TP de programmation fonctionnelle et logique
TP 2 : arbres n-aires et exceptions

Arbres n-aires

Un arbre n-aire est soit une feuille (un entier), soit un nœud interne (composé d'un entier, la clef, et d'une liste de sous-arbres). Nous définissons un arbre n-aire par le type suivant :

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

Fonctions élémentaires sur les arbres

1. Écrire une fonction `make_arbre` qui construit un arbre à partir d'un entier et d'une liste de sous-arbres.
2. Écrire des fonctions `clé` et `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 `appartient_opt`, nouvelle version de `appartient` optimisée par l'usage d'exceptions.
4. 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);;
 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).