par rapport aux difficultés que j'ai rencontrées, elles sont survenues lorsque j'ai commencé le module bloc. En effet, à partir de ce moment, il fallait trouver comment représenter les blocs mais surtout gérer comment passer aux blocs suivants. Ne voyant pas forcément comment faire, je suis alors passé au main et j'ai ajouté au fur et à mesure les fonctions dont j'avais besoin dans le module bloc. Par exemple, au début du main, je suis parti du principe qu'il fallait parcourir tous les blocs, alors j'ai eu l'idée d'implémenter une fonction qui compte le nombre de blocs d'une image afin d'utiliser une boucle while pour terminer le programme.

Cependant, mon terminal ne permet pas de compiler le programme en raison de l'utilisation des double slash pour les commentaires.