TP4 : Structures chaînées

Pour ce TP, l'objectif est de manipuler des structures chaînées. Comme prétexte, nous allons nous intéresser à des représentations vectorielles d'images simples et à des manipulations de base. Le TP3 sera utile pour recycler quelques fonctions ou principes.

Une image vectorielle se caractérise par une description géométrique (lignes cercles) indépendante de toute échelle de représentation (par opposition à une image dite *bitmap*, dont la description ne s'intéresse qu'à sa représentation sous forme de pixels). Le format que nous allons utiliser ici n'est pas standard, mais rend bien compte des avantages et inconvénients de ce genre d'image.

1) Environnement de base

Bien que ce ne soit pas strictement nécessaire, il est conseillé de récupérer la définition de `struct color` du TP2. Cela allègera les prototypes de fonctions. Vous aurez également besoin de récupérer la définition de `struct picture` du TP3 ainsi que des fonctions suivantes :

- `new_pic()`
- `save_pic()`
- `draw_line()` (et donc `set_pixel()` dont elle dépend très certainement)

Nous allons travailler à partir de deux images données comme ressources sur la page du projet : `cat.txt` et `kang.txt`. Le format de ces fichiers est le suivant :

- Chaque ligne contient une succession de 4 flottants en représentation ASCii sur 9 caractères chacun.
- Les quatre flottants de chaque ligne représentent respectivement les valeurs x1, y1, x2 et y2 d'un segment à tracer.
- Les 10^{e}, 20^{e} et 30^{e} caractères de chaque ligne sont toujours des espaces (numéros 9, 19 et 29) et servent de séparateurs entre les flottants.
- Chaque ligne a une taille fixe de 40 caractères (en comptant le saut de ligne final).

Les valeurs de chaque point sont exprimées en pixels, mais sont représentées en flottants pour permettre une plus grande souplesse lors de leur manipulation. En effet, comme l'image est codée comme une succession de lignes, il reste possible d'effectuer plusieurs types d'opérations (zoom, rotation, …) sans en altérer la qualité.

Cette première partie permet de vérifier votre bonne compréhension du format de fichier.

Ecrivez un programme qui crée une image de 500x500 pixels et qui trace l'ensemble des lignes de l'image `cat.txt` en utilisant au maximum les fonctions du TP3. Le fichier à lire obéit à un format précis, il est donc possible d'utiliser fscanf pour s'éviter des manipulations trop complexes.

2) Mémorisation en structure chaînée

Nous allons maintenant nous intéresser à une série de fonctions permettant d'effectuer diverses transformations. Pour appliquer ces transformations, il est nécessaire de mémoriser les coordonnées

de chaque ligne. Nous allons le faire dans une structure chainée pour garder un maximum de souplesse.

Une structure chainée est une structure de donnée constituée d'éléments distincts mais dont chacun contient un pointeur vers un élement suivant. Toute fonction ayant accès au premier élément peut donc parcourir l'ensemble des éléments en suivant les références successives. L'organisation constituée par ces éléments chaînés est appelé *liste*[1]. Elle est représentée figure 1.

Déclarez un type `struct vector`, qui contient quatre champs de type `double` (x1, y1, x2, y2) ainsi qu'un pointeur vers un `struct vector` (qui s'appellera next). Chaque élément `struct vector` décrit un segment à tracer. L'organisation constituée par l'ensemble des `struct vector` liés un à un constituera la figure complète.

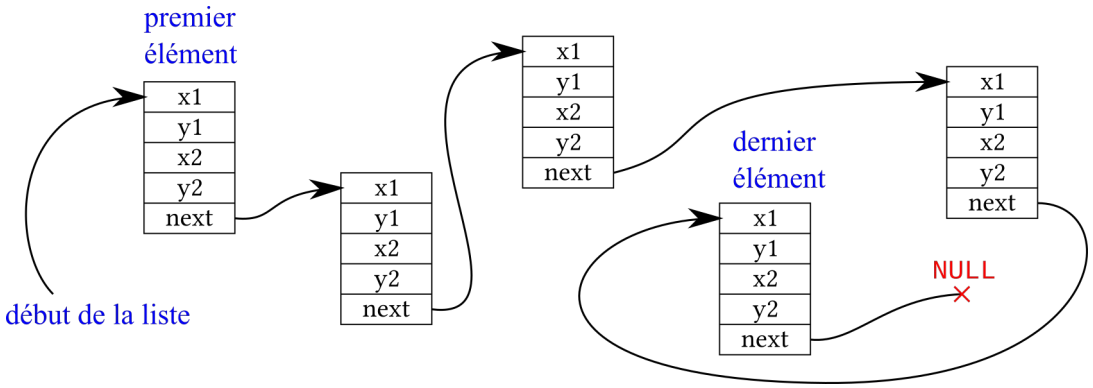

Figure 1: représentation d'une liste de 5 lignes

Ecrivez une fonction `read_vector_file()` qui reçoit un nom de fichier sous forme de chaîne de caractères, et qui renvoie un pointeur vers un `struct vector`. Pour créer la liste des segments, voici l'algorithme le moins pénible (également illustré figure 2):

- créer deux pointeurs : un destiné à référencer le premier élément (initialisé à `NULL` : pas de premier élément), l'autre référencer temporairement chaque nouvel élément (2.a)
- pour chaque élément à ajouter :
  - réserver la place pour créer un nouvel élément (2.b)
  - remplir les champs classiques du nouvel élément
  - faire pointer le champ next vers l'ancien premier élément (qui devient le deuxième) (2.c)
  - copier l'adresse du nouveau premier élément dans le pointeur destiné à cet effet (2.d).
- Retourner le pointeur vers le premier élément.

Le prix de la simplicité de cet algorithme est que la liste est créée à l'envers. Ce n'est pas important pour notre application puisqu'il n'y a pas de relation d'ordre entre les différents segments d'une figure.

Ecrivez une fonction `draw_vector()` qui reçoit un pointeur vers un `struct vector` (une chaine de `struct vector` donc), un `struct picture` et une couleur, et qui trace dans l'image et avec la couleur fournie, la figure constituée des segments de la chaîne.

---

1 On retrouve (enfin) la correspondance avec le langage python. En effet, en Python, l'objet délimité par les crochets [ ] n'est pas implémenté comme un tableau mais comme une structure chaînée. C'est ce choix d'implémentation qui donne à Python sa grande souplesse, mais qui réduit les performances du langage. En effet, là où, dans un tableau en C, il suffit de faire une multiplication pour accéder à l'élément n, dans une liste en Python, la machine doit parcourir les n-1 éléments qui précèdent.

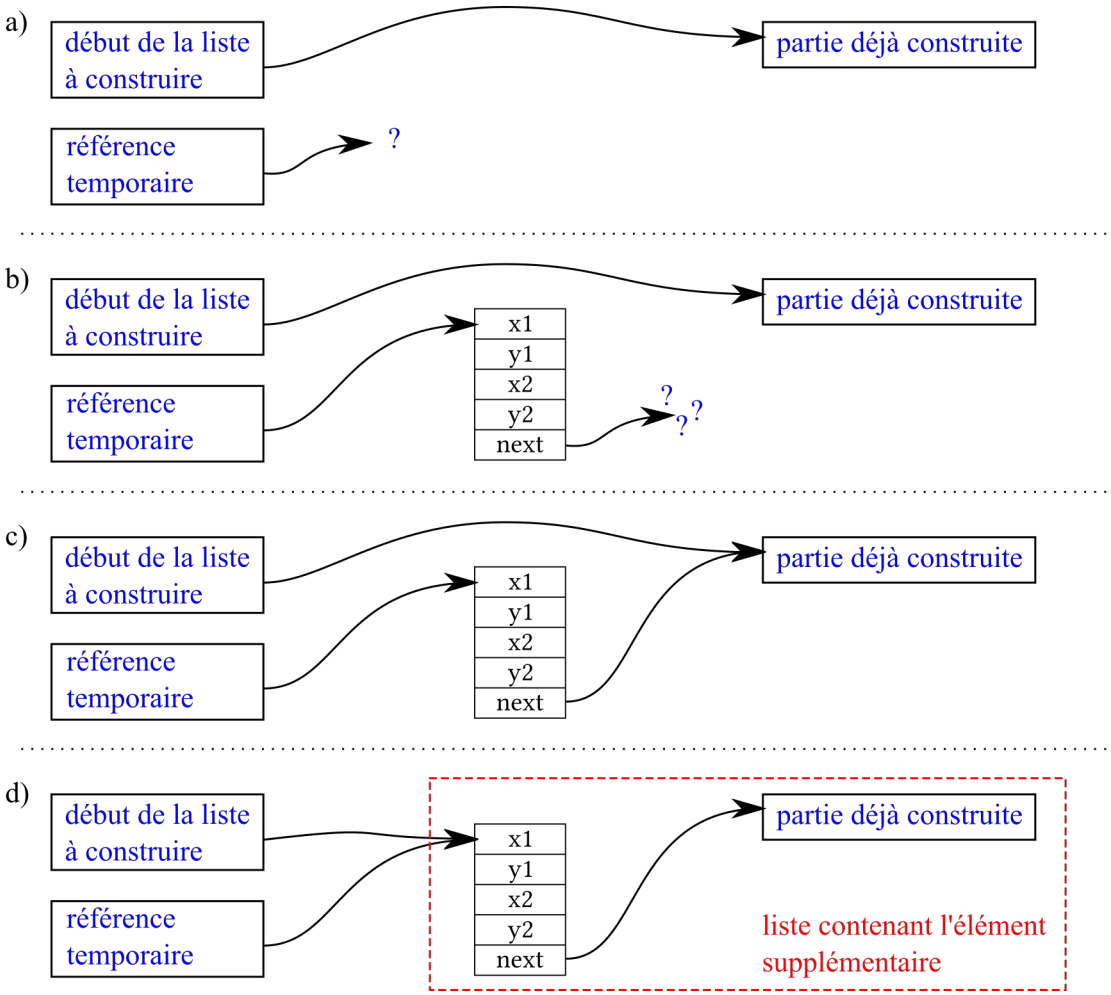

Figure 2: étapes d'insertion d'un nouvel élément en début de liste. (a) Situation de départ ; (b) création du nouvel élément ; (c) enchainement de la partie existante à la suite du premier élément ; (d) référencement du nouveau premier élément

Testez vos fonctions avec un programme qui crée une image à partir de `cat.txt`. A ce stade, vous devriez obtenir la même image qu'à la partie 1. L'avancée semble faible, mais cette structure va nous permettre de jouer un peu avec la figure...

3) Transformation de la figure

Ecrivez une fonction `scale_vector()` qui reçoit un `struct vector *` et un `double` (scale), et qui multiplie l'échelle de la figure par *scale*. Cette mise à l'échelle se fera sans considération de point de référence. On peut donc se contenter de multiplier chaque coordonnée de chaque segment par *scale*.

Testez la fonction `scale_vector()` en modifiant la taille de la figure `cat.txt`.

Ecrivez une fonction `shift_vector()` qui reçoit un `struct vector *` et deux `double` (x,y), et qui déplace la figure selon le vecteur x,y (dit plus simplement, les coordonnées x,y sont ajoutées à chaque point de coordonnée dans la figure).

Testez la fonction `shift_vector()`.

Ecrivez une fonction `flip_vector()` qui reçoit un `struct vector *` et qui applique un effet miroir horizontal. Encore une fois, nous ne prendrons pas de considérations sur la position de l'image, il est donc suffisant de changer le signe des coordonnées x de chaque point.

Testez les fonctions précédentes en affichant deux chats qui se font face l'un l'autre, chacun ayant la moitié de la taille de la figure originale.

Vous pouvez maintenant utiliser la figure kang.txt qui reprend le logo du concours kangourou des mathématiques. Ce logo est un pavage. Il devrait donc occuper tout l'écran. Le symbole de base est donnée dans kang.txt et occupe les dimensions de 100x100.

- Créez la ligne de départ en affichant quatre fois la figure de base aux coordonnées (0, 0), (100, 0), (200, 0) et (300, 0)
- créez une deuxième ligne identique aux coordonnées suivantes (0, 200), (100, 200), (200, 200) et (300, 200)
- créez la ligne intermédiaire en calculant le miroir de la figure de base (qui sera alors positionné en (-100, 0), puis en l'affichant aux coordonnées (50, 100), (150, 100) et (250, 100)

Vous pouvez bien sûr étendre le pavage comme bon vous semble...

4) (Optionnel) Gestion des copies...

A la fin de la partie 3, vous aurez remarqué qu'il serait très pratique de créer une copie de la figure au lieu de toujours modifier l'original. Ecrivez une fonction `duplicate_vector()` qui reçoit un `struct vector *`, et qui crée une copie de la figure en renvoyant un pointeur vers le premier élément de la copie.

Testez votre programme en créant une nouvelle version du programme créant le pavage kangourou, mais utilisant un algorithme plus simple.

Si on peut créer une copie d'une figure, il faut aussi pouvoir effacer une copie pour éviter les fuites mémoires. Ecrivez la fonction `free_vector()` qui reçoit un `struct vector *`, et qui détruit la figure (en libérant la mémoire qu'elle utilise).

5) (Optionnel) Création d'une bibliothèque de fonctions...

Pour pouvoir réellement manipuler les figures, les fonctions suivantes sont nécessaires :

- repérage des bornes inférieure et supérieure en x et en y de la figure (de préférence en un seul passage pour augmenter les performances)
- mise à l'échelle en gardant un point de référence (coin supérieur gauche, ou centre de l'image)
- transformation miroir en utilisant une abscisse de référence,
- rotation d'un angle donné par rapport à un point donné (cette fonction nécessitera l'utilisation de math.h pour accéder aux fonctions de trigonométrie)