Introduction to Programming 1: Assignment 3

Due: November 8, 2015. 11 pm

Important Instructions: Submit your solution in a single file named *loginid.3.hs* on Moodle. For example, if I were to submit a solution, the file would be called *spsuresh.3.hs*. You may define auxiliary functions in the same file, but the solutions should have the function names specified by the problems.

1. Consider the following declaration of a binary tree storing integers.

```
data BTree = Nil | Node BTree Int BTree
```

Recall that *inorder traversal* is one where at each node, you inductively list out the left subtree (according to inorder traversal), then list out the value at the node, and finally list out the right subtree (in inorder). The code is given below:

```
inorder :: BTree -> [Int]
inorder Nil              = []
inorder (Node tl x tr)   = inorder tl ++ [x] ++ inorder tr
```

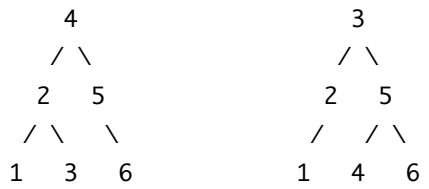
We can similarly define *preorder traversal* and *postorder traversal*.

```
preorder :: BTree -> [Int]
preorder Nil             = []
preorder (Node tl x tr)  = [x] ++ preorder tl ++ preorder tr

postorder :: BTree -> [Int]
postorder Nil            = []
postorder (Node tl x tr) = postorder tl ++ postorder tr ++ [x]
```

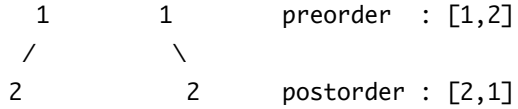
Here are some examples:

```
    4        inorder   : [1,2,3,4,5,6]
   / \
  2   5      preorder  : [4,2,1,3,5,6]
 / \   \
1   3   6    postorder : [1,3,2,6,5,4]


    3        inorder   : [1,2,3,4,5,6]
   / \
  2   5      preorder  : [3,2,1,5,4,6]
 /   / \
1   4   6    postorder : [1,2,4,6,5,3]
```

In general, one cannot uniquely recover the tree from one traversal. For example. for both the following trees, the inorder traversal is `[1,2,3,4,5,6]`.



One can show similar examples for preorder and postorder traversals as well. Sometimes it is not possible to recover the original tree even from two different traversals. Here is an example where two trees have the same preorder and postorder traversals, respectively.



But the good news is that we can always reconstruct a tree from its preorder and inorder traversals, or from its inorder and postorder traversals. For instance, suppose `[6,5,4,3,2,1]` and `[3,5,6,4,2,1]` are the given inorder and preorder traversals respectively. From the preorder traversal, we know that the root is `3`. From the inorder traversal, we know that the inorder traversal of the left subtree is `[6,5,4]`. These are three elements, and the next three elements after `3` in the preorder traversal are `[5,6,4]`. So we need to recursively form the tree with inorder traversal `[6,5,4]` and preorder traversal `[5,6,4]`. Similarly find the tree with inorder traversal `[2,1]` and preorder traversal `[2,1]`. One can similarly work out the procedure when the inorder and postorder traversals are provided.

It is not the case that for any two lists `xs` and `ys`, we can find a tree whose inorder traversal is `xs` and preorder traversal is `ys`. More specifically, if `xs` and `ys` have different lengths or if the sets represented by `xs` and `ys` are different, there is no tree whose inorder traversal is `xs` and preorder traversal is `ys`. These two problem situations may occur during recursive calls, so one needs to craft the procedure carefully.

> **Problem 1a:** Define a function `reconstructFromInPre :: [Int] -> [Int] -> Maybe BTree` that reconstructs a binary tree from its inorder traversal (first argument) and preorder traversal (second argument), if it exists For example:

```
reconstructFromInPre [] [] = Just Nil
reconstructFromInPre [1] [1] = Just (Node Nil 1 Nil)
reconstructFromInPre [1] [2] = Nothing
reconstructFromInPre [1,2,3] [1,2,4] = Nothing
reconstructFromInPre [1,2,3] [1,2,3,4] = Nothing
reconstructFromInPre [1..10] [5,4,3,7,8,2,6,1,9,10] = Nothing
reconstructFromInPre [1,2,3] [1,2,3] =
            Just (Node Nil 1 (Node Nil 2 (Node Nil 3 Nil)))
```

```
reconstructFromInPre [6,5,4,3,2,1] [3,5,6,4,2,1] =
            Just (
            Node
            (Node (Node Nil 6 Nil) 5 (Node Nil 4 Nil))
            3
            (Node Nil 2 (Node Nil 1 Nil))
            )
```

Problem 1b: Define a function `reconstructFromInPost :: [Int] -> [Int] -> Maybe BTree` that reconstructs a binary tree from its inorder traversal (first argument) and postorder traversal (second argument). For example:

```
reconstructFromInPost [] [] = Just Nil
reconstructFromInPost [1] [1] = Just (Node Nil 1 Nil)
reconstructFromInPost [1] [2] = Nothing
reconstructFromInPost [1,2,3] [1,2,4] = Nothing
reconstructFromInPost [1,2,3] [1,2,3,4] = Nothing
reconstructFromInPost [1..10] [5,4,3,7,8,2,6,1,9,10] = Nothing
reconstructFromInPost [1,2,3] [1,2,3] =
            Just (Node (Node (Node Nil 1 Nil) 2 Nil) 3 Nil)
reconstructFromInPost [6,5,4,3,2,1] [6,4,5,1,2,3] =
            Just (
            Node
            (Node (Node Nil 6 Nil) 5 (Node Nil 4 Nil))
            3
            (Node Nil 2 (Node Nil 1 Nil))
            )
```

2. A *red-black tree* is a binary search tree where each node has a color – either **Red** or **Black**. The declaration is as follows:

```
data Color = Red | Black
    deriving (Eq,Ord,Show)
data RBTree a = RBNil | RBNode (RBTree a) a Color (RBTree a)
    deriving (Eq,Ord,Show)
```

In addition to satisfying the search tree property, a red-black tree also satisfies the following two *balance invariants*.

Balance Invariant 1: No red node has a red child.

Balance Invariant 2: Every path from root to a leaf contains the same number of black nodes.

These conditions together ensure that the longest path in the tree is at most twice as long as the shortest path. As a consequence, we can prove that a red-black tree with n nodes has height at most $2\lfloor \log(n+1) \rfloor$.

> **Problem 2a:** Define a function `isRedBlack :: Ord a => RBTree a -> Bool` that tests if the input is a red-black tree. (You should check both the search tree property and the balance invariants.)
>
> **Problem 2b:** Define a function `member :: Ord a => RBTree a -> a -> Bool` that checks if the input value is in the red-black tree given as input.

3. Write a program `infList :: [Integer]` that generates an infinite list of numbers with the following properties:

   (a) The list is in strictly increasing order

   (b) The list begins with the number 1

   (c) If the list contains the number x, then it also contains the numbers $2x$, $3x$, $5x$

   (d) The list contains no other numbers

   For example, `take 20 infList` produces the following output.

```
[1,2,3,4,5,6,8,9,10,12,15,16,18,20,24,25,27,30,32,36]
```