1. (4 points) Write a Haskell function
   \[\text{isPalindrome :: [Char] -> Bool}\]
   to recognize palindromes like "Madam, I'm Adam"
   Notice you need to make all capital letters lower case and throw out punctuation before comparing.

2. (4 points) Define a function to replace one substring with another
   \[\text{subst:: [Char] -> [Char] -> [Char] -> [Char]}\]
   The result of
   \[\text{subst "Fish & Chips and Vinegar" "Chip" "Boat" yields "Fish &Boats and Vinegar"}\]
   If there is no occurrence of the replacement string, the original string should remain unchanged.

   What happens if you enter
   \[\text{subst "Fish & Chips and Vinegar" "" ""Boat"}\]

3. Define a Haskell type for a binary search tree, given the following Haskell type for Integer binary trees
   \[\text{data Tree = Nil | Node Int Tree Tree}\]
   \[\text{deriving (Show)}\]

   Define the following functions on binary search trees.
   a) (3 points) Insert an item into a binary search tree, discarding duplicates. Call the function insertNode.
   b) (3 points) Create a tree from a list of items using foldr. Call the function makeTree.
   c) (3 points) Double all the items of a list and then insert them into a tree. Call the function insertAll2.
   d) (3 points) Return the total value of all items in a binary search tree. Call the function sumTree.
   e) (3 points) Create a list the items in the binary search tree, in descending order. Call the function listDesc.
   f) (6 points) Remove an element from the tree. Call the function deleteNode.
g) (3 points) Compute the height of the binary tree (consider a leaf node as height 1). Call the function `heightTree`.

Hint: I found it helpful to have functions which returned a constant tree.

```haskell
myTree = makeTree [4,5,24,1,52,3,26]
mytree1 = makeTree [11,15,24,14,3,5,9,8,41]
```

4. (3 points) Write a function `perms` to generate a list of all possible permutations of a list. `perms [1,2,3]` yields something like `[[1,2,3], [1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]`

5. (5 points) Generate all possible permutations of the list `[1,2,3,4,5,6,7]`. For each permutation, create a binary search tree by inserting the elements of the list in order. Compute the average height of the tree formed by the permutations.

To convert to float, you may need to use the function `fromIntegral` (in GHCI). Thus, the following computes the average of a list of integers.

```haskell
computeAvg :: (Fractional b, Integral a) => [a] -> b
computeAvg a = (fromIntegral (sum a)) / (fromIntegral (length a))
```