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.

1 (/8) :
2 (/12) :
3 (/9) :
4 (/9) :
5 (/10) :
TOTAL (/48) :
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 :: (a -> b) -> (b -> c) -> a -> c
mystery1 g h = h . g

(B)  
mystery2 :: Ord a => a -> [a] -> a
mystery2 x [] = x
mystery2 x (y:ys) | x < y = mystery2 x ys
| otherwise = mystery2 y ys

(C)  
mystery3 :: (a -> b -> c) -> [a] -> b -> [c]
mystery3 _ [] _ = []
mystery3 f (x:xs) y = f x y : mystery3 f xs y

(D)  
mystery4 :: Num a => [a] -> [a] -> a
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.

> take 5 $ listsOf 1
[[],[1],[1,1],[1,1,1],[1,1,1,1]]

> take 10 $ listsOf 'x'
["","x","xx","xxx","xxxx","xxxxx","xxxxxx","xxxxxxx","xxxxxxxx","xxxxxxxxx"]

(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.

> revUntil 0 "hello"
"hello"

> revUntil 3 "abcdefghi"
"cbadefghi"

> revUntil 10 "abcdefghi"
"ihgfedcba"

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

> takeEvery 0 "abcdefghi"
"

> takeEvery 1 "abcdefghi"
"abcdefghi"

> takeEvery 3 "abcdefghi"
"cfi"
listsOf :: a -> [[a]]
listsOf x = [repeat n x | n <- [0..]]
    where repeat 0 x = []
           repeat n x = x : repeat (n-1) x

revUntil :: Int -> [a] -> [a]
revUntil n xs = reverse (take n xs) ++ drop n xs
    where reverse [] = []
           reverse (x:xs) = reverse xs ++ [x]

takeEvery :: Int -> [a] -> [a]
takeEvery _ [] = []
takeEvery 0 _ = []
takeEvery n xs = let ys = drop (n-1) xs
                    in if null ys then [] else head ys : takeEvery n (tail ys)
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 \(\text{map}\) and \(\text{filter}\)?

\[
\text{map } f \ $ \ \text{filter } p \ xs
\]

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

\[
> \text{lst2int} \ [4, \ 1, \ 2, \ 5] \\
4125
\]

\[
\text{lst2int} = \text{foldl} \ (\lambda x \rightarrow r*10+x) \ 0
\]

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

\[
\text{mklist } p \ h \ t \ x \mid p \ x = [] \\
\mid \text{otherwise} = h \ x : \text{mklist } p \ h \ t \ (t \ x)
\]

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

\[
\text{map } f = \text{mklist } \text{null} \ (f \ . \ \text{head} \ \text{tail})
\]
4. Evaluating Folds (9 points):
Show the result of evaluating each of the following expressions involving either `foldr` or `foldl`.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>(A)</strong></td>
<td><code>foldl</code> iter (0,[]) [1..10] where iter (n,ys) x</td>
<td>Ans: (30,[9,7,5,3,1])</td>
</tr>
<tr>
<td></td>
<td></td>
<td>even x = (n+x, ys)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>otherwise = (n, x:ys)</td>
</tr>
<tr>
<td><strong>(B)</strong></td>
<td><code>foldr</code> iter (True,[]) &quot;brownfox&quot; where iter x (b,r) = if b then (not b, x:r) else (not b, r++[x])</td>
<td>Ans: (True,&quot;rwfxonob&quot;)</td>
</tr>
<tr>
<td><strong>(C)</strong></td>
<td><code>foldr</code> (\x g -&gt; (\n -&gt; g (x:n))) id [1..10] []</td>
<td>Ans: [10,9,8,7,6,5,4,3,2,1]</td>
</tr>
</tbody>
</table>
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) \[ \text{inf1} \ f \ n \ (x:x:s) \mid f \ x \land n = 0 = x \]
\[ \mid f \ x = \text{inf1} \ f \ (n-1) \ x s \]
\[ \mid \text{otherwise} = \text{inf1} \ f \ n \ x s \]

\[ \text{Ans: 42} \]

(B) \[ \text{inf2} \ g = \text{foldr} \ \lambda(x \ \mathrm{ys} \rightarrow \text{if } \ g \ x \ \text{then } [x] \ \text{else } \mathrm{ys}++[x]) \ [] \]

\[ \text{Ans: [10,9,8,7,6,5,4,3,2,1]} \]

(C) \[ \text{inf3} \ n = \text{foldl} \ \lambda(\mathrm{ys} \ x \rightarrow \text{if } \text{length ys} > n \ \text{then } \mathrm{ys} \ \text{else } x:\mathrm{ys}) \ [] \]

\[ \text{Ans: FAIL} \]

(D) \[ \text{inf4} = \text{foldl} \ (+) \ 0 . \ \text{take} \ 5 . \ \text{drop} \ 5 \]

\[ \text{Ans: 40} \]

(E) \[ \text{inf5} \ h = \text{sum} . \ \text{take} \ 5 . \ \text{map} \ h \]

\[ \text{Ans: 30} \]