Info 316, parcours MATHINFO, Université PARIS 11

1 Introduction

Le but du module "Travail d'Etude et de Recherche" est de vous familiariser avec les pointeurs, la récursivité, les modes de passages, et de vous donner de bonnes habitudes et de bons réflexes de programmation.

A la fin du semestre, vous enverrez votre code, ainsi qu'un petit rapport.

Il pourra y avoir un examen apres la fin du TER, auquel vous ne pourrez réussir que si vous avez travaillé ce dernier. Il y sera demandé de donner le code de quelques nouvelles fonctions qui seront proches des fonctions que vous aurez codées lors du TER.

Le projet s'effectue en monôme.

Régulièrement, vous m'enverrez l'état courant de votre code. Je regarderai le travail effectué depuis le dernier envoi. Je critiquerai, conseillerai, discuterai, ferai refaire, répondrai aux questions ou donnerai des indications...

Je ferai sans doute beaucoup refaire. Notez qu'il faut du temps pour reprendre le travail à refaire et pour réfléchir aux questions les plus délicates. Il vous est plus que vivement conseillé de ne pas vous y prendre au dernier moment et de m'envoyer régulièrement vos travaux. Ceux qui ne l'ont pas fait les années précédentes n'ont pas réussi à rendre un rapport final correct.

1.1 Rapport

Le rapport contiendra :

() Le code complet avec toutes les fonctions implémentées,

() Un rapport texte dans lequel vous parlerez des fonctionnalités affichage-pixel et nébuleuse.

- Vous en expliquerez le fonctionnement et l'implémentation. Vous chercherez à avoir une explication de qualité. L'explication est de qualité quand elle est facile a comprendre Elle n'est pas nécessairement longue.

- vous donnerez dans la mesure du possible des considérations de complexité.

- vous comparerez avec d'autres algos que vous avez envisagés et justifierez vos choix

- vous donnerez enfin un jeu de tests. Un bon jeu de test passe par toutes les lignes de votre code, et explore tous les cas types de figure que vous avez été amenés ou que l'on aurait pu être amené à distinguer. Il doit convaincre une personne qui voit les tests mais pas le code que votre programme est implantable. Les tests doivent donc être variés et exhaustifs : Vérification des situations élémentaires (cas de base entre autres) et des situations complexes, des cas particuliers et de leurs contraires. Un bon jeu de test est effectué par quelqu'un de malin et retors dont le but est de planter votre programme (chargé de TER, concurrent qui veut faire planter votre programme devant le client).

() Si vous faites des parties optionnelles, signalez ces parties, dites ce qu'elles font (lecture améliorée surtout), et si possible expliquez leur fonctionnement.

1.2 Note du projet

Les principaux critères qui sont pris en compte dans la note finale sont

() La qualité du code : qualité de programmation (votre code est beau, propre, lisible (même par quelqu'un qui ne connait pas bien C et n'a pas pensé à votre algo), commenté) et qualité algorithmique (bonnes complexités entre autres). C'est le code final qui est pris en compte, et non les versions intermédiaires qui auront été envoyées en cours de semestre.

() La qualité du rapport : Explication claire, Calcul de complexité, Comparaison avec d'autres algos, Bons tests. Un bon rapport est clair, pédagogue, pertinent, convaincaint, agréable et facile à lire

() L'examen.

2 Programmes et environnement de programmation

Le langage à utiliser pour le projet est le *C*. Nous vous recommandons de vous munir d'un livre de programmation C. Sachez prendre de la distance par rapport à ces livres, ils sont en général faits pour être exhaustifs, pas pour inciter à une programmation propre. Ils sont en général pousse-au-crime.

Un certain nombre de remarques concernant les méthodes de programmation sont à prendre en considération (la liste n'est pas exhaustive) :

- programmer de façon modulaire : en particulier bien structurer les programmes (une procédure ou une fonction ne devrait pas dépasser la taille de visualisation d'un écran), rassembler dans un même fichier les procédures et fonctions qui ont des objectifs voisins, éviter la duplication du code pour une même fonctionnalité, et réduire au strict minimum l'utilisation des variables globales !
- distinguer entre procédure et fonction, faire attention au choix des boucles, choisir avec soin les identificateurs, etc.
- soigner la présentation : utiliser des règles de présentation uniques pour tous les programmes (indentation, minuscules/majuscules, ...), mettre des commentaires pertinents, éviter les programmes lourds (une instruction par ligne) et de manière générale faire attention à tout ce qui peut améliorer la lisibilité des programmes.

3 Enoncé du Projet

Note : on suppose que tout programme contient dans l'en-tête :

```
#define ISNOT !=
#define NOT !
#define AND &&
#define OR ||

typedef enum {
    false,
    true
} bool;
```

3.1

L'objet du TER est de manipuler des images en noir et blanc. Pour cela on considère qu'une image est

- soit blanche
- soit noire
- soit se décompose en 4 sous-images, respectivement haut-gauche, haut-droite, bas-gauche, bas-droite

On représentera ces images avec la structure suivante :

```
typedef struct tmp_image
        { bool toutnoir ;
          struct tmp_image * fils[4] ;
        } r_image ;
typedef r_image *image ;
```

Quand le pointeur est `NULL`, l'image est blanche.

Quand il pointe vers un record dont le champ `toutnoir` est `true`, l'image est noire et les champs `fils[0], fils[1], fils[2], fils[3]` sont sans signification. Il est conseillé mais non obligatoire de les mettre a NULL.

Quand il pointe vers un record dont le champ `toutnoir` est `false`, l'image est obtenue en decoupant l'image en 4, et en placant respectivement les images `fils[0], fils[1], fils[2], fils[3]` en haut à gauche, en haut à droite, en bas à gauche, en bas à droite.

3.2 Entrées Sorties

On utilisera la notation préfixe pour les entrée sortie aux claviers. La notation préfixe consiste à écrire

- B pour une image blanche
- N pour une image noire
- $.x_1x_2x_3x_4$ pour une image décomposée, avec x_1, x_2, x_3, x_4 les notations pour les sous images respectivement haut-gauche, haut-droite, bas-gauche, bas-droite.

Par exemple, l'image ..BBBN.BBNB.BNBB.NBBB est un carré noir au centre de l'image.

Les caractères autres que . B N sont sans signification et doivent donc être ignorés à la lecture. Ils peuvent servir (notamment le blanc, le retour chariot, les parenthèses) à améliorer la lisibilité des affichages.

Le mode simple affiche une image en écriture préfixe.

En mode profondeur, le degré de profondeur est donné après chaque symbole. Par exemple,
. N .BBNB B .N.NNB.NBNNBN sera affiché comme :
.0 N1 .1 B2 B2 N2 B2 B1 .1 N2 .2 N3 N3 B3 .3 N4 B4 N4 N4 B2 N2

En mode 2^k-pixel, l'affichage se fait sur 2^k lignes et 2^k colonnes, en utilisant le point pour le blanc, le $-$ pour le gris et le 8 pour le noir.

L'affichage 2^3-pixel de . N .BBNB B .N.NNB.NBNNBN donne :

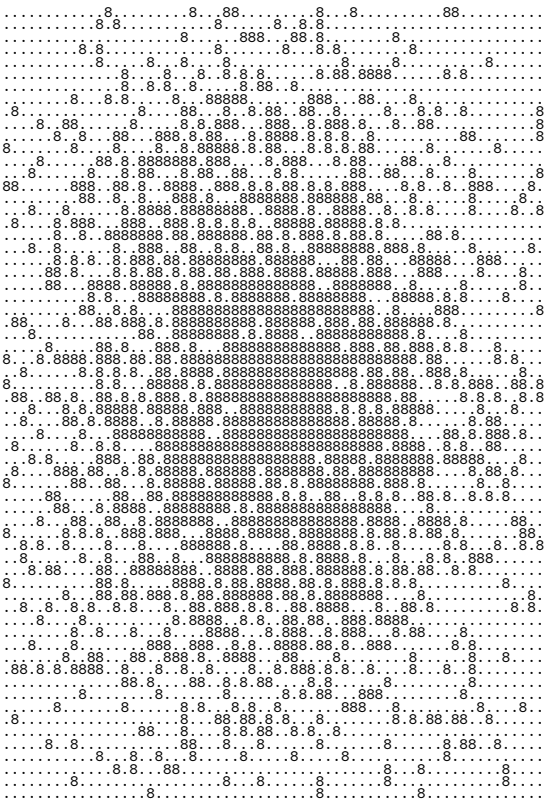
```
8888....
8888....
888888..
888888..
....8888
....88.-
......88
......88
```

3.3 Fonctions de bases à écrire (partie 1)

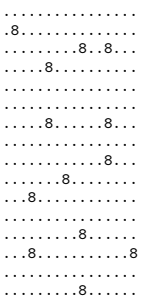
Ecrire les fonctions et procédures (elles ne sont pas données par ordre croissant de difficulte) :

- de construction d'images (construit_blanc() et construit_noir() rendent une image blanche, resp. noire à partir de rien, construit_composee(i1,i2,i3,i4) construit un image composée dont les 4 sous-images sont i1,i2,i3,i4)
- de lecture au clavier (et - optionnel - dans un fichier)
- d'affichage (modes normal, profondeur, 2^k pixel). Note : il est interdit d'utiliser un tableau de taille variable, ou tout autre utilitaire évolué (bibliothèque d'affichage graphique) pour l'affichage pixel.
- est-noir et est-blanche qui testent si l'image est noire, resp. blanche
- meme_dessin qui teste si deux images représentent le même dessin.
- simplifie qui simplifie la représentation d'une image (simplifie(. . N N .NNNN N .BBBB N .BNBN) TRANSFORME l'image en .NBN.BNBN)
- copie
- négatif (procédure qui TRANSFORME une image en son négatif)
- difference qui prend deux images et rend une image qui est noire exactement aux pixels où les 2 images sont de couleurs différentes
- aire (de la zone noire en considérant que le carre est de côté 1)
- rendmemoire qui rend tous les champs d'une image à la mémoire
- arrondit qui prend une image et un entier k et arrondit aux pixels de taille $1/2^k$ (arrondit(image,0) transforme une image en blanc ou noir, arrondit(. N .BBNB B .N[.NNB(.NBNN)]BN,2) arrondit l'image en . N .BBNB B .NNBN)

- Donnez une fonction nebuleuse qui prend k en argument et rend une image de profondeur k qui ressemble à une nébuleuse : la couleur de chaque pixel est tirée aléatoirement de telle manière que la densité de noir varie de quasiment 1 au centre à quasiment 0 aux angles. Par exemple, nebuleuse(6) pourrait donner



- Alea qui prend en argument la profondeur k, et un entier n et qui rendra une image dont la partie noire sera consistuée de n pixels noirs à profondeur k, positionnés aléatoirement. Par exemple, Alea(1,2) rendra avec équiprobabilité l'une des 6 images .NNBB .BBNN .NBNB .BNBN .BNNB .NBBN, tandis que Alea(4,13) pourrait rendre



- -optionnel- menu (ne perdez pas de temps à faire un menu sophistiqué, cela vous rapportera moins de points que de faire de la belle algo sur les fonctionnalités avancées du sujet)

3.4 Amélioration optionnelle de la lecture

Lors de la lecture, les caractères autres que le point, le N et le B majuscules sont ignorés.

Vous pouvez donner des messages d'erreurs lors de lecture dans un fichier (fichier contenant .BNN : image incomplète). un fichier contenant .BNNNN n'est pas une erreur, il contient deux images .BNNN puis N ; Si vous voulez déceler une erreur au cas où une partie est en trop, vous pouvez imposer l'apparition d'un ";" en fin d'image, donc .BNNNN; sera une erreur, mais pas .BNNN;N; Ne pas utiliser le newline comme symbole de fin de lecture car pour la lisbilité, une image doit pouvoir être donnée sur plusieurs lignes.

Vous pouvez dire combien d'images supplémentaires doivent être entrées pour finir l'image à la vue d'un ? :
. .BBNB B .BB ? : il me faut encore 3 images (en effet, 2 à profondeur 2 pour compléter le .BB et une à profondeur 1 pour finir l'image totale)

Vous pouvez faire des vérifications de parenthèsage et afficher des warnings:
.(.(.NBNB)BBB))NNN : warning (une parenthèse fermée en trop)
.(.NBNB]NNN : warning (la parenthèse ouvrante ronde est fermée par une parenthèse carrée)
.(.NNN)BNBN : warning (le contenu de la parenthèse ne correspond pas à une image)
.[(.NNBN])BBB : warning (mauvais usage du rond et du carré)

Vous pouvez faire ignorer les . B et N en commentaires : . .BN (* je peux écrire BN. ici, ca ne fait rien *) NB BBB : revient à . .BNNB BBB

...

3.5 Fonctionnalité optionnelle : Zoom

Les coordonnées d'un point sont données par son abscisse et son ordonnée, que l'on supposera être de la forme $n/2^k$ (On acceptera donc 5/8 mais pas 1/3). Le point inférieur gauche a donc pour coordonnées $(0, 0)$, le centre de l'image a pour coordonnées $(1/2, 1/2)$, etc.

On représentera ces nombres par un record contenant n et k (ce qui évitera les problèmes d'approximation sur les réels)

La fonction Zoom prend en argument une image, une abscisse x et une ordonnée y, et rend la sous-image obtenue sur un carré de taille $1/2 * 1/2$ dont le point inférieur gauche est en (x, y). Par exemple Zoom(.NBBN , 3/8, 1/4) donne . .NBNB B .BNBN N .Ce qui déborde de l'image sera blanc, Par exemple Zoom(N , 1/4. 7/8) donne . B B .BBNN .BBNB

Expliquez comment on peut implémenter cette fonction.

Implémentez-la

3.6 Autres fonctionnalités optionnelles

Ceux qui ont bien avancé et veulent se voir proposer d'autre fonctions subtilesà écrire peuvent me demander des suggestions.