Programmation fonctionnelle
TP n° 2 : Programmer avec des listes

Exercice 1 (Un piège classique)

1. Que fait la fonction suivante ?

```
let commence_par x = function
  | x :: _ -> true
  | _ -> false;;
```

2. Corrigez le problème.

Exercice 2 (Pattern-matching)

Définissez les fonctions suivantes. Vous avez interdiction d'utiliser le branchement conditionnel `if`.

1. La fonction `val_abs` qui renvoie la valeur absolue d'un réel.
2. La fonction `signe` telle que `signe x` renvoie `0` si `x = 0`, `1` si `x > 0` et `-1` sinon.
3. La fonction `fac` qui calcule la factorielle de son argument.
4. La fonction `ackermann`. Celle-ci se définit mathématiquement de la manière suivante :

$$Ack(m,n) = \begin{cases} n+1 & \text{si } m = 0, \\ Ack(m-1,1) & \text{si } n = 0, \\ Ack(m-1, Ack(m,n-1)) & \text{sinon.} \end{cases}$$

Exercice 3 (Quelques fonctions sur les listes)

Définissez les fonctions suivantes.

1. La fonction `nombre_occurrences` qui renvoie le nombre d'occurrences d'une valeur dans une liste.
   Ex : `nombre_occurrences 0 [0; 1; 2; 0; 3; 4; 0; 5]` renvoie `3`.
2. La fonction `aplatit` qui prend en paramètre une liste de listes et supprime les listes internes en concaténant tous leurs éléments.
   Ex : `aplatit [[0; 1; 2]; [3]; [4; 5]]` renvoie `[0; 1; 2; 3; 4; 5]`.
3. La fonction `compacte` prend en paramètre une liste et renvoie une liste de couples. Chaque couple est composé d'une valeur de la liste initiale et de son nombre de répétition avant de rencontrer une valeur différente.
   Ex : `compacte [0; 0; 1; 2; 2; 2; 1; 1]` renvoie `[(0, 2); (1, 1); (2, 3); (1, 2)]`.

Exercice 4 (Listes circulaires)

Une liste circulaire est une liste dont les éléments se répètent. Le successeur du dernier élément de la liste est la tête de la liste. C'est une manière d'implémenter des listes infinies (mais périodiques). Les primitives associées aux listes circulaires sont :

— `liste_circulaire` qui transforme une liste classique en liste circulaire,

— `head l` qui renvoie le premier élément de la liste circulaire `l`,
— `tail l` qui renvoie une liste circulaire identique à la liste circulaire `l` privée de son premier élément,
— `nth l k` qui renvoie l'élément d'indice `k` de la liste circulaire `l`.

1. On représente une liste circulaire par un couple `(k, l)` telle que `l` est la liste des éléments qui se répète et `k` est l'indice de l'élément courant. Implémentez les primitives associées.

```
let l1 = liste_circulaire [1; 2];; (* l1 = (0, [1; 2]) *)
head l1;; (* renvoie 1 *)
let l2 = tail l1;; (* l2 = (1, [1; 2]) *)
head l2;; (* renvoie 2 *)
let l3 = tail l2;; (* l3 = (0, [1; 2]) *)
head l3;; (* renvoie 1 *)
nth l1 5;; (* renvoie 1 *)
```

2. Quelle est l'inconvénient de cette représentation ?
3. On représente maintenant une liste circulaire par un couple `(l1, l2)`. La liste `l1` est la liste des prochains éléments à renvoyer. La liste `l2` est une copie de tous les éléments à répéter. Lorsque la liste `l1` se vide suite à des appels à la fonction `tail`, on la remplace par une copie de la liste `l2`. Implémentez les primitives associées pour cette nouvelle représentation.

```
let l1 = liste_circulaire [1; 2];; (* l1 = ([1; 2], [1; 2]) *)
head l1;; (* renvoie 1 *)
let l2 = tail l1;; (* l2 = ([2], [1; 2]) *)
head l2;; (* renvoie 2 *)
let l3 = tail l2;; (* l3 = ([1; 2], [1; 2]) *)
head l3;; (* renvoie 1 *)
nth l1 5;; (* renvoie 1 *)
```

Exercice 5 (Mandelbrot)

L'ensemble de Mandelbrot est une figure fractale définie de la manière suivante. Un point M d'affixe c appartient à l'ensemble de Mandelbrot si et seulement la suite z telle que

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

est bornée.

La figure obtenue est particulièrement complexe. On en obtient des visualisations par des approximations numériques. On sait que si un élément z_k de la suite vérifie $|z_k| > 2$ alors le point M correspondant n'appartient pas à l'ensemble. En itérant le calcul des valeurs de la suite, on peut alors déterminer des frontières de plus en plus précises de la figure.

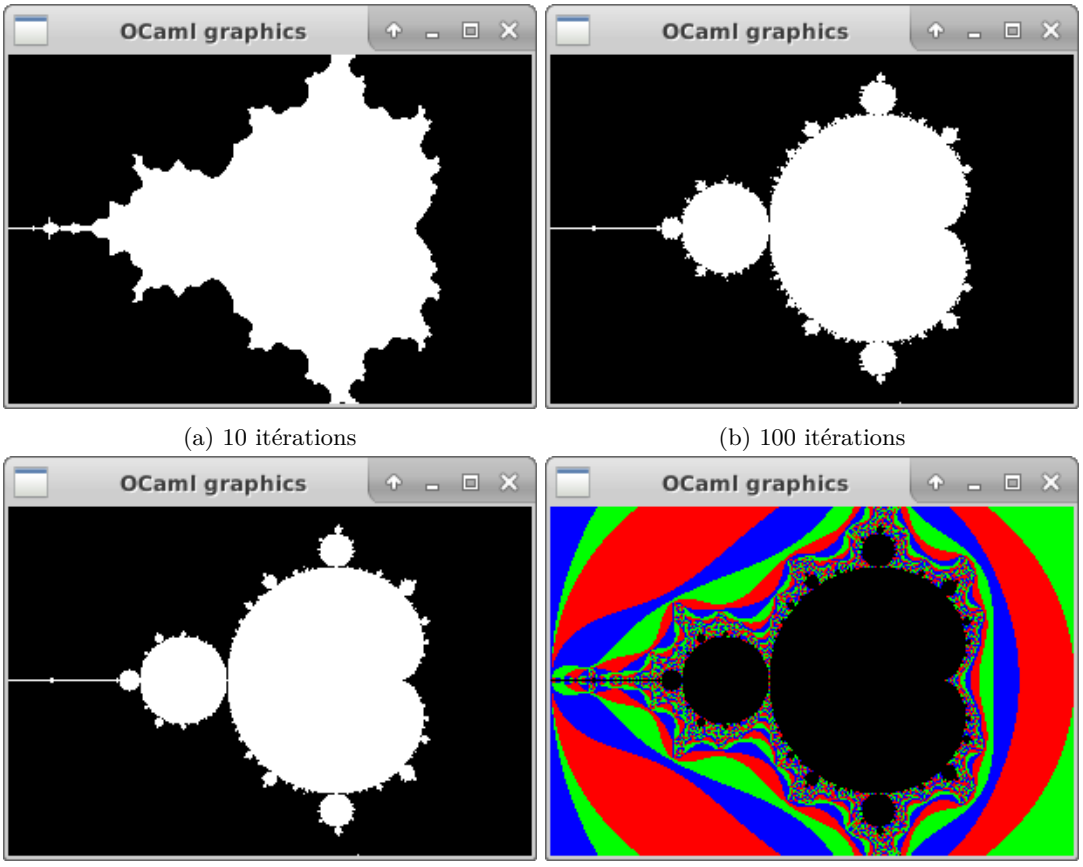
1. On représente un nombre complexe par un couple de flottants pour ses parties réelles et imaginaires. Définissez les fonctions suivantes :
   — `add_complexe` qui renvoie la somme de deux nombres complexes,
   — `mul_complexe` qui renvoie le produit de deux nombres complexes,
   — `norme` qui renvoie la norme d'un nombre complexe et
   — `valeur_suivante` telle que `valeur_suivante z c` renvoie la valeur $z^2 + c$.
2. Définissez une fonctions `liste_pixels` telle que `liste_pixels n m` renvoie la liste de toutes les coordonnées entières `(i, j)` avec `0 <= i < n` et `0 <= j < m`. Ce sont les coordonnées graphiques que vous allez manipuler.
3. Il faut associer chaque coordonnée graphique au point du plan qui lui correspond (via son affixe) et à la valeur courante de la suite z associée (initialement 0). Il s'agit principalement d'un changement de repère où l'on transforme le couple (i, j) en $((i, j), (\delta_x + i \times d, \delta_y + j \times d), (0.0, 0.0))$. Le complexe $\delta_x + i\delta_y$

est l'affixe associé au point de coordonnée graphique $(0, 0)$. La valeur d est la distance (en terme d'affixe) séparant deux coordonnées graphiques adjacentes.

Définissez une fonction `ajouter_affixes pixels delta_x delta_y d` qui réalise cette transformation pour toutes les coordonnées graphiques de la liste `pixels`.

4. Charger la bibliothèque graphique et créez une fenêtre de taille 300 par 200.
5. Définissez une fonction `mandelbrot_une_etape` qui prend en paramètre une liste de données de la forme $((i, j), c, z)$ et qui
   — Colorie le pixel de coordonnées (i, j) si $|z| > 2$ et sors la donnée associée de la liste. Utilisez la fonction `Graphics.plot` pour cela.
   — La transforme en la donnée $((i, j), c, z^2 + c)$ dans le cas contraire.
   On obtient en sortie de la fonction la liste des données à traiter lors de l'itération suivante.

   On rappelle que les valeurs c et z sont des complexes et sont donc représentée par des couples.
6. Déduisez-en une fonction `mandelbrot n` qui itère sur chacun des points de la fenêtre graphique pour déterminer son appartenance à l'ensemble de Mandelbrot. La fonction termine au bout de `n` itérations. Les données numériques à utiliser sont $\delta_x = -2$, $\delta_y = -1$ et $d = 0.01$. On rappelle que la fenêtre graphique est de taille 300 par 200.
7. *Bonus.* Il est usuel de dessiner l'ensemble de Mandelbrot avec plusieurs couleurs. Une couleur différente est utilisée pour chaque itération. Ceci permet de visualiser la vitesse de divergence des points à la frontière. Sauriez-vous modifier la fonction précédente pour y parvenir ?



(a) 10 itérations (b) 100 itérations

(c) 1000 itérations (d) Version colorée

FIGURE 1 – Exemples de sorties