Homework 2: Lectures 3 & 4
CS 440: Programming Languages and Translators, Fall 2019
Due Wed Sep 11, 11:59 pm

9/9: Q.2,5,7

How to submit
See http://cs.iit.edu/~cs440 → Homework Policies for information on working with others, how to submit, etc. If you want to submit multiple files, zip them together and submit the zipped file.

Problems [50 points]

1. (6 = 3 * 2 points) Let \( f \) \( g \) \( x \) = length (filter \( g \) \( x \)).
   a. What is the type of \( f \)?
   b. Briefly, what does this function do?
   c. Rewrite this definition using \( f = \) function composition on length and filter (and anything else?)

2. (4 = 2+2 points) Let \( f :: (a \rightarrow a \rightarrow a) \rightarrow a \rightarrow a \rightarrow a \).
   a. Rewrite \( f \cdot (2, 3) \) so that it has no syntax errors and yields 6 if \( f \) \( g \) \( x \) \( y \) = \( g \) \( x \) \( y \) [9/9]
   b. Write the definition of a function \( g :: ((a, a) \rightarrow a, (a, a)) \rightarrow a \) so that \( g \) is an uncurried version of \( f \). Calling \( g \) (uncurried_mult, (2,3)) should also yield 6. [9/9]

3. (2 points) Let \( f \) \( x \) \( y \) \( z \) = \( x \) : ( [ \( y \] ] : [ \( z \] ] ). What is the type of \( f \)?

4. (4 = 2+2 points) Let \( f \) \( x \) \( n \) = take \( n \) \( x \) == take \( n \) (drop \( n \) \( x \))
   a. Briefly (in English), what does \( f \) do on \( x \) and \( n \)?
   b. Give an example of an \( x \) for which \( f \) \( x \) 5 is true.

5. (9 = 3 * 3 points) Let’s re-implement the foldl function in multiple ways. Your foldl only needs to work on lists. [9/9]
   a. Write a definition for foldl using conditional expressions: foldl1 \( f \) \( a \) \( x \) = if \( x \) == [ ] then etc.
   b. Rewrite the definition using function definition by cases: foldl12 ...
   c. Rewrite the definition using a case expression: foldl13 \( f \) \( a \) \( x \) = case \( x \) ....
6. (15 = 10+5 points) The function \texttt{twice} should take a list and return true iff some value occurs twice in the list. E.g.,

\begin{verbatim}
> map twice [[] , [1] , [1,2] , [2,2] , [1,2,3] , [1,1,2] , [1,2,2] ]
[False,False,True,False,False,True,True,True]
\end{verbatim}

a. (10 points) What problems are there with the following code? (Give a brief human-understandable description — don’t just write out Haskell error messages. Also find the semantic errors.)

```haskell
:{
    twice :: [a] -> Bool
    twice [] = False
    twice [_] = False
    twice [x,x] = True
    twice [_,_] = False
    twice (h1 : h2 : t) = (h1 == h2 || twice h1 t)
    twice (_ ++ [x] ++ _ ++ [y] ++ _) = x == y
    :}
```

b. Rewrite \texttt{twice} so that it works correctly. (Maintain the function definition by cases style. If you have to drop a clause, so be it.)

c. (2 points extra credit) Write a definition by cases for twice that only has two cases (one recursive, one not).

7. (8 points) (Fun with lambda functions)

a. Rewrite \(f \ g \ x \ y = g \ x \ (y \ x)\) \[9/9\] three ways, first \(f \ g \ x = \) unnamed lambda function, then \(f \ g = \) unnamed lambda function, and finally \(f = \) unnamed lambda function'.

b. Briefly, why does \(\text{var} = \text{unnamed lambda function}\) illustrate the concept of first-class functions?

8. (2 points) Consider the following claim: “A Haskell function is higher order if and only if its type has more than one arrow.” Is this correct? Give a brief argument.

\footnote{You’ll want to look up information in the \textit{Learn You a Haskell} ... book}