Hand in your solutions on paper and email them to cs457acc@cs.pdx.edu. All the programs should be placed in a single .hs file, which can be the body of your email message or an attachment. It is not necessary to show evidence that you have loaded and tested your programs, but this is of course the only sensible way to make sure that you have found correct answers!

1. Write a function

   \[
   \text{subStrings :: String} \rightarrow [\text{String}]
   \]

that returns a list containing all the substrings of its argument. For example, the substrings of "abc" are "abc", "ab", "bc", "a", "b", "c", and "". Each distinct substring must appear exactly once in the result list, but the order does not matter.

(But, for extra credit, write a version whose result lists the substrings in decreasing order by length, e.g. for input "abc", output should be in the order listed above.)

2. (From original Homework 3.) Write implementations of the following functions making use of the \text{foldr} function instead of explicit recursion. (You can’t use list comprehensions either.) I’ve specified variant forms of the usual names to avoid conflicts with the existing Prelude functions.

   (a) \text{map’ :: (a} \rightarrow \text{b)} \rightarrow [\text{a}] \rightarrow [\text{b}]

   (b) \text{filter’ :: (a} \rightarrow \text{Bool)} \rightarrow [\text{a}] \rightarrow [\text{a}]

   (c) \text{(+++) :: [a]} \rightarrow [\text{a}] \rightarrow [\text{a}] (meant to act like (++)).

   (d) \text{average :: [Int]} \rightarrow \text{Float} (using only one call to foldr).

3. (From original Homework 3.) Give a simple argument showing that it is impossible to express \text{foldr} in the form

   \[
   \text{foldr f i} = \text{map g}
   \]

   where \( g = \ldots \)

4. (From original Homework 3.) Do Hudak problem 5.9. Try to write your solution as a \text{foldl} (although in fact a direct recursion may be more readable in this case).

5. Newton’s method says we can calculate \( \sqrt{x} \) by taking the limit of the following sequence of approximations: \( a_0 = 1.0, a_1 = (a_0 + x/a_0)/2.0, \ldots, a_n = (a_{n-1} + x/a_{n-1})/2.0, \ldots \) Implement a function

   \[
   \text{squareRoot :: Float} \rightarrow \text{Float}
   \]

that returns the best possible approximation to the square root of its argument using Newton’s method. (Note: You can stop taking approximations when the difference between successive terms is less than \( e \), for a suitable small value of \( e \).)