CS344: Programming Languages
Homework 7

Required Problems

1. (a) Define a function `addFirstA` which takes a list of integers and returns a list in which each element is the sum of the first and corresponding elements of list, without using higher-order functions. For example:

   ```
   addFirst [4,3,2,1] = [8,7,6,5]
   ```

   (b) Repeat the problem in part a and write `addFirstB`, but you should use a higher-order function.

2. Dene a function `commaSeparate :: [String] -> String` that takes a list of strings and returns a single string that contains the given strings in the order given, separated by ", ". For example,

   ```
   commaSeparate [] = ""
   commaSeparate ["a", "b"] = "a, b"
   commaSeparate ["Monday", "Tuesday", "Wednesday", "Thurssday" ]
           = "Monday, Tuesday, Wednesday, Thursday"
   ```

3. Write a function `deleteAll :: (Eq a) => a -> ([a] -> [a])` that takes an item (of a type that is an instance of the Eq class) and a list, and returns a list just like the argument list, but with the each occurrence of the item (if any) removed. For example.

   ```
   deleteAll 1 [1, 2, 3, 2, 1, 2, 3, 2, 1] = [2, 3, 2, 2, 3, 2]
   deleteAll 4 [1, 2, 3, 2, 1, 2, 3, 2, 1] = [1, 2, 3, 2, 1, 2, 3, 2, 1]
   deleteAll 3 [1, 2, 3] = [1, 2]
   ```

   Do this (a) using a list comprehension, and (b) by writing out the recursion yourself. Submit both solutions (and please call the first one aDeleteAll and the second version bDeleteAll, so you don't have to put them in separate files).

4. Write a function `deleteSecond :: (Eq a) => a -> ([a] -> [a])` that takes an item (of a type that has an == function dened for it) and a list, and returns a list just like the argument list, but with the second occurrence of the item (if any) removed. For example.

   ```
   deleteSecond 1 [1, 2, 3, 2, 1, 2, 3, 2, 1] = [1, 2, 3, 2, 2, 3, 2, 1]
   deleteSecond 4 [1, 2, 3, 2, 1, 2, 3, 2, 1] = [1, 2, 3, 2, 1, 2, 3, 2, 1]
   deleteSecond 3 [1, 2, 3] = [1, 2, 3]
   ```

5. Write a function `associated :: (Eq a) => a -> [(a,b)] -> [b]` such that associated x pairs is the list, in order, of the second elements of pairs in pairs, whose rst element is equal to the argument x. For example:

```
associated 3 [(3,4), (5,7), (3,6), (9,3)] = [4, 6]
associated 2 [(1,a), (3,c), (2,b), (4,d)] = [b]
associated c (zip [c, c ..] [1, 2 ..]) = [1, 2 ..]
```