TP1 Caml : Arbres

1 Opérations de base sur les arbres binaires

On considère des arbres binaires décrits par le type Caml suivant :

```
type ('f,'n) arbre =
  Feuille of 'f
| Noeud of 'n * ('f,'n) arbre * ('f,'n) arbre ;;
```

1. Ecrire une fonction `hauteur` qui calcule la hauteur d'un arbre.

2. Ecrire une fonction `nombre_feuille` qui calcule le nombre de feuilles d'un arbre.

3. Ecrire une fonction `nombre_noeud` qui calcule le nombre de nœuds d'un arbre.

4. Ecrire une fonction `miroir` qui calcule l'image miroir d'un arbre.

2 Arbres binaires de recherche

Un arbre binaire de recherche est un arbre binaire tel qu'en chaque nœud, la valeur apparaissant est plus grande que toutes les valeurs apparaissant dans le fils gauche et plus petite que toutes les valeurs apparaissant dans le fils droit, pour une relation d'ordre donné. Seuls les nœuds possèdent une information, donc il n'y a pas besoin d'étiquette sur les feuilles. On prend ainsi comme type :

```
type 'n arbre_r =
  Vide
| Noeud of 'n * 'n arbre_r * 'n arbre_r ;;
```

On prendra comme convention que tous les éléments d'un arbre binaire de recherche sont distincts.

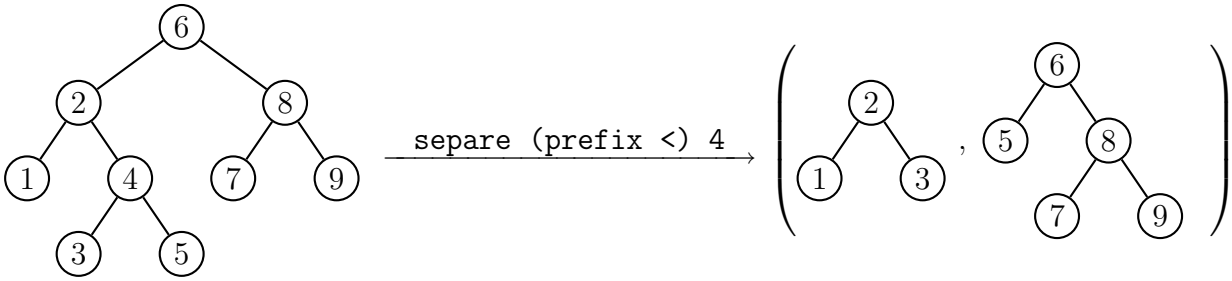
5. Ecrire une fonction `insere` telle que `insere comp x arbre` insère l'élément `x` de type `a` dans l'arbre binaire de recherche `arbre`, pour la relation de comparaison `comp` de type `a -> a -> bool` (on insère, lorsque c'est nécessaire, aux niveaux des feuilles de l'arbre).

6. Ecrire une fonction `retire_plus_grand` qui, appliquée à un arbre binaire de recherche, renvoie une paire formée de son plus grand élément et de l'arbre de départ privé de cet élément (sans connaître la relation de comparaison utilisée).

7. En déduire une fonction `retire_racine` qui, appliquée à un arbre binaire de recherche `arbre`, retourne cet arbre privé de sa racine.

8. En déduire une fonction `retire` telle que `retire comp x arbre` retourne l'arbre binaire de recherche `arbre` privé de l'élément `x` (s'il appartient à l'arbre).

9. Ecrire une fonction `separe` telle que `separe comp x arbre` retourne un couple `(arbre1, arbre2)` d'arbres binaires de recherche où `arbre1` (resp. `arbre2`) contient tous les éléments de l'arbre binaire de recherche `arbre` inférieurs (resp. supérieurs) à `x` pour l'ordre `comp`, au sens strict. Par exemple :



10. En déduire une fonction `insere_racine` qui ajoute un élément à un arbre binaire de recherche en l'insérant à la racine.

11. Ecrire une fonction `test` telle que `test comp a b arbre` vérifie si l'arbre `arbre` est un arbre binaire de recherche dont tous les éléments sont supérieurs à `a` et inférieurs à `b` pour l'ordre `comp`.

12. En déduire une fonction `test_int` qui vérifie si un arbre est un arbre binaire de recherche sur les entiers (pour l'ordre `prefix <`).

3 Représentation avec des pointeurs

Dans certains sujets de concours, les arbres manipulés ne sont pas représentés par le type précédent. Nous allons étudier ici un autre type de représentation. :

```
type 'n arbre_p =
  Vide_p
| Noeud of 'n enreg_Noeud
and 'n enreg_Noeud =
  { mutable val : 'n;
    mutable gauche : 'n arbre_p;
    mutable droit  : 'n arbre_p };;
```

On utilise ici les champs mutables des enregistrements pour définir les arbres. Grâce à cette mise en œuvre, on peut modifier un arbre sans le recopier.

13. Réécrire la fonction `miroir` et la fonction `insere` (pour un arbre non vide) avec ce nouveau type de représentation. Dans la mesure du possible on modifiera les champs de l'arbre donné en paramètre, plutôt que d'un recréer un.