Homework 6: Recursion over trees

Programming Languages

Due: Friday, September 30

We define a new type of *labeled trees* in which *both leaves and nodes* are labeled by data elements of type `a`:

```
data LTree a = LLeaf a | LNode a (LTree a) (LTree a)
  deriving Show
```

Problem Set 1

Implement each of the following functions, using recursion and pattern-matching.

1. `getLeaves :: LTree a -> [a]`
   takes a labeled tree and returns the contents of the *leaves* of the tree in a single list:
   `getLeaves (LNode "root" (LLeaf "left") (LLeaf "right")) = ["left","right"]`

2. `countNodes :: LTree a -> Integer`
   counts the number of *non-leaf nodes* in the given tree.

3. `sumTree :: LTree Integer -> Integer`
   sums up all the integers occurring in a tree of integers, *including leaves and nodes*:
   `sumTree (LNode 1 (LLeaf 2) (LLeaf 3)) = 6`

4. `occursInLeaves :: (a -> Bool) -> LTree a -> Bool`
   takes a predicate `p :: a -> Bool`, and returns `True` if there is some element in one of the *leaves* satisfying this predicate, and returns `False` otherwise. For example,

   occursInLeaves (== 'a') (LNode 'a' (LLeaf 'b') (LLeaf 'c')) = **False**
   occursInLeaves (== 'a') (LNode 'b' (LLeaf 'a') (LLeaf 'c')) = **True**
   occursInLeaves **odd** (LNode 1 (LLeaf 2) (LLeaf 3)) = **True**
   occursInLeaves **odd** (LNode 1 (LLeaf 2) (LLeaf 4)) = **False**

5. `checkNoCover :: (Eq a) => a -> LTree a -> Bool`
   returns `True` if the first argument occurs among the leaves of the tree given in the second argument, *and does not occur on any node above on the path from the root to that leaf.*

It returns `False` if every occurrence in the leaves is "covered" by another occurrence higher in the tree, which the leaf is a descendant of.

```
checkNoCover 0 (LLeaf 0) = True
checkNoCover 0 (LNode 2 (LLeaf 1) (LLeaf 3)) = False
checkNoCover 0 (LNode 2 (LLeaf 0) (LLeaf 3)) = True
checkNoCover 0 (LNode 0 (LLeaf 0) (LLeaf 3)) = False
checkNoCover 0 (LNode 0 (LLeaf 0) (LLeaf 0)) = False
checkNoCover 0 (LNode 1 (LNode 0 (LLeaf 0) (LLeaf 0)) (LLeaf 0)) = True
```

Problem Set 2

Implement all of the problems 1–5 from the above, but this time *without using recursion.* Instead, use the "tree folding" function defined below.

Inside your implementation file, attach an apostrophe `'` to the name of every function you define, to distinguish it from your answers to part 1 of the homework. (`minTree'`, `countLeaves'`, etc.)

```
foldTree :: (a -> b -> b -> b) -> (a -> b) -> LTree a -> b
foldTree comb base (LLeaf x) = base x
foldTree comb base (LNode y t1 t2) = comb y (foldTree comb base t1)
                                            (foldTree comb base t2)
```

Submission instruction

Enter all your definition in the same file, named `LASTNAMEhw6.hs` and upload it to ASUlearn before the deadline.