CS 340 Spring 2019
Midterm Exam

Instructions:

• This exam is closed-book, closed-notes. Electronic devices of any kind are not permitted.

• Write your final answers, tidily, in the boxes provided. Scratch paper is attached at the end of the exam.

<table>
<thead>
<tr>
<th>1 (8) :</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (12) :</td>
</tr>
<tr>
<td>3 (9) :</td>
</tr>
<tr>
<td>4 (9) :</td>
</tr>
<tr>
<td>5 (10) :</td>
</tr>
<tr>
<td>TOTAL (48) :</td>
</tr>
</tbody>
</table>
1. Function Type Declarations (8 points):

For each of the following function definitions, correctly complete the preceding type declaration. Be sure to include any necessary class constraints.

(A)

    mystery1 ::
    mystery1 g h = h . g

(B)

    mystery2 ::
    mystery2 x [] = x
    mystery2 x (y:ys) | x < y = mystery2 x ys
                      | otherwise = mystery2 y ys

(C)

    mystery3 ::
    mystery3 _ [] _ = []
    mystery3 f (x:xs) y = f x y : mystery3 f xs y

(D)

    mystery4 ::
    mystery4 [] _ = 0
    mystery4 _ [] = 0
    mystery4 (x:xs) (y:ys) = x + y + mystery4 xs ys
2. Basic Recursion (12 points):

Refer to the following function descriptions and sample call(s)/result(s), and implement them on the following page using explicit recursion. You may use only the following built-in functions in your implementation:

- basic arithmetic (+, -, *, /)
- list construction (·, [], ++, list comprehensions)
- head, tail, take, drop, null

(A) listsOf, which takes a value and returns an infinite list of lists, the first one empty, and each successive list containing one more copy of the value.

\[
\begin{align*}
> & \text{take 5 $ listsOf 1} \\
& [[]],[1],[1,1],[1,1,1],[1,1,1,1]] \\
& > \text{take 10 $ listsOf 'x'} \\
& ["","x","xx","xxx","xxxx","xxxxx","xxxxxx","xxxxxxx","xxxxxx"x"]
\end{align*}
\]

(B) revUntil, which takes a number \( n \) and a list, and returns the contents of the list reversed, but only up to (and not including) index \( n \).

\[
\begin{align*}
> & \text{revUntil 0 "hello"} \\
& \"hello\" \\
> & \text{revUntil 3 "abcdefghi"} \\
& \"cbadefghi\" \\
> & \text{revUntil 10 "abcdefghi"} \\
& \"ihgfedcba\"
\end{align*}
\]

(C) takeEvery, which takes a number \( n \) and a list, and returns the list composed of every \( n \)th value from the list.

\[
\begin{align*}
> & \text{takeEvery 0 "abcdefghi"} \\
& \"\" \\
> & \text{takeEvery 1 "abcdefghi"} \\
& \"abcdefghi\" \\
> & \text{takeEvery 3 "abcdefghi"} \\
& \"cfi\"
\end{align*}
\]
listsOf :: a -> [[a]]

revUntil :: Int -> [a] -> [a]

takeEvery :: Int -> [a] -> [a]
3. Higher Order Functions (9 points):

(A) How could you express the list comprehension \[ \{ f \ x \mid x \leftarrow xs, p \ x \} \] using the higher-order functions map and filter?

(B) Using foldl, define the function \lst2int :: [Integer] -> Integer, which converts a list representation of a number into an integer. E.g.,

\begin{verbatim}
> lst2int [4, 1, 2, 5]
4125
\end{verbatim}

\begin{verbatim}
lst2int =
\end{verbatim}

(C) The following function, \textit{mklist}, expresses a pattern of recursion for generating lists:

\begin{verbatim}
  mklist \ p \ h \ t \ x | p \ x = []
  | otherwise = h \ x \ : \ mklist \ p \ h \ t \ (t \ x)
\end{verbatim}

where the predicate \textit{p} determines when the list terminates, and the functions \textit{h} and \textit{t} are called on the last argument to give the head and tail of the list at each step of the recursion. Define \textit{map} using \textit{mklist}.

\begin{verbatim}
  map \ f =
\end{verbatim}
4. Evaluating Folds (9 points):

Show the result of evaluating each of the following expressions involving either `foldr` or `foldl`.

(A) \[
\text{foldl iter (0,[]) \{1..10\} where iter (n,ys) x | even x = (n+x, ys) \\
| otherwise = (n, x:ys)}
\]

(B) \[
\text{foldr iter (True,[]) "brownfox" where iter x (b,r) = if b then (not b, x:r) else (not b, r++[x])}
\]

(C) \[
\text{foldr (\x g -> (\n -> g (x:n))) id [1..10] []}
\]
5. Laziness and Infinite Lists (10 points):
Each of the following presents a function definition and an invocation of that function on an infinite list. If the function returns, write the return value in the space provided; if the function doesn’t return (i.e., it gets stuck processing the infinite list), write “FAIL” instead.

(A) inf1 f n (x:xs) |
  | f x && n == 0 = x
  | f x = inf1 f (n-1) xs
  | otherwise = inf1 f n xs

> inf1 even 20 [1..]

(B) inf2 g = foldr (\x ys -> if g x then [x] else ys++[x]) []

> inf2 (==10) [1..]

(C) inf3 n = foldl (\ys x -> if length ys > n then ys else x:ys) []

> inf3 4 [1..]

(D) inf4 = foldl (+) 0 . take 5 . drop 5

> inf4 [1..]

(E) inf5 h = sum . take 5 . map h

> inf5 (*2) [1..]