Université Paris Diderot — Programmation Fonctionnelle
L2 Math-Info/L3 Informatique/M1 Linguistique — Année 2011-2012

TP n°6

Entrées-Sorties, Exceptions, et Graphismes

1 Graphismes

Dans cette partie, on introduit le module `Graphics` de la librairie de OCaml. Pour plus de details, voir `http://caml.inria.fr/pub/docs/manual-ocaml/libref/Graphics.html`.

Voici des extraits de la documentation du module `Graphics` qui illustrent les primitives utilisées dans la solution.

- `open_graph : string -> unit`
  Show the graphics window or switch the screen to graphic mode. The graphics window is cleared and the current point is set to (0, 0). 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.
- `moveto : int -> int -> unit`
  Position the current point.
- `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.

Pour pouvoir utiliser le mode intéractif, il est nécessaire de charger le module graphics.cma dans le toplevel OCaml. Il est possible de le charger de deux manières différentes :

- Lancer le toplevel OCaml avec comme argument graphics.cma
- Charger le module `Graphics` dans une session toplevel avec la directive toplevel suivante : `#load "graphics.cma";;`.

Pour pouvoir tester vos programmes, vous pourrez utiliser la fonction `Graphics.read_key` qui attend que l'utilisateur appuie sur une touche.

Exercice 1 [Fonctions trigonométriques] Afficher dans une fenêtre graphique le graphe de la fonction sinus entre 0 et 2π en rouge, et aussi du cosinus en bleu.

Exercice 2 [La courbe du dragon]

- La courbe du dragon d'ordre 1 entre les points A et B est le segment $[AB]$;
- Soit C le point formant un triangle isocèle rectangle en C avec A et B, à droite de $\overrightarrow{AB}$. Si A et B sont de coordonnées respectivement (x, y) et (z, t), alors C est de coordonnées (u, v) avec :
  $u = (x + z)/2 + (t - y)/2$
  $v = (y + t)/2 - (z - x)/2$
  La courbe du dragon d'ordre n est :
  - La courbe du dragon d'ordre $n - 1$ entre A et C
  - La courbe du dragon d'ordre $n - 1$ entre B et C

Écrire un programme qui dessine la courbe du dragon.

2 Ensemble de Mandelbrot

Exercice 3 Pour cette exercice, nous utiliserons le module et `Complex` de la bibliothèque standard OCaml. On rappelle juste qu'à chaque point du plan de coordonnées (x, y), on peut associer un nombre complexe noté $x+iy$. Pour représenter les complexes, la bibliothèque `Complex` définit le type :

```
type t = {
    re : float;
    im : float;
   }
```

ainsi que toutes les opérations de base (addition, multiplication...).

1. Commencez par ouvrir une fenêtre graphique de 500 par 500 et écrivez une fonction `remplit : unit -> unit` qui va remplir point par point cette fenêtre de rouge. On utilisera uniquement des fonctions récursives (pas de boucles). Le but étant de parcourir tout les points, il est interdit d'uliser les fonctions de remplissages du module `Graphics`.
2. Pour chaque point du plan d'affixe c, on définit la suite de Mandelbrot par :

$$\begin{aligned} c_0 &= c \\ c_{n+1} &= c_n^2 + c \end{aligned}$$

   L'ensemble de Mandelbrot est l'ensemble des points pour lesquels la norme des éléments de cette suite ne tend pas vers l'infini.
   On peut montrer que si la norme dépasse 2, alors la suite tend vers l'infini.
   On fera l'approximation suivante : si au bout de 50 itérations la norme est toujours inférieure à 2, alors le point est dans l'ensemble.
   Écrire une fonction
   `diverge : int -> Complex.t -> bool`
   telle que `diverge nb_iter c` renvoie `true` lorsqu'en moins de `nb_iter` itérations, la norme de `c` est plus grande que 2, et `false` sinon.
   Vérifier que la suite diverge en $(1, 1)$ mais pas en $(0, 0)$.
3. Soit `c : Complex.t` le point en bas à gauche du carré que nous voulons afficher et largeur : float sa largeur. La fonction suivante renvoie l'affixe correspondant au point de coordonnées (x, y) sur la fenêtre graphique 500x500 :

   ```
   let calcule_affixe c largeur x y =
     Complex.add c
         {re= (float_of_int x) *. largeur /. 500. ;
          im= (float_of_int y) *. largeur /. 500. }
   ```

   transformer la fonction `remplit` en une fonction
   `mandelbrot : int -> Complex.t -> float -> unit`
   qui prend en paramètres le nombre maximum d'itérations pour chaque point, `c` et `largeur`, et qui dessine l'ensemble de Mandelbrot.
   Calculer `mandelbrot 50 re= -1.5; im= -1. 2.`
4. Modifier le programme pour que la couleur soit choisie en fonction de la vitesse de divergence de la suite (nombre d'itérations pour dépasser 2).

3 Entrée-sortie depuis un fichier

Exercice 4 Créer le fichier `entreesortie.ml` et y écrire un programme qui lit indéfiniment des flottants dans un fichier `bidon.lst` et, chaque fois qu'un flottant f est lu, calcule et affiche $f + 1$.

On ne demande pas que ce programme termine (on pourra supposer que `bidon.lst` est infini), cependant **on n'utilisera pas de boucle** `while` : on écrira plutôt une fonction auxiliaire récursive.

On pourra utiliser les fonctions :

- `open_in : string -> in_channel` ouvre en lecture seule le fichier dont le nom est passé en argument, et renvoie un canal d'entrée correspondant
- `input_line : in_channel -> string` lit une ligne dans le canal d'entrée passé en argument
- `float_of_string : string -> float` vérifie si la chaîne passée en argument est un flottant, et le renvoie dans ce cas
- `print_float : float -> unit` affiche un flottant à l'écran

Tester ce programme en créant un fichier `bidon.lst` et en y entrant un flottant par ligne. Que se passe-t-il :

- lorsque le programme arrive à la fin du fichier ?
- si l'une des lignes ne contient pas de flottant ?

Le programme peut-il terminer autrement ?

Exercice 5 [Entrée depuis un fichier] Modifier le fichier `entreesortie.ml` pour y écrire une fonction : `lire : in_channel -> float list` qui, étant donné un canal d'entrée (supposé déjà ouvert), y lit des flottants jusqu'à la fin du fichier d'entrée, lève une exception si une des lignes lues n'est pas un flottant, et sinon, renvoie la liste de tous les flottants lus.

On demande que les flottants de la liste renvoyée soient dans l'ordre où ils ont été lus dans le fichier.

Remarque : on ne fermera pas le canal d'entrée dans la fonction `lire`. C'est dans le programme testant cette fonction qu'il faut le faire, en utilisant `close_in : in_channel -> unit`