UNIVERSITÉ LOUIS PASTEUR STRASBOURG

**Département d'Informatique**
*LICENCE 2ème année*
*Mardi 28 octobre 2008*

# Travaux Dirigés de Programmation Fonctionnelle

# «Schémas de programmes»

## I Schémas de réduction

1. Les schémas `List.fold_right` et `List.fold_left` sont rappelés ci-dessous :

```
List.fold_right f [a1; ...; an] b = f a1 (f a2 (... (f an b) ...))
List.fold_left  f a [b1; ...; bn] = f (... (f (f a b1) b2) ...) bn
```

   Définir les fonctions suivantes, d'abord en utilisant `List.fold_right`, puis en utilisant `List.fold_left` :

   (a) Concaténation des éléments d'une liste de chaines de caractères.
   Exemple : `concat_string ["a";"b";"c"] = "abc"`

   (b) Liste des nombres pairs d'une liste de nombres entiers (en préservant l'ordre des éléments dans la liste).
   Exemple : `pairs [2;3;5;7;4] = [2;4]`

2. On cherche à définir un schéma `reduce_tree` de réduction des arbres binaires en s'inspirant du schéma de réduction des listes. Le principe du schéma `reduce_tree` est de synthétiser un résultat en un noeud de l'arbre à partir des résultats en ses noeuds fils. On utilisera le type suivant :

```
type 'a tree = Leaf of 'a | Branch of 'a tree * 'a tree;;
```

   des arbres binaires à valeur aux feuilles. Définir une fonctionnelle qui implante le schéma `reduce_tree`.
   Exemple d'utilisation de ce schéma pour calculer la somme des éléments d'un arbre de nombres entiers :

```
# reduce_tree (+) (Branch (Branch (Leaf 5, Leaf 2), Leaf 4));;
- : int = 11
```

3. Utiliser le schéma `reduce_tree` de la question précédente pour définir une fonction de tri des éléments dans un arbre. (On considérera un arbre dont les éléments sont des listes à 1 élément et on définira une fonction `fusion` qui fusionne deux listes triées.)

## II Schéma map et ses dérivés

1. Le schéma `map` est rappelé ci-dessous :

   ```
   map f [e1; ...; en] = [f e1; ...; f en]
   ```

   Donner une définition de ce schéma en utilisant `List.fold_right`.

2. Utiliser le schéma `map` pour définir une fonctionnelle `adjoint` qui ajoute un même élément en tête de tous les éléments d'une liste de listes.
   Exemple : `adjoint 1 [[];[2];[3;4]] = [[1];[1;2];[1;3;4]]`

3. Utiliser la fonctionnelle `adjoint` pour définir récursivement la fonction `prefix` qui retourne tous les débuts de liste d'une liste.
   Exemple : `prefix [1;2;3;4] = [[];[1];[1;2];[1;2;3];[1;2;3;4]]`

4. Utiliser les schémas `map` et `List.fold_right` pour définir une fonction `scan` qui retourne la liste des sommes partielles d'une liste de nombres entiers.
   Exemple : `scan [1;2;3;4] = [0;1;3;6;10]`

## III Schéma de récusivité terminale

1. Donner une définition performante des fonctions suivantes (on se servira d'une fonction auxiliaire à récursivité terminale) :
   (a) Somme des $n+1$ premiers termes d'une suite
   Exemple : `somme (fonction n -> n) 100000 = 705082704`
   (b) Liste des entiers entre $n$ et $m$
   Exemple : `entre 4 12 = [4; 5; 6; 7; 8; 9; 10; 11; 12]`

2. Peut-on trouver une définition récursive terminale pour la fonctionnelle `List.fold_left` ? Même question avec `List.fold_right`.