Université Paris Diderot
L3 Informatique

Programmation Fonctionnelle
Année 2015-2016

# TP n°6

## Récursion terminale

**Exercice 1.** On considère la fonction `listn` qui étant donné un entier positif `n` renvoie une liste contenant tous les entiers de `n` à `0`. Une façon de l'écrire est celle-ci :

```
let rec listn n = if n = 0 then [0] else n::(listn (n-1));;
```

Jusqu'à quelle valeur de `n` cela marche ? Écrire une version `listn1` utilisant la récursion terminale. Jusqu'à quelle valeur de `n` cela marche ?

**Exercice 2.** On considère la fonction qui calcule la longueur d'une liste.

```
let rec length l = match l with
     [] -> 0
   | _::t -> 1 + (length t);;
```

Utiliser la fonction `listn1` pour tester `length`. Jusqu'à quelle taille de liste cela fonctionne ? Écrire une fonction récursive terminale `length1` qui calcule la longueur d'une liste. Vérifier que cela améliore la situation.

**Exercice 3.** Écrire une fonction récursive terminale pour la suite de Fibonacci.

**Exercice 4.**

1. Écrire une fonction récursive terminale `append'` qui prend en paramètre deux listes `left` et `right` et qui retourne la concaténation de l'inverse de la liste `left` et de la liste `right`. Par exemple, un appel à `append' [1;2;3] [4;5;6]` retournera `[3;2;1;4;5;6]`.
2. Écrivez une fonction `append` qui concatène les deux listes passées en paramètre en appelant `append'`.

**Exercice 5.** Pour compter le nombre de lignes d'un fichier on pourrait utiliser la fonction suivante :

```
let rec compter_lignes_c c n =
  try
    let l = input_line c in compter_lignes_c c (n+1)
  with End_of_file -> n;;
let compter_lignes f =
  let c = open_in f in compter_lignes_c c 0;;
```

Appliquer cette fonction sur un fichier de grande taille (par exemple le fichier `mots` sur Didel). Que se passe-t-il ? Écrire une version récursive terminale de `compter_lignes` qui fonctionne sur ce fichier.

**Exercice 6.** On se donne le type

```
type 'a tree = Nil | Node of 'a * 'a tree * 'a tree;;
```

On rappelle qu'on appelle *taille* d'un arbre $A$ le nombre de nœuds de $A$.

1. Écrire une fonction `taille1` (pas forcément récursive terminale) qui calcule la taille d'un arbre.
2. Écrivez une fonction récursive terminale

   ```
   taille' : 'a tree list -> int -> int
   ```

   qui prend en paramètre une liste d'arbres $L$ et un entier $n$ et retourne la somme
   — des tailles des éléments de $L$ et
   — de l'entier $n$.
3. Écrivez une fonction `taille` qui calcule la taille de l'arbre passé en paramètre en appelant `taille'`.

**Exercice 7.** On reprend des exercices du TP 2. Ecrire les fonctions suivantes avec **récursion terminale**. Toutes les fonctions auxiliaires utilisées doivent aussi être récursives terminales.

1. Ecrire une fonction `insert` qui étant donnée une liste supposée triée pour l'ordre strictement croissant et un élément `x`, renvoie la liste obtenue en insérant `x` à la bonne place. Si `x` est déjà dans la liste, celle-ci sera renvoyée telle quelle.

   Exemples :

   ```
   insert 5 [1;3;8] = [1;3;5;8]
   insert 'e' ['a';'c';'g'] = ['a';'c';'e';'g']
   ```
2. En utilisant la fonction `insert`, écrire une fonction `sort` permettant de trier une `'a list` quelconque par ordre croissant, en fusionnant les doublons.

   Exemple : `sort [7;8;5;2;8] = [2;5;7;8]`
3. Ecrire les opérations d'union et d'intersection (`union_sorted, inter_sorted`) de deux listes triées.

   Exemples :

   ```
   union_sorted [1;3;5] [2;5;8] = [1;2;3;5;8]
   inter_sorted [1;3;5] [2;5;8] = [5]
   ```