TP 4 Programmation fonctionnelle

Exercice 1 : MAP

1. Redéfinir la fonction `map : ('a-> 'b)-> 'a list-> 'b list` telle que `map f [x1 ;...; xn]` renvoie `[ f x1 ;...; f xn]`.
2. En utilisant la fonction prédéfini map : écrire la fonction `increment_all : [1; 2; 3; 4; 5] --> [2; 3; 4; 5; 6]`
3. En utilisant la fonction prédéfini map : écrire la fonction `parity [1; 2; 3; 4; 5] --> [true; false; true; false; true]`

Exercice 2 : FILTER

1. Redéfinir la fonction `filter ('a-> bool)->'a list-> 'a list` telle que `filter f l` renvoie la sous-liste des éléments x de l pour lesquels `f x` renvoie true.
2. En utilisant la fonction prédéfini filter : écrire la fonction `keep positive : [10; -15; 12; 13]--> [10; 12; 13]`
3. En utilisant la fonction prédéfini filter : écrire la fonction `keep even: [10; -15; 12; 13] --> [10; 12]`

Exercice 3 : FOLD

1. Redéfinir les fonctions `fold_right: ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b` et `fold_left: ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`
2. Imprimer les éléments aux index pairs dans la liste à l'aide de fold
3. Implémenter la fonction slice qui extrait un fragment d'une liste en utilisant fold

Exercice 4. L'objet de cet exercice est d'explorer la programmation d'ordre supérieur sur les arbres. On réalisera l'opérateur `suffix` réalisant le calcul suffixe sur un arbre. Le calcul suivra donc les principes du parcours suffixe sur les arbres. La fonction `suffix f t v` admet 3 arguments : 1) une fonction f à 3 paramètres déterminant le calcul à réaliser en fonction du résultat des sous-arbres et la valeur courante d'une cellule, 2) l'arbre t, et une valeur v quand l'arbre est vide (Null). Par exemple la somme des valeurs sera décrite ainsi pour l'arbre t : `suffix (fun a l r ->a+l+r) t 0`.
Ensuite refaire la fonction `height` en utilisant cet opérateur et la fonction qui compte le nombre de nœuds.