CMPT 340

Assignment 3 Due: Tuesday, March 7, 2017, 11:59pm

Higher-Order Functions and Lazy Evaluation

Total: 100 Points

[Note: For each of the following problems you must explicitly include the function type signature.]

Problem 1 [15 + 15 Points]. A higher-order function unfold can be defined as follows to encapsulate a pattern of recursion for producing a list:

```
unfold p h t x    | p x          = [ ]
                  | otherwise    = h x : unfold p h t (t x)
```

That is, the function *unfold p h t* produces the empty list if the predicate *p* is true of the argument, and otherwise produces a non-empty list by applying the function *h* to give the head, and the function *t* to generate another argument that is recursively processed in the same way to produce the tail of the list. For example, a function*int2bin* (to convert integers to binary numbers) can be written as follows:

```
int2bin  =  unfold (==0) (`mod`2) ('div`2)
```

[Note: putting function names mod and div inside back quotes allows them to be used infix]

Define the following functions using *unfold*:

a) *map f*

b) *iterate f*, where *iterate f x* produces a list by applying the function f to x an increasing number of times, as follows:

```
iterate f x = [x, f x, f (f x), f (f (f x)), ... ]
```

Problem 2 [20 Points]. Define a function altMap, which takes two functions (instead of the one for map) to apply to a list.   altMap alternately applies its two argument functions to the successive elements of the list, the first function to the first element, the second function to the second element, and so on. For example, altMap (+10) (+100) [0, 1, 2, 3, 4] evaluates to [10, 101, 12, 103, 14]

Problem 3 [10 Points]. Using altMap, define the function luhn :: [Int] -> Bool for the Luhn algorithm, which was assigned for Assignment 1.

Problem 4 [20 Points]. Show how the single comprehension *[(x, y) | x <- [1, 2, 3], y <- [4, 5, 6]]* with two generators can be alternatively expressed using two comprehensions with single generators. [Hint: make use of the library function *concat :: [[a]] -> [a]*, and nest one comprehension within the other.]

Problem 5 [20 Points]. A positive integer is perfect if it equals the sum of all of its factors, excluding the number itself. For example, 6, 28, 496, and so on. Define a list comprehension to generate an infinite list of perfect numbers. Do not make external function calls. Try to be efficient in the search for factors. [Hint: One way to do this is by (1) placing a comprehension for finding factors in a qualifier of the the top-level comprehension, (2) adding the factors using an appropriate higher-order function, and (3) checking to see whether the sum equals the number.]

Submission:

Create a directory with your nsid as its name. Inside this directory, create separate sub-directories with names like problem2, problem3, etc. for each programming problem. Under each programming problem's folder, include a file with your program, as well as a text file showing a transcript of your testing of the program.

Once you have everything in your directory, create a zip file for the entire directory. If your nsid is <your_nsid> name the zip file <your_nsid>.zip. When opened, it should create a directory called <your_nsid>.

You may submit multiple times before the deadline, so you are advised not to wait till the last minute to submit your assignment to avoid system slowdown. You are encouraged to submit completed parts of the assignment early. Late submissions will not be accepted.