CSE 262: Programming Languages

Fall 2016

Homework 3: **Due on October 20th at 9pm on CourseSite.**

READINGS: Learn You a Haskell: Chapter 5. Real World Haskell: Chapter 3.

SUBMISSION REQUIREMENTS: Submit a text file (.hs/.txt) of your code. **Do not submit a screenshot** of it, you will not receive credit if you do.

NOTES: Argument order of functions must be in the order specified by the question definition.

1. (10 points): The price of turtles starts at x dollars (whole number amount) and increases by a factor of y each day (y must be greater than or equal to 1.0). Using recursion, determine the price of turtles after z days. Call this priceEval.

```
priceEval 25 1.2 0 -> 25.0
priceEval 25 1.2 1 -> 30.0
priceEval 25 1.2 2 -> 36.0
priceEval 40 1.2 2 -> 57.599999999999994
priceEval 40 1.2 5 -> 99.53279999999998
priceEval 25 0.8 0 -> *** Exception: Turtles never lose value
```

2. (10 points): Given a list of Ints, recursively report the sum of the list if it contains exactly 3 odd numbers. Otherwise, report the "quantity of odd numbers" minus the "quantity of even numbers". Call this numWeird.

```
numWeird [1..7] -> 1
numWeird [1..6] -> 21
numWeird [2..7] -> 27
numWeird [2,4..20] -> -10
```

3. For these problems, use a List datatype defined with the Cons and Nil structure as shown in class.

The following code provides the operator +++ that you can use to append lists as you would the ++ operator.

```
(+++) :: List a -> List a -> List a
(+++) x Nil = x
(+++) Nil x = x
(+++) (Cons a b) e@(Cons c d) = (Cons a (b +++ e))
```

Note that +++ is really just acting as a name of a function here. This is meant to give you an idea of how to use these user defined data types.

(a) (10 points): Create a recursive function called makeList, it takes a list and uses it to create a List. Order matters.

```
makeList [3,4,2] = Cons 3 (Cons 4 (Cons 2 Nil))
```

(b) (10 points): Create a recursive function called maxList, it returns the maximum value of the List. If the List is empty, report "Empty List" as an error.

```
maxList Nil -> *** Exception: Empty List
maxList (Cons 4 Nil) -> 4
maxList (Cons 2 (Cons 7 (Cons 3 (Cons 4 Nil)))) -> 7
```

(c) (10 points): Create a recursive function called removeList. Given an element and a list, return the resulting list without the first occurrence of that element.

```
removeList 5 (makeList [3,4,5,6,7,5,5,5]) ->
    Cons 3 (Cons 4 (Cons 6 (Cons 7 (Cons 5 (Cons 5 (Cons 5 Nil))))))
removeList 3 Nil -> Nil
```

(d) (10 points): Same as previous but call it removeListAll and remove all occurrences of that element.

```
removeListAll 5 (makeList [3,4,5,6,7,5,5,5]) ->
    Cons 3 (Cons 4 (Cons 6 (Cons 7 Nil)))
removeListAll 3 Nil -> Nil
```

4. For the following problems, you will create a Tree datatype using our Node (Node a Tree a Tree a) and Empty representation and deriving Show. This tree must be a Binary Search Tree, use less than or equal to for left subtrees. Create the following recursive functions to manipulate this Tree.

   (a) (10 points): Create the recursive function insertTree, it takes a value to insert and a Tree of the same type. Insert that data into the tree while maintaining the Binary Search Tree invariant.

```
insertTree 5 Empty -> Node 5 Empty Empty
insertTree 6 (insertTree 5 Empty) ->
    Node 5 Empty (Node 6 Empty Empty)
insertTree 7 (insertTree 6 (insertTree 5 Empty)) ->
    Node 5 Empty (Node 6 Empty (Node 7 Empty Empty))
insertTree 4 (insertTree 7 (insertTree 6 (insertTree 5 Empty))) ->
    Node 5 (Node 4 Empty Empty) (Node 6 Empty (Node 7 Empty Empty))
```

   (b) (10 points): Create the recursive function findTree, it takes a value to be searched for and a Tree of the same type. Return True if the data exists in the Tree, otherwise False.

```
findTree 4 Empty -> False
findTree 4 (insertTree 2 Empty) -> False
findTree 4 (insertTree 4 (insertTree 2 Empty)) -> True
```