Programmation Fonctionnelle :
des concepts aux applications web
(NFP119)

Les arbres binaires

On rappelle le type `'a tree` des arbres binaires polymorphes (où une valeur est associée à chaque nœud) vu en cours :

```
type 'a tree = Empty | Node of 'a * 'a tree * 'a tree
```

1. Définir un arbre de type `int tree` contenant au moins 5 nœuds.

2. Définir un arbre de type `(string option) tree` contenant au moins 5 nœuds et au moins 3 chaînes de caractères.

3. Ecrire une fonction `mkLeaf : 'a -> 'a tree` qui construit un arbre réduit à un unique nœud (une feuille).

4. Ecrire une fonction `traversal : 'a tree -> 'a list` qui construit la liste des valeurs associée aux nœuds d'un arbre parcouru dans l'ordre infix (de gauche à droite).

5. Ecrire une fonction `tmap : ('a -> 'b) -> 'a tree -> 'b tree` telle que `tmap f t` renvoie l'arbre obtenu en applicant la fonction `f` aux valeurs contenues dans les nœuds de l'arbre `t`.

6. Vérifier sur un exemple que `map f (traversal t) = traversal (tmap f t)` (où la fonction `map` est celle du module `List`).

On rappelle le type `'a ltree` des arbres binaires polymorphes (où une valeur est associé à chaque feuille) vu en cours :

```
type 'a ltree = Leaf of 'a | LNode of 'a ltree * 'a ltree
```

1. Ecrire une fonction `ltraversal : 'a ltree -> 'a list` qui construit la liste des feuilles d'un arbre parcouru de gauche à droite.

2. Ecrire une fonction `lmap : ('a -> 'b) -> 'a ltree -> 'b ltree` telle que `lmap f t` renvoie l'arbre obtenu en appliquant la fonction `f` à chaque feuille de `t`.

3. Ecrire une fonction `convert : 'a ltree -> ('a option) tree` qui convertit un arbre de type `'a ltree` en arbre de type `('a option) tree`. On pourra utiliser la fonction `mkLeaf o Some` pour construire les feuilles.

4. Verifier sur des examples ce que calcule la fonction `traversal o convert`. Comment peut-on utiliser cette fonction pour redéfinir `ltraversal` ?