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"
```

```haskell
listsOf :: a -> [[a]]
listsOf x = iter []
  where iter y = y : iter (x:y)
```

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

```haskell
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 | x <- xs, p x]` using the higher-order functions `map` and `filter`?

```
map f $ filter p xs
```

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

```
> lst2int [4, 1, 2, 5]
4125
```

```
lst2int = foldl (\r x -> r*10+x) 0
```

(C) The following function, `mklist`, expresses a pattern of recursion for generating lists:

```
mklist p h t x | p x = []
               | otherwise = h x : 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 `map` using `mklist`.

```
map f = mklist null (f . head) tail
```

4. Evaluating Folds (9 points):

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

(A)

```
foldl iter (0,[]) [1..10]
  where iter (n,ys) x | even x = (n+x, ys)
                      | otherwise = (n, x:ys)

Ans: (30,[9,7,5,3,1])
```

(B)

```
foldr iter (True,[]) "brownfox"
  where iter x (b,r) = if b then (not b, x:r) else (not b, r++[x])

Ans: (True,"rwfxonob")
```

(C)

```
foldr (\x g -> (\n -> g (x:n))) id [1..10] []

Ans: [10,9,8,7,6,5,4,3,2,1]
```

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..]

Ans: 42
```

(B)

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

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

Ans: [10,9,8,7,6,5,4,3,2,1]
```

(C)

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

> inf3 4 [1..]

Ans: FAIL
```

(D)

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

> inf4 [1..]

Ans: 40
```

(E)

```
inf5 h = sum . take 5 . map h

> inf5 (*2) [1..]

Ans: 30
```