Programmation fonctionnelle - TP - séance 3

Exercice 1 (**Quelques fonctions simples sur les listes**)

1. Écrire en OCaml la fonction `repeat` qui prend deux arguments, un entier `n` et une valeur x, et calcule la liste de longueur `n` ne contenant que des `x`. Quel est le type de cette fonction ?

2. Soit la fonction `nb_occ` dont l'interface est donnée ci-dessous :

   ```
   (** nb_occ
   @param x, un élement, l une liste
   @return le nombre d'occurrences de x dans la liste l
   *)
   ```

   Quel est le type de cette fonction ? L'écrire en OCaml et la tester.

3. Écrire la fonction `last` qui retourne le dernier élément d'une liste. Elle échoue si la liste est vide. Vous commencerez par écrire l'interface de la fonction. Vérifier son type.

4. Écrire la fonction `is_increasing` qui teste si une liste est croissante. On considérera que la liste vide est croissante.

5. La fonction `moyenne` renvoie la moyenne (de type `float`) des nombres d'une liste d'entiers.
   Écrire une version de la fonction qui utilise les fonctions `somme` et `lg` (TD précédent) et une version plus optimale (penser à écrire une fonction auxiliare qui calcule somme et nombre d'éléments en même temps. Pour cela, inspirez-vous de la fonction vue en cours qui calcule reste et quotient simultanémént).
   La fonction `float_of_int` convertit un entier en le flottant correspondant : ainsi l'expression `(float_of_int 2) +. 3.5` est une expression bien typée.

Exercice 2 (**Listes triées**)

On désire maintenant manipuler des listes triées dans l'ordre croissant par exemple.

1. Ecrire une fonction qui insére un élément dans une liste triée. Le résultat obtenu doit être une liste triée dans l'ordre croissant.

2. En utilisant la fonction précédente, écrire la fonction qui réalise le tri d'une liste avec la méthode *tri par insertion*.

3. Faire la concaténation de deux listes triées de manière à obtenir une nouvelle liste triée (on parle alors de fusion de deux listes).

Exercice 3 (**Fonctionnelles**)

Dans cet exercice on utilisera les fonctionnelles `map`, `fold_left` ou `fold_right` (définies dans le module `List` de la bibliothèque standard).

1. ```
   let sqrt x = x * x;;
   let mylist = [3 ; 12 ; 3 ; 40 ; 6 ; 4 ; 6 ; 0];;
   ```

   - Sans définir de nouvelles fonctions, créer la liste des carrés des éléments de `mylist`.

   ```
   carres mylist;;
    - : int list = [9; 144; 9; 1600; 36; 16; 36; 0]
   ```

   - Sans définir de nouvelles fonctions, créer la liste des doubles des éléments de `mylist`.
   ```

```
doubles mylist;;
 - : int list = [6; 24; 6; 80; 12; 8; 12; 0]
```

2. En une ligne (ou parfois deux), en utilisant les fonctions d'ordre supéerieur sur les listes, écrire les fonctions suivantes :

   (a) `somme` qui renvoie la somme des éléments d'une liste d'entiers.

   (b) `lg` qui renvoie la taille d'une liste.

   (c) `pairs` qui compte le nombre d'entiers pairs dans une liste.

   (d) `nb_occ` qui compte le nombre d'occurrences d'un élément dans une liste.

   (e) `minl` qui renvoie le plus petit élément d'une liste non vide.

   (f) `moyenne` qui renvoie la moyenne (en flottant) des nombres entiers d'une liste (sans utiliser `lg` et `somme`).

   (g) `map`.

Exercice 4 (**Pour les plus rapides - Nombres parfaits**)

Un nombre parfait est un nombre naturel égal à la somme de ses diviseurs (lui-même non compris). Par exemple, 6 (= 1 + 2 + 3) est un nombre parfait. 28 (= 1+2+4+7+14) est un nombre parfait.

1. Écrire une fonction `est_parfait` qui teste si un nombre est parfait ou non. On commencera par écrire une fonction qui calcule la liste des diviseurs propres puis on écrira la fonction recherchée.

2. Réécrire ensuite cette fonction de manière à ce qu'elle ne construise pas la liste intermédiaire des diviseurs propres.

Exercice 5 (**Pour les plus rapides - Tri sélection**)

Écrire une fonction qui trie une liste d'entiers en utilisant le principe du tri par sélection.