Homework 3: Lectures 4 & 5
CS 440: Programming Languages and Translators, Fall 2019
Due Mon Sep 23, 11:59 pm

What to submit
Put everything into a Haskell source file (e.g., Smith_Jones_440_hw3.hs) and submit that on Blackboard. For Problem 7, see the instructions on what to include.

Problems [50 points]
1. (12 points) Fix the code below so that it behaves as in the sample run
   :
   pgm = do
       putStrLn "Enter a list of ints > ">
       line = getLine()
       let list = read line
       putStrLn $ "Has length " ++ show length list
   :}
   Sample run: (Note extra spaces before / after / within the input list are okay)
   Prelude> pgm
   Enter a list of ints >  [2,3, 5]
   Read in, reversed: [2,3,5]
   Has length 3

2. (4 points) Write a definition for function \( \text{itr} \) where \( \text{itr} \ n \ f = (f \ . \ f \ . \ ...etc... \ . \ f) \) (n times), where \( . \) is infix function composition. E.g., \( \text{itr} \ 3 \ \text{sqrt} \ 65536 = 4.0 \). If \( n \leq 0 \), return the identity function \( \text{id} \). You are required to implement \( \text{itr} \) using a \text{fold} function on a list of \( n \) copies of \( f \) (i.e., \( [f, f, f, ...etc... f] \) \( n \) times. (Hint: List comprehension.)

3. (6 points) Write a function \( \text{dropPrefix} :: \text{Eq} \ a \Rightarrow [a] \rightarrow [a] \rightarrow \text{Maybe} \ [a] \) where \( \text{dropPrefix} \ x \ y \) checks to see if \( y \) begins with the same values as in \( x \) (i.e., for some \( z \), \( y = x ++ z \)), then \( \text{dropPrefix} \) returns \( \text{Just} \ z \), else it returns \( \text{Nothing} \). E.g., \( \text{dropPrefix} "abc" "abcd" = \text{Just} "d" \) but \( \text{dropPrefix} "abc" "abz" = \text{Nothing} \).

For Problems 4 – 6, Let’s define our own linked list datatype
   data List a = None | Node a (List a) deriving (Show)
   -- Don’t include deriving (Eq) [see Problem 6]
4. (6 points) Write a function `listEq :: Eq a => List a -> List a -> Bool` that checks for equality between two `List` values. E.g., say `x = (Node 1 (Node 2 (Node 0 None)))` and `y = (Node 1 (Node 2 None))`, then `listEq x x` = True and `listEq x y` = False.

5. (8 points) Similarly, write a function `listLE :: Ord a => List a -> List a -> Bool` that checks for the first list being less than or equal to the second list. E.g., with `x` and `y` as in part (a), `listLE y x` = True and `listLE x y` = False. (The analogy is `[1,2] ≤ [1,2,0]` but `[1,2,0]` is not ≤ `[1,2]`.)

6. (4 points) Write code to make `List a` an instance of `Eq` and also an instance of `Ord`. After the declarations from Problems 3 and 4 above, add a couple of lines

   ```haskell
   instance (Eq a) => Eq (List a) where
     -- declare == to be listEq
   ```

   Similarly, add a couple of lines to declare that if `Ord a`, then `Ord (List a)`, using `listLE` for <=.

7. [10 points] Go to Lecture 6 on Regular Expressions and read Section G "Code for Simple Regular Expressions with just Concatenation and Alternation." Study the notes and the code shown there (and repeated below) nd implement the match clause for `P_and`.

   When turning in your answer, include the code below declaring `Pattern` and the given part of `match` along with your code for the missing `P_and` clause. Don't include any of the tests in the lecture; we can run our own.

   ```haskell
   -- Regular expressions with only concatenation and alternation
   data Pattern a =
     P_const a
   | P_or [Pattern a]
   | P_and [Pattern a] deriving (Eq, Read, Show)

   match :: Eq a => Pattern a -> [a] -> Maybe [a]

   -- for P_const val, check head of list for == val
   match (P_const _) [] = Nothing
   match (P_const x) (h:t) | h == x = Just t
                           | otherwise = Nothing
   ```
-- for P_or, fail if the list of patterns is empty, otherwise
-- first try matching the head pattern; if that succeeds, we’re done
-- else try the OR of the remaining patterns
match (P_or []) vals = Nothing
match (P_or (p:ps)) vals
  = case match p vals of
      Nothing -> match (P_or ps) vals
      ok @ (Just leftover) -> ok

-- (... You add P_and code ... )