CSCI 365 Module 7

Due Wednesday 13 April, 2022

Specification

- To receive **Level 1** credit for this module:
  - Complete all exercises (1–4).
  - You **must not** write any explicitly recursive definitions. All recursion must be accomplished through the use of standard library functions.
- To receive **Level 2** credit for this module:
  - Complete all requirements for Level 1.
  - Additionally ensure that your solution to each exercise uses no more than 200 characters in total (not counting whitespace, comments, or type signatures).

Standard library functions you may find helpful on this module include: `map`, `filter`, `foldr`, `zip`, `zipWith`, `iterate`, `repeat`, `replicate`, `take`, `drop`, `takeWhile`, `dropWhile`, `and`, `or`, `any`, `all`, `concat`, `concatMap`, `scanl` (but there are also many others that may be helpful; feel free to also use any others you find). Remember that you cannot use explicit recursion on this module!

Wholemeal programming

Exercise 1 Rewrite each function below as a pipeline of standard library functions, without any explicit recursion. For example, given the definition

```
f :: [Int] -> [Int]
f [] = []
f (x:xs) = x+1 : f xs
```

you should rewrite it as `f = map (+1)`.

(a)
```
g :: [Int] -> Int
g [] = 0
g (x:xs)
  | x < 5 = g xs
  | otherwise = (8*x) + g xs
```

(b)

```
h :: String -> Bool
h str = q 'A' str
  where
    q _ [] = True
    q cur (c:cs)
      | cur > c   = False
      | otherwise = q (succ cur) cs
```

Exercise 2

Write a function

```
skips :: [a] -> [[a]]
```

The output of `skips` is a list of lists. The first list in the output should be the same as the input list. The second list in the output should contain every second element from the input list. . . and the *n*th list in the output should contain every *n*th element from the input list.

For example:

```
skips "ABCD"       == ["ABCD", "BD", "C", "D"]
skips "hello!"     == ["hello!", "el!", "l!", "l", "o", "!"]
skips [1]          == [[1]]
skips [True,False] == [[True,False], [False]]
skips []           == []
```

Note that the output should be the same length as the input.

Exercise 3

A *local maximum* of a list is an element of the list which is strictly greater than both the elements immediately before and after it. For example, in the list `[2,3,4,1,5]`, the only local maximum is `4`, since it is greater than the elements immediately before and after it (`3` and `1`). `5` is not a local maximum since there is no element that comes after it.

Write a function

```
localMaxima :: [Integer] -> [Integer]
```

which finds all the local maxima in the input list and returns them in order. For example:

```
localMaxima [2,9,5,6,1] == [9,6]
localMaxima [2,3,4,1,5] == [4]
localMaxima [1,2,3,4,5] == []
```

Exercise 4

For this task, write a function

```
histogram :: [Integer] -> String
```

which takes as input a list of `Integer`s between 0 and 9 (inclusive), and outputs a vertical histogram showing how many of each number were in the input list. You may assume that the input list does not contain any numbers less than zero or greater than 9 (that is, it does not matter what your function does if the input does contain such numbers). Your output must exactly match the output shown in the examples below.

```
histogram [1,1,1,5] ==

 *
 *
 *   *
==========
0123456789
```

```
histogram [1,4,5,4,6,6,3,4,2,4,9] ==

    *
    *
    * *
 ******  *
==========
0123456789
```

Important note: If you type something like `histogram [3,5]` at the `ghci` prompt, you should see something like this:

```
"   * *    \n==========\n0123456789\n"
```

This is a textual *representation* of the `String` output, including `\n` escape sequences to indicate newline characters. To actually visualize the histogram as in the examples above, use `putStr`, for example, `putStr (histogram [3,5])`.