Projet de *Logique et Circuits*

Dates et modalités

Ce projet est composé de deux problèmes : les immeubles et les imarbres. Il doit être résolu par groupes de deux personnes. Le calendrier des rendus demandés est le suivant :

- *Pour les immeubles :* le 22 avril au secrétariat du Deug MIAS (bureau 6 de la Maison de la Pédagogie). Les horaires d'ouverture du secrétariat sont de 9h30 à 12h30 et de 14h à 17h tous les jours (si le bureau était fermé, essayer dans le bureau voisin).

  Vous devez fournir à cette occasion le rapport complet et les sources OCAML.

  Tout retard sera sévèrement sanctionné.

- *Pour les imarbres :* la semaine du 12 mai au moment de la soutenance. Cette soutenance portera sur les deux problèmes du projet et aura lieu pendant votre séance habituelle de TP. Les chargés de TP communiqueront aux étudiants les heures de passage de chaque groupe.

  Vous devez fournir à cette occasion les deux rapports finaux complets et les deux sources OCAML.

  Aucun retard n'est possible.

Il sera accordée la plus grande attention aux points suivants :
- *Spécification :* chaque fonction sera accompagnée de son type et d'une phrase en français explicitant son rôle et la nature de ses arguments.
- *Réalisation :* correction algorithmique, considération de cas exceptionnels, choix des opérations.
- *Codage :* lisibilité du code, notamment choix pertinents de noms, indentation, commentaires.
- *Jeux de tests :* cas exceptionnels, cas quelconques mais pas triviaux.
- *Rapport :* bonne structuration et présentation du problème.
- *Soutenance :* chaque étudiant devra être bien familiarisé avec son programme et il devra être capable de le faire tourner sur un jeu de test arbitraire fournit par son enseignant.

1 Partie I : les immeubles

Le but de cet exercice est de calculer et d'afficher la silhouette (le profil vu de loin) d'un ensemble d'immeubles qui se trouvent dans plusieurs rues parallèles.

1.1 Description du problème

1.1.1 Les rues.

On considère plusieurs rues parallèles indifférenciées qui comportent 100 numéros pairs chacune : du 2 au 200. On ne s'intéresse pas ici aux numéros impairs. Par arrêté municipal, les immeubles ont au maximum une hauteur de 23 étages.

1.1.2 Un immeuble.

Un immeuble est représenté par son adresse et sa hauteur. Ainsi, par exemple un immeuble ira du numéro 36 au numéro 42 et sera de hauteur 4.
On représente un immeuble par une expression de type `immeuble`.

```
type immeuble = Immeuble of adresse * hauteur
  and adresse = Adresse  of int * int
  and hauteur = Hauteur  of int
;;
```

Un ensemble d'immeubles (dessinés figure 1) est représenté par une liste d'expressions de type `immeuble`.

1.1.3 Un profil.

Un profil est la ligne d'horizon enveloppant les immeubles représentée sur la figure 2.

1.2 But du problème

On devra fournir :

1. Des fonctions de construction et d'accès pour le type abstrait `immeuble`. La fonction de construction devra vérifier que l'immeuble créé est correct : la hauteur est valide et son adresse est constituée de deux numéros valides dans l'ordre croissant.
2. Une fonction `genere_immeubles` capable de générer aléatoirement une liste d'immeubles.
3. Une fonction `affiche_immeubles` capable d'afficher une liste d'immeubles.
4. Une structure de donnée (de type `profil`) pour modéliser un profil.
5. Une fonction `calcule_profil` capable de calculer le profil d'un ensemble d'immeubles.
6. Une fonction `affiche_profil` capable d'afficher un profil.
7. Une fonction `teste_programme` capable de
   - afficher une liste d'immeubles,
   - calculer son profil,
   - puis l'afficher.

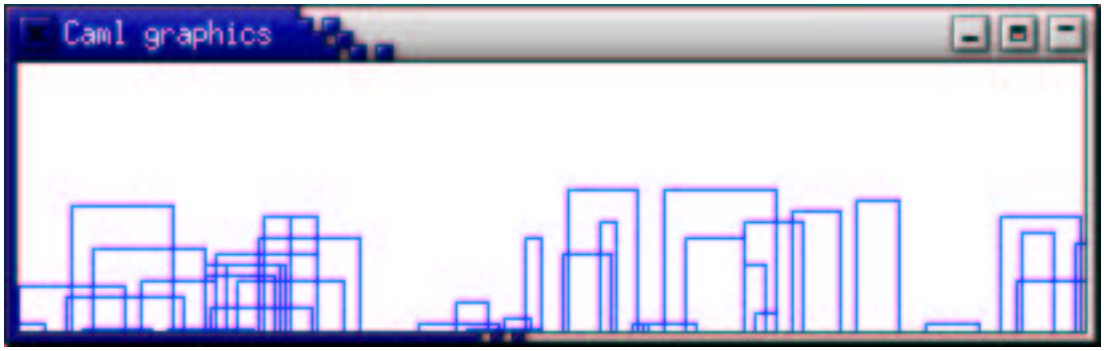


FIG. 1 – Immeubles

FIG. 2 – Profil

8. Au moins cinq jeux de test bien choisis par fonction.
9. Un rapport contenant quelques lignes d'explication pour chaque élément à fournir y compris les jeux de test et des copies de la fenêtre graphique (on doit pouvoir utiliser la commande `ksnapshot` dans un terminal).

1.3 Quelques éléments de Ocaml

1.3.1 Le tri

- La fonction `List.sort compare l` trie la liste `l` avec la fonction `compare`. L'instruction `compare x y` doit renvoyer un entier (strictement positif si `x` est plus grand que `y`, strictement négatif si `x` est plus petit que `y` et 0 sinon).

1.3.2 La génération aléatoire

- L'instruction `Random.self_init();;` initialise le générateur aléatoire.
- L'instruction `Random.int nombre` génère un nombre entier au hasard entre 0 et `nombre-1`.

1.3.3 La partie graphique

La partie graphique est prise en charge par Ocaml :
- Pour créer un toplevel graphique, on peut utiliser la commande

`ocamlmktop -o topGraphics graphics.cma`
On le lancera dans un terminal avec la commande `./topGraphics`.
- Dans ce toplevel, l'expression `Graphics.open_graph " 400x100";;` ouvrira une fenêtre de taille 400*100 pixels. Attention, l'espace dans la chaîne de caractères est important.
- Quelques instructions utiles :
  - `Graphics.moveto : int -> int -> unit` pour déplacer le curseur à une position donnée. Par exemple, `Graphics.moveto 42 12` déplace le curseur à la position (42,12).
  - `Graphics.lineto : int -> int -> unit` pour tracer une ligne à partir de la position courante vers une position donnée.
  - `Graphics.set_color : color -> unit` pour changer la couleur (il y a `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan` et `magenta` accessibles par `Graphics.black`...).
  - `Graphics.read_key();;` pour attendre la pression d'une touche au clavier.
  - Il y a d'autres instructions dans le module `Graphics` mais celles-ci devraient suffire.

2 Partie II : les imarbres

Le but de cette section est de définir une structure de donnée récursive de manipulation d'image ainsi que des primitives adaptées.

2.1 Les imarbres

L'idée générale est de décrire récursivement une image comme
- ou bien une image toute blanche
- ou bien une image toute noire
- ou bien une image constituée de quatre parties (haut-gauche, haut-droit, bas-gauche et bas-droit), chaque partie étant elle même composée de quatre parties ou d'une image blanche ou noire...

Pour simplifier, les tailles des images seront des puissances de 2 (1,2,4,8,16,32,64,128,256...).
On devra fournir :

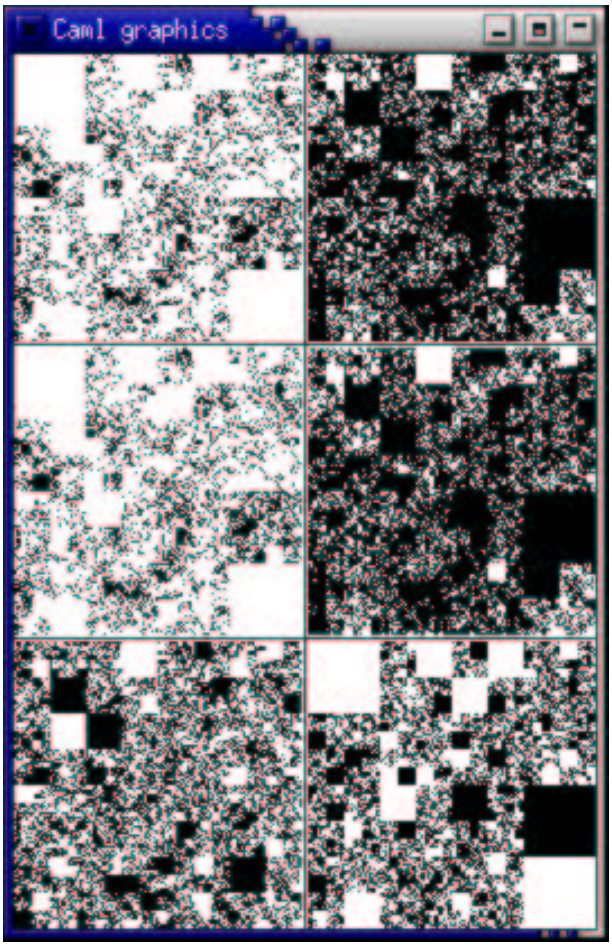
1. La définition d'un type abstrait `imarbre` pour représenter les images. On fournira les fonctions de construction, accès et test. Cette définition sera arborescente, les *feuilles* représentant les images monochromes et les *noeuds* les images constituées de quatre sous-images.
2. Une fonction `random_imarbre n` qui crée une image aléatoire de taille `n*n`. Pour des raisons esthétiques, à chaque noeud, on pourra mettre directement blanc ou noir avec une certaine probabilité.
3. Une fonction `affiche_imarbre image n x y` qui affiche une `image` de taille `n*n` à partir de la position `(x,y)`.
4. Une fonction `simplifie_imarbre image` qui simplifie la représentation d'une `image` de telle sorte qu'aucun noeud ne contienne quatre fils de la même couleur.
5. Une fonction `taille_memoire image` qui calcule le nombre de noeuds (et feuilles) dans l'arbre représentant une image. Ceci permet de tester la fonction `simplifie_imarbre`.
6. Une fonction `intersection image1 image2` calculant l'intersection de deux images (noir et noir donne noir et sinon blanc).

7. Une fonction `union image1 image2` calculant l'union de deux images (blanc et blanc donne blanc et sinon noir).
8. Une fonction `rotation image` qui tourne une image d'un quart de tour dans le sens direct (inverse des aiguilles d'une montre).
9. Une fonction `coupe image profondeur` qui coupe l'arbre représentant l'image à une certaine profondeur.
10. Une fonction `affiche_imarbre_coupe image n x y` qui affiche une `image` de taille `n*n` aux coordonnées `(x,y)` après l'avoir coupée à la profondeur $\log_2 n$.
11. [Question subsidiaire] Une fonction `fractale cote image n imageB imageN` qui crée une nouvelle image dont la longueur du coté est `cote` en itérant `n` fois la transformation suivante dans image :
    - remplacer les carrés blancs par `imageB`
    - remplacer les carrés noirs par `imageN`
12. Au moins cinq jeux de test bien choisis par fonction.
13. Un rapport contenant quelques lignes d'explication pour chaque élément à fournir y compris les jeux de test.

2.2 Exemples

- Les fonctions `intersection`, `union` et `simplifie_imarbre`.

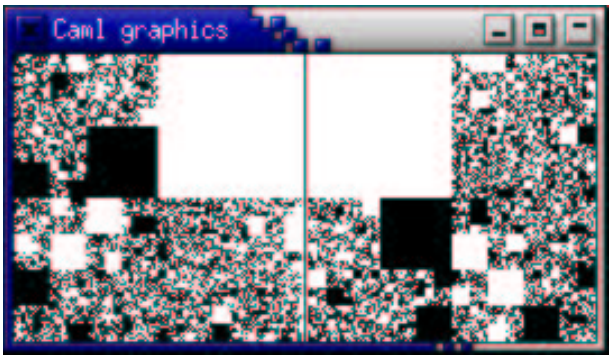
```
Graphics.close_graph();
Graphics.open_graph " 259x390";
let i1=random_imarbre 128 in
let i2=random_imarbre 128 in
let i3=intersection i1 i2 in
let i4=union i1 i2 in
let i5=simplifie_imarbre i3 in
let i6=simplifie_imarbre i4 in
affiche_imarbre i1 128 0 0;
affiche_imarbre i2 128 131 0;
affiche_imarbre i3 128 0 131;
affiche_imarbre i4 128 131 131;
affiche_imarbre i5 128 0 262;
affiche_imarbre i6 128 131 262;
Graphics.set_color Graphics.black;
Graphics.moveto 0 129;
Graphics.lineto 259 129;
Graphics.moveto 0 260;
Graphics.lineto 259 260;
Graphics.moveto 129 0;
Graphics.lineto 129 390
;;
```



La disposition des images sur la figure de droite est :

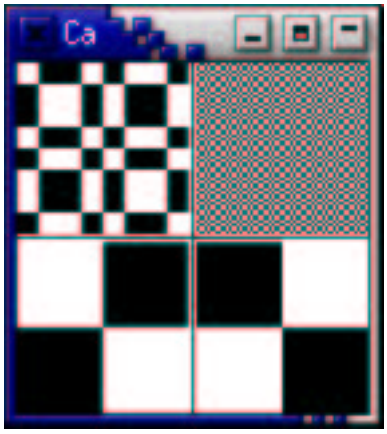
i5	i6
i3	i4
i1	i2

– La fonction `rotation`.




```
Graphics.close_graph();
Graphics.open_graph " 256x128";
let i1=random_imarbre 128 in
let i2=rotation i1 in
affiche_imarbre i1 128 0 0;
affiche_imarbre i2 128 128 0
```

– Une fractale simple




```
Graphics.close_graph();
Graphics.open_graph " 131x131";
let i1=Multicolor(Blanc,Noir,Noir,Blanc)in
let i2=Multicolor(Noir,Blanc,Blanc,Noir)in
let i3=fractale 64 Blanc 5 i1 i2 in
let i4=fractale 64 Blanc 20 i1 i2 in
affiche_imarbre i1 64 0 0;
affiche_imarbre i2 64 67 0;
affiche_imarbre i3 64 0 67;
affiche_imarbre i4 64 67 67;
Graphics.set_color Graphics.black;
Graphics.moveto 0 65;
Graphics.lineto 130 65;
Graphics.moveto 65 0;
Graphics.lineto 65 130;
;;
```

La disposition des images sur la figure de droite est :

i3	i4
i1	i2

2.3 Ocaml

– `Graphics.fill_rect x y l1 l2` dessine rectangle à la position x, y de coté l_1xl_2.