CIS 623 Exercises 7: IO

Complete this by class time Monday March 26

Background Reading

Either of:

- Chapters 9–12 of *Learn You a Haskell for Great Good* (online version)
- Chapters 8–13 *Learn You a Haskell for Great Good* (print version)

suffice for the following.

Problems

◆ ***Problem 1*** ◆

Write a Haskell program that reads a file from standard input and prints the mod256 check sum of the file. Suppose the executable file is called `check1`. Then running it on Unix might look like the following.[1]

```
unxPrompt% ./check1
This is a line of text
and this is another one
^D
The check sum is 69
unxPrompt%
```

◆ ***Problem 2*** ◆

Same as above except that the program takes the name of the file to check from the command line.[2] Suppose the executable file is called `check2`. Then running it on Unix might look like the following.

```
unxPrompt% ./check2 file.txt
The check sum of file.txt is 6022.
unxPrompt%
```

◆ ***Problem 3*** ◆

Write a program that repeatly reads `Float`s, one per line, until `0.0` is read; then the program prints the average of the numbers.[3]

◆ ***Problem 4*** ◆

Write a Haskell program that takes the name of a file from the command line and then produces a sorted word count from the input file.[4] For example, for a file sample.txt:

```
This is line one
and this is line one plus one
```

[1] The text I typed is in **blue-bold.**

[2] We will not worry about excepts for this exercise.

[3] Useful functions for this include `read`.

[4] Useful functions for this include: `getArgs`, `group`, `map`, `putStrLn`, `readFile`, `sort`, `unlines`, and `words`.

Then running the program on sample.txt might look like:

```
unxPrompt% ./wrdcount sample.txt
and 1
is 2
line 2
one 3
plus 1
this 2
unxPrompt%
```

You may assume that the file consists of just letters and whitespace characters.

◆ ***Problem*** *5* ◆

Suppose we represent binary trees via

```
data BinTree a = Empty | Branch a (BinTree a) (BinTree a)
                 deriving (Show,Eq)
```

Definition: A *binary search tree* (abbreviated: *BST*) is a binary tree with the property that, for each node `(Branch k tl tr)`:

- the labels in the left subtree (`tl`) are $< k$ and
- the labels in the right subtree (`tr`) are $> k$.

Our BST's will have no repeated values.

Below, do not assume a `(BinTree a)` is a BST unless we state it is.

(a) Write functions

```
preord, inord, postord :: BinTree a -> [a]
```

that given a `(BinTree a)`, returns the list of labels in, respectively, preorder, inorder, and postorder.[5]

[5] See https://en.wikipedia.org/wiki/Tree_traversal

(b) Write a function

```
isBST :: (Ord a) => (BinTree a) -> Bool
```

that tests if a `(BinTree a)` is a BST.

(c) Write a function

```
insrt :: (Ord a) => (BinTree a) -> a -> (BinTree a)
```

such that, given that t is a BST, `(insrt t v)` returns an updated version of t with the value v inserted inorder. When t already contains v, then the function simply returns t.

(d) Write a function

```
postRebuild :: (Ord a) => [a] -> (BinTree a)
```

such that, given postorder listing of the labels of a BST, rebuilds the tree.[6] I.e., given a BST t, you should have:

[6] Using `foldr`, this is a one-line program.

(postRebuild (postord t)) == t

(e) Write a function

```
preRebuild :: (Ord a) => [a] -> (BinTree a)
```

such that, given preorder listing of the labels of a BST, rebuilds the tree.[7] I.e., given a BST t, you should have:

(preRebuild (preord t)) == t

[7] Using `foldl` and `flip`, this is another one-line program.

(f) Write a function

```
rebuild :: [a] -> [a] -> (BinTree a)
```

such that, given preorder and inorder listings of the labels of a `(BinTree a)` t, reconstructs t, where t is a `(BinTree a)` with no repeated labels.[8] I.e., given a such a t, you should have:

(rebuild (preord t) (inord t)) == t

[8] This t is ***not*** necessarily a BST.

Hint: Think about what information you need to set up the next recursions.