CS 421 --- Algebraic Data Types Activity

Manager	Keeps team on track	
Recorder	Records decisions	
Reporter	Reports to class	
Reflector	Assesses team performance	

Learning Objectives

1. ...

Understanding the Types

Here is a datatype to implement a BST.

```haskell
data BST a = Empty
    | Node a (BST a) (BST a)
  deriving (Show,Eq)
```

Problem 1) Consider the following assignments `t1`, `t2`, and `t3`. Which are legal, and which are not? For the ones that are not, why not?

```haskell
t1 = Node 4 Empty (Node 5 Empty Empty)
t2 = Node 8 (BST 3) (BST 4)
t3 = Node "hi" Empty (Node 10 Empty Empty)
```

Problem 2) Consider the following helper functions. Two allow us to deal with leaf nodes, and one performs a left-rotation. There are quite a few references to `Node` and `Empty` here, but only a few of them cause memory to be allocated. Where are they, and how can you tell?

```haskell
isLeaf (Node x Empty Empty) = True
isLeaf _ = False

mkLeaf n = Node n Empty Empty

rotateLeft (Node b a (Node d c e)) = Node d (Node b a c) e
```

Problem 3) Can you write the corresponding `rotateRight` function?

Implementing Add

```haskell
add elt Empty = mkLeaf elt
add elt n@(Node x a b) | elt < x = Node x (add elt a) b
                       | elt > x = Node x a (add elt b)
                       | otherwise = n
```

Problem 4) We haven't gone over the `n@` syntax yet. What do you think it means, and what would happen if we didn't have it?

Problem 5) How does this data structure handle it if we add multiple copies of an element?

Problem 6) Write a function that will create a tree from the elements of a list. For extra Haskell points, do it in **one line** using a higher order function.

```
Prelude> list2Tree [1,3,2]
Node 2 (Node 1 Empty Empty) (Node 2 Empty Empty)
```

Implementing Delete

```
del victim Empty = Empty
del victim (Node foo left right)
          | foo > victim = Node foo (delete victim left) right
          | foo < victim = Node foo left (delete victim right)
          | foo == victim =
   case (left,right) of
      (Empty,Empty) -> Empty
```

Problem 7) What cases does the starter code above handle? Oh, and there's a bug; please fix that.

Problem 8) Extend the code to handle the case where there is one child.

Problem 9) Consider the following helper function.

```
goLeft (Node a _ Empty) = a
goLeft (Node a _ b)     = goLeft b
```

How can this function be of use to us?

Problem 10) Implement two child deletion. Using `let`, you can actually do this in one or two lines.

Algebraic Data Types Activity--- Reflector's Report

Manager	Keeps team on track	
Recorder	Records decisions	
Reporter	Reports to Class	
Reflector	Assesses team performance	

1. What was a strength of your team's performance for this activity?

2. What could you do next time to increase your team's performance?

3. What insights did you have about the activity or your team's interaction today?