CIS 194: Homework 3

Due Monday, Feburary 4

Code golf!

This assignment is simple: there are three tasks listed below. For each task, you should submit a Haskell function with the required name and type signature which accomplishes the given task and is *as short as possible*.

Rules

- Along with your solution for each task you *must* include a comment explaining your solution and how it works. **Solutions without an explanatory comment will get a score of zero.** Your comment should demonstrate a complete understanding of your solution. In other words, anything is fair game but you must demonstrate that your understand how it works. If in doubt, include more detail.
- Comments do not count towards the length of your solutions.
- Type signatures do not count towards the length of your solutions.
- `import` statements do not count towards the length of your solutions. You may import any modules included in the Haskell Platform.
- **Whitespace does not count towards the length** of your solutions. So there is no need to shove all your code onto one line and take all the spaces out. Use space as appropriate, indent nicely, *etc.*, but otherwise try making your code as short as you can.

- You are welcome to include additional functions beyond the ones required. That is, you are welcome to break up your solutions into several functions if you wish (indeed, sometimes this may lead to a very short solution). Of course, such additional functions will be counted towards the length of your solution (excluding their type signatures).
- Your final submission should be named `Golf.hs`. Your file should define a module named `Golf`, that is, at the top of your file you should have

```
module Golf where
```

- The three shortest solutions (counting the total number of characters, excluding whitespace and the other exceptions listed above) for each task will receive two points of extra credit each. You can get up to a total of four extra credit points.
- Otherwise, the length does not really matter; long but correct solutions will receive full credit for correctness (although they may or may not get full credit for style, depending on their style).

Hints

- Use functions from the standard libraries as much as possible—that's part of the point of this assignment. Using (say) `map` is much shorter than implementing it yourself!
- In particular, try to use functions from the standard libraries that encapsulate recursion patterns, rather than writing explicitly recursive functions yourself.
- You may want to start by getting something that works, without worrying about the length. Once you have solved the task, try to figure out ways to make your solution shorter.
- If the specification of a task is unclear, feel free to ask for a clarification on Piazza.
- We will test your functions on other inputs besides the ones given as examples, so to be safe, so should you!

Tasks

Exercise 1 Hopscotch

Your first task is to 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 nth list in the output should contain every nth 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 2 Local maxima

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 3 Histogram

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])`.