COMP 356 Homework Assignment 7

Part A (6 points)

1. (3 points) Write a tail-recursive version of the WeirdFactorial function, which is defined for positive integers as follows:

$$\text{WeirdFactorial}(n) = \begin{cases} 1 \times 3 \times 5 \times \ldots \times n & \text{if } n \text{ is odd} \\ 2 \times 4 \times 6 \times \ldots \times n & \text{if } n \text{ is even} \end{cases}$$

2. (3 points) Write a tail-recursive version of the IntegerLog function. $\text{IntegerLog}_b(n)$ is identical to the usual log function $\log_b(n)$ for a number n and base b, except that (i) IntegerLog is defined only for positive integers b, n; and (ii) the result is rounded down to the nearest integer. For example:

$$\begin{aligned} \text{IntegerLog}_2(8) &= 3 \\ \text{IntegerLog}_2(20) &= 4 \\ \text{IntegerLog}_5(24) &= 1 \\ \text{IntegerLog}_{10}(7) &= 0 \\ \text{IntegerLog}_{10}(100001) &= 5 \end{aligned}$$

Your solution may only use basic integer arithmetic in Scheme (`+,-,*,quotient`). You may not use any other numerical or mathematical operations. You may assume the arguments to your functions are positive integers.

Part B (20 points)

This part of the assignment asks you to implement, in Scheme, an instance of the *map-reduce* framework popularized by Google. (Optionally, if you're interested, take a look at the 2008 paper *MapReduce: simplified data processing on large clusters*, by Jeff Dean and Sanjay Ghemawat, in Communications of the ACM, 51(1), pages 107-113.)

The input will be a corpus of web pages, represented by a Scheme list called `webpages`. Each element of this list is itself a list representing an individual web page. Specifically, a web page is specified by a list of words that occur on the page, in the order that they appear (and with repeats). For example, the input might be:

```
(define page1 (list "the" "cat" "sat" "on" "the" "mat"))
(define page2 (list "the" "cat" "sat" "on" "the" "dog"))
(define page3 (list "the" "dog" "sat" "on" "the" "cat"))
(define page4 (list "the" "dog" "sat" "on" "the" "cat" "and" "the" "mat"))
(define page5 (list "the" "cat" "in" "the" "hat" "came" "back"))

(define webpages (list page1 page2 page3 page4 page5))
```

The ultimate objective will be to compute the three highest-scoring words in the corpus of web pages, where the *score* of the words on a page is computed by a Scheme function `(score page)`. The input to `score` is a parameter `page`, which is just a list of words as in the above example. The output is a list of *word-score pairs*, one for each unique word appearing on the page. A word-score pair is a two-element list consisting of the word followed by a non-negative integer, which is the word's score. The output of `score` is guaranteed to be sorted in alphabetical order. Your final program should work correctly with any reasonable definition of the `score` function. One particularly simple but useful `score` function will be described below—you should implement this version of the `score` function, but it is recommended that you also test your code using other variants of the `score` function that you invent yourself. The specific example of a `score` function will now be described. The function computes a very simple score for each word: it is the *frequency* of the word on the page—that is, the number of times the word occurred on the page. For example, if the word "wombat" has a score of 5 when this scoring function is applied to a particular page, this means the word "wombat" occurs 5 times on the page. Hence, the output of the simple frequency-based scoring function on `page1` defined above would be

```
(list
  (list "cat" 1)
  (list "mat" 1)
  (list "on"  1)
  (list "sat" 1)
  (list "the" 2)
)
```

Your overall task, therefore, is to apply the scoring function to each page and accumulate the results so that you can compute the three highest-scoring words in the entire corpus, and output those words together with their scores (sorted in descending order by score). For example, the final output on the above `webpages` example should be:

```
(list
  (list "the" 11)
  (list "cat" 5)
  (list "on" 4)
)
```

As a hint, note that there are two overall phases in the computation, which the Google paper cited above describes as the *map* and *reduce* phases, respectively. The map phase is particularly easy, since you can use the built-in Scheme function `map` to apply the `score` function to each individual page in the `webpages` list. The reduce phase is where you aggregate the data from each individual page into a single list. Finally, you just need to extract the top three words from the reduced data. Therefore, your final computation might look like this:

```
(firstThree (reduceScores (map score webpages)))
```

Of course, you may need to define several helper functions in addition to writing the `firstThree` and `reduceScores` functions.

For full credit, your solution must use a functional style of programming. Do not use imperative data structures such as hash tables. Use only simple functional data structures such as lists, and lists of lists. You are permitted to use the built-in `sort` function to sort your lists.

Submit all code for this assignment (Parts A and B) as a single zip file to Moodle.