Cours Programmation
MIM 1, 2000–2001, Devoir 1

Luc Bougé, Alice Bonhomme

29 septembre 2000

1 Présentation du devoir

Ce devoir est à faire par groupes de 2. Il est à rendre pour le

vendredi 13 octobre

Vous devez rendre pour chaque groupe :

- une description de la méthode de programmation utilisée ;
- les listings des programmes ;
- une copie d'écran des sorties (utiliser le programme snapshot du menu Workspace/ Programs du fond d'écran).

Au cours du TP suivant, chaque groupe devra faire la démonstration de son programme.

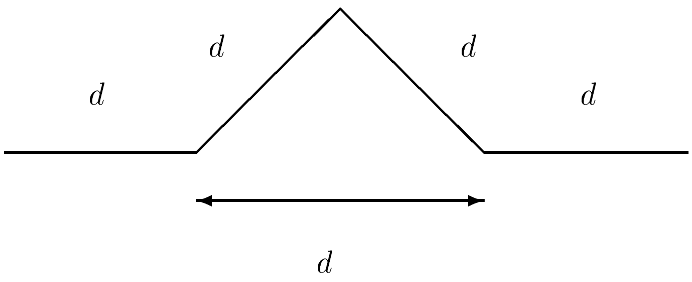
L'ensemble est évalué en tout à 4 heures de travail et 2 heures de rédaction pour chacun.Pour toute difficulté, contactez votre "TDman" par courrier électronique :

Alice.Bonhomme@ens-lyon.fr, Luc.Bouge@ens-lyon.fr

Attention ! la qualité de la rédaction est très importante. La notation prend en compte le soin, la structure et l'orthographe. En particulier, l'aspect "esthétique" de vos programmes est déterminant : indentation, aération, lisibilité, choix des identificateurs, etc. Vos programmes doivent être aussi élégants qu'une démonstration de maths !

2 Le flocon de von Koch

Il s'agit ici de dessiner une courbe fractale sur l'écran. Pour le flocon de Von Koch, il faut définir le motif de base dont la taille dépend du paramètre c et de la génération n. Le motif de base du flocon est le suivant :



À la génération 0, on avance simplement de la longueur d'un côté, c. Sinon, il faut tracer les quatre morceaux du motif de base, ces quatre morceaux étant eux mêmes des motifs.

Écrire une procédure avance distance angle qui trace à partir du point courant un segment de longueur distance dans la direction définie par angle.

Écrire une procédure récursive motif qui trace les quatre motifs de la génération $n + 1$ en s'appellant pour tracer des motifs de génération n. Cette procédure sera donc de la forme :

```
let motif generation distance angle =
    if generation = 0
       then avance distance angle
       else begin
            (...)
            end;;
```

À partir de la procédure motif dessiner un flocon en plaçant un motif de génération n sur chaque côté d'un triangle équilatéral.

Pour aller plus loin si vous avez le temps (ceci n'est pas noté) Modifiez légèrement le motif (angles, longueurs des segments, etc.) pour obtenir des flocons différents.

Aide Le problème tel qu'il est énoncé ici peut être résolu avec la seule récursivité. En particulier tous les paramètres nécessaires sont passés à la procédure motif. Il n'est donc pas nécessaire (et il est donc *interdit*... du moins dans un premier temps) d'utiliser des tableaux ou autres artifices pour passer plus d'informations...

Faites attention aux problèmes de précision des calculs en cascade : la figure peut se déformer pour les grandes valeurs de n. À vous de proposer une solution ! Commentez donc *très soigneusement* les artefacts de la figure obtenue : points manquants, distorsions, etc.

Remarque La solution s'écrit en moins de 10 lignes...

3 Les tours de Hanoï

La règle du jeu Le jeu des tours de Hanoï consiste en une plaquette de bois sur laquelle sont plantées trois tiges. Sur ces trois tiges sont enfilés des disques de bois de diamètre tous différents. La seule règle du jeu est de ne jamais poser un disque sur un disque plus petit que lui. Au début du jeu tous les disques sont posés sur la tige de gauche. Le but du jeu est de déplacer tous les disques d'une tige sur l'autre, sans jamais violer la règle, pour finalement les amener tous sur la tige de droite.

L'algorithme Supposons que les tiges s'appellent A, B et C, que n soit le nombre de disques, tous posés sur la tige A au départ, et qu'il faut les déplacer sur la tige C. La solution simple de ce jeu réside dans la remarque suivante : si on sait déplacer $n - 1$ disques, il suffit de déplacer les $n - 1$ plus petits de A vers B, puis de déplacer le disque restant de A vers C, et enfin de déplacer les $n - 1$ disques de B vers C.

Le programme Il vous est demandé d'écrire une procédure récursive qui se contente de résoudre le problème et d'afficher les mouvements avec le schéma ci-dessous :

```
let mouvement origine destination =
    print_string
        ("Je déplace un disque de la tige " ^
        origine ^
        " à la tige " ^
        destination);
    print_newline()
;;
```

Ajouter à la procédure un compteur qui donne à la fin du jeu le nombre de coups nécessaires. Calculer par les méthodes d'algorithmique ce nombre de coups pour n disques. Vérifier expérimentalement pour $n = 0, 1, 2, 3, 4, 5, \ldots$ (voir le texte d'un compteur en annexe).

Remarque Ici aussi, la solution tient en quelques lignes...

Pour aller plus loin si vous avez le temps (non noté) Une fois cette procédure écrite on souhaite remplacer l'appel de la procédure mouvement par celui d'une fonction qui dessinerait l'état du jeu à chaque étape pour faire une véritable animation.

Une possibilité est de garder en mémoire le nombre et l'identité des disques présents sur chaque tige. Pour garder en mémoire les données nécessaires, on peut par exemple utiliser un tableau linéaire nombre_de_disques dont la i^{e} case conserve le nombre de disques présent sur la i^{e} tige, et un tableau à deux dimensions dont la case (i, j) conserve l'identité du j^{e} disque présent sur la i^{e} tige.

Écrire une nouvelle procédure mouvement qui répercute en mémoire le mouvement, et qui affiche la nouvelle configuration.

A Un compteur en *Objective Caml*

Voici le texte d'un petit compteur en Caml (on peut bien sûr faire mieux !) :

```
let c= ref(0);;
let initialiser()= (c:=0);;
let incrementer()= (c:= !c+1);;
let lire()= (!c);;
```

Essayer par exemple :

```
  initialiser();
  incrementer();
  incrementer();
  print_int(lire());
  print_newline()
  ;;
```

B Documentation sur le graphisme en *Objective Caml*

Une documentation complète sur Objective est disponible à l'URL suivant (à ouvrir avec netscape) : http://caml.inria.fr/ocaml/htmlman/

Voici la marche à suivre pour pouvoir utiliser le graphisme en Caml. Les instructions ci-dessous constituent une recette de cuisine quelque peu magique...

1. Créer un "toplevel" qui contiendra toutes les fonctions graphiques par la commande suivante :

   ```
   ocamlmktop -custom -o myocaml graphics.cma
           -cclib -lgraphics -cclib -lX11
   ```

2. Appeler directement ce nouveau "toplevel" :

   ```
   myocaml
   ```

3. Enfin, ouvrir dans ocaml le module Graphics avec la commande

   ```
   open Graphics;;
   ```

Voici une liste non exhaustive de fonctions de graphismes qui pourraient vous être utiles... Vous pourrez trouver une description *en français* de ces fonctions à l'URL ci-dessus et dans le Manuel de référence de Caml disponible à la bibliothèque. (Attention, ce livre correspond à Caml Light, une version antérieure de Objective Caml : il y a quelques différences mineures !)

value open_graph: string -⟩ unit Show the graphics window or switch the screen to graphic mode. The graphics window is cleared. The string argument is used to pass optional information on the desired graphics mode, the graphics window size, and so on. Its interpretation is implementation-dependent. If the empty string is given, a sensible default is selected.

value close_graph: unit -⟩ unit Delete the graphics window or switch the screen back to text mode.

value clear_graph: unit -⟩ unit Erase the graphics window.

value_size_x: unit -⟩ unit, value_size_y: unit -⟩ unit Return the size of the graphics window. Coordinates of the screen pixels range over 0 ... size_x()-1 and 0 ... size_y()-1. Drawings outside of this rectangle are clipped, without causing an error. The origin (0,0) is at the lower left corner.

value set_color: color -⟩ unit Set the current drawing color. Possible values are (among many !) : red, green, etc. (see the extensive documentation at the above URL).

value moveto: int -⟩ int -⟩ unit Position the current point.

value current_point: int -⟩ int $*$ int Return the position of the current point.

value lineto : int -⟩ int -⟩ unit Draw a line with endpoints the current point and the given point, and move the current point to the given point.