Programmation fonctionnelle et logique
TD6 et TP3 : itérateurs et exceptions

Itérateurs sur arbres n-aires

```
type arbre = Feuille of int | Noeud of (int * arbre list);;
```

Rappel sur les itérateurs

`List.iter : ('a -> unit) -> 'a list -> unit`.
`List.iter f [a1 ; ... ; an]` est équivalent à `begin f a1 ; ... ; f an ; () end`.

`List.map : ('a -> 'b) -> 'a list -> 'b list`.
`List.map f [a1 ; ... ; an]` est équivalent à `[f a1 ; ... ; f an]`.

`List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`.
`List.fold_left f a [b1 ; ... ; bn]` est équivalent à `f ( ... (f (f a b1) b2) ... ) bn`.

`List.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b` .
`List.fold_right f [a1 ; ... ; an] b` est équivalent à `f a1 (f a2 (... (f an b) ...))`.

Fonctions élémentaires sur les arbres

1. Écrire une fonction `make_arbre` qui construit un arbre à partir d'un entier et d'une lise de sous-arbres.
2. Écrire des fonctions `clé`, `fils` qui extraient respectivement d'un nœud interne sa clé et la liste de ses fils.
3. Définissez une fonction `feuille` qui renvoie le booléen `true` si et seulement si son argument est une feuille.

Calcul sur les arbres

1. Définissez une fonction qui calcule la hauteur d'un arbre.
2. Construisez les fonctions `nb_feuilles` et `nb_noeuds` qui calculent respectivement le nombre de feuilles et de nœuds (nœuds internes et feuilles) d'un arbre.

Parcours dans les arbres

1. Écrivez une fonction `parcours_profondeur` qui énumère les clefs d'un arbre en profondeur d'abord.
2. Écrivez une fonction `appartient` qui renvoie `true` si et seulement si l'entier argument est contenu dans l'arbre étudié.
3. Écrivez une fonction `parcours_largeur` qui énumère les clefs d'un arbre en largeur d'abord.

Exceptions

On se place de nouveau dans le cadre des arbres binaires définis par le type suivant :

```
type arbre = Feuille of int | Noeud of (int * arbre  * arbre);;
```

- Écrivez une fonction `appartient` qui renvoie `true` si et seulement si l'entier argument est contenu dans l'arbre étudié ; et qui se termine dès que l'élément recherché a été trouvé.
- Écrivez une fonction `appartient_prof` ayant le même comportement que la fonction `appartient`, mais renvoyant en plus la profondeur du nœud contenant l'élément recherché (on renverra 0 si l'élément recherché est absent de l'arbre).
- Écrivez une fonction `appartient_arbre` ayant le même comportement que la fonction `appartient`, mais renvoyant en plus le sous-arbre dont la racine est l'élément recherché (on renverra l'arbre de départ si l'élément recherché en est absent).
- Écrivez une fonction `appartient_arbre_père` qui recherche un élément dans un arbre et qui renvoie le sous-arbre dont un des fils a l'élément recherché pour racine (si celui existe).