Coverage & Logistics

This homework covers material through the fifth chapter of *Haskell: The Craft of Functional Programming* (HCFP).

This homework is officially due in class on **Thursday, February 9**, but it comes with an automatic extension: anything submitted to the CIS 252 bin near CST 4-226 by **noon on Friday, February 10** will be accepted as being on time.

You may work singly or in pairs on this assignment.

What to turn in

The same general grading criteria from Assignment 2 applies for this assignment as well. You should submit hard copies of (1) your source code and (2) a clean transcript demonstrating convincingly that your code is correct. As always, include a completed disclosure cover sheet.

In writing your functions, make sure to abide by the following constraints:

(i) All of your functions should be **recursive** functions.

Although these functions can be written without recursion, the purpose of this assignment is to give you practice with recursion.

(ii) ***Do not use*** `++` to construct lists: use `:` instead.

Exercises

You may use any code from lecture that you wish: *in fact, you're encouraged to do so.* However, you should include a note in your comments indicating that you are doing so and specifying which functions you are reusing.

1. Write a **recursive** Haskell function

   ```
   squarePairs :: Int -> Integer -> [(Integer,Integer)]
   ```

   such that `squarePairs n i` returns a list of `n` pairs of the form (x, x^2); the value of x in the first pair is `i`, and the value increases by 1 in each subsequent pair. If `n` is negative, the empty list is returned.

   For example, `squarePairs 4 (-1)` returns `[(-1,1),(0,0),(1,1),(2,4)]` and `squarePairs 5 4` returns `[(4,16),(5,25),(6,36),(7,49),(8,64)]`.

2. Write a **recursive** Haskell function

   ```
   countDownBy :: Int -> Int -> Int -> [Int]
   ```

   such that `countDownBy m n diff` generates the list of values that starts with `m`, each subsequent value is obtained by subtracting `diff`, and the list ends when the next value would be less than `n`; when `m` is less than `n`, the function returns the empty list.

   For example, `countDownBy 17 2 3` returns `[17,14,11,8,5,2]`, and `countDown 16 2 3` returns `[16,13,10,7,4]`.

3. Write a **recursive** Haskell function

   ```
   steps :: Int -> Int -> [[Int]]
   ```

   such that `steps m n` returns a list containing `n-m+1` lists (provided that `m` is less than or equal to `n`): the i^{th} list is the sequence of values from `m` to $\texttt{m} + i - 1$. (If `m` is greater than `n`, the empty list is returned.)

   For example, codesteps 7 3 returns `[[]]`, and `steps 3 7` returns `[[3],[3,4],[3,4,5],[3,4,5,6],[3,4,5,6,7]]`.

   Hints: Use a helper function that takes a parameter (say, `i`) that indicates which list (e.g., the i^{th}) is currently being constructed, and make use of a function from lecture.

4. Write a **recursive** Haskell function

   ```
   indexChar :: Int -> Int -> Char -> String
   ```

   such that `indexChar n i c` returns a string of length `n` (i.e., a list of `n` characters): the `i`th character is `'!'`, and every other character is `c`. (If `n` is negative, the empty list/string should be returned. If `i` is negative or larger than `n`, the resulting list contains only instances of `c`.)

   For example, `indexChar 7 2 'w'` returns `"w!wwwww"`, whereas `indexChar 7 11 'w'` returns `"wwwwwww"`.

5. Write a **recursive** Haskell function

   ```
   diag :: Int -> Char -> [String]
   ```

   such that `diag n c` returns a list of `n` strings of length `n`: the `i`th string primarily contains copies of `c`, except that it contains `'!'` in the `i`th position. If `n` is negative, the empty list is returned.

   For example, `diag 7 'w'` returns `["!wwwwww","w!wwwww","ww!wwww","www!www","wwww!ww","wwwww!w","wwwwww!"]`.

   Hint: Use a helper function as in Exercise 3.