Understanding the Types

Here is a datatype to implement a BST.

```haskell
0 data BST a = Empty
1     | Node a (BST a) (BST a)
2 deriving (Show, Eq)
3
4 data Box a = Box a
5
6 t1 = Node 4 Empty (Node 5 Empty Empty)
7 b1 = Box 20
```

**Problem 1)** For the BST type, what is the purpose of Empty and Node?

**Problem 2)** The text `Box a` occurs twice in the above code. What is the difference between the two occurrences?

**Problem 3)** The Haskell expression `show t1` works, but `show b1` does not. Why not?

**Problem 4)** Consider the following assignments for `b2`, `t2`, and `t3`. They are not legal. why not?

```haskell
0 b2 = Box 10 20
1 t2 = Node 8 (BST 3) (BST 4)
2 t3 = Node "hi" Empty (Node 10 Empty Empty)
```

**Problem 5)** 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
0 isLeaf (Node x Empty Empty) = True
1 isLeaf _ = False
2 mkLeaf n = Node n Empty Empty
3 rotateLeft (Node b a (Node d c e)) = Node d (Node b a c) e
```
Implementing Add

**Problem 6)** Can you write the corresponding `rotateRight` function?

Consider this code:

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

**Problem 7)** 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 8)** How does this data structure handle it if we add multiple copies of the same element?

**Problem 9)** 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.

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

\( \text{del victim } \text{Empty} = \text{Empty} \)
\( \text{del victim } (\text{Node } \text{foo left right}) \)
\( \quad | \text{foo} > \text{victim} = \text{Node } \text{foo} \text{ (delete victim left) right} \)
\( \quad | \text{foo} < \text{victim} = \text{Node } \text{foo left} \text{ (delete victim right)} \)
\( \quad | \text{foo} == \text{victim} = \)
\( \text{case (left,right) of} \)
\( \quad (\text{Empty,Empty}) \rightarrow \text{Empty} \)

**Problem 10)** What cases does the starter code above handle? Oh, and there's a bug; please fix that.

**Problem 11)** Extend the code to handle the case where there is one child.

**Problem 12)** Consider the following helper function.

\( \text{goLeft } (\text{Node } \text{a _ Empty}) = \text{a} \)
\( \text{goLeft } (\text{Node } \text{a _ b}) = \text{goLeft b} \)

How can this function be of use to us?

**Problem 13)** Implement two child deletion. Using \texttt{let}, you can actually do this in one or two lines.
Algebraic Data Types Activity --- Team's Assessment (SII)

Manager or Reflector: Consider the objectives of this activity and your team's experience with it, and then answer the following questions after consulting with your team.

1. What was a **strength** of this activity? List one aspect that helped it achieve its purpose.

2. What is one thing we could do to **improve** this activity to make it more effective?

3. What **insights** did you have about the activity, either the content or at the meta level?