COMP 356 Homework Assignment 6

Acknowledgment. This assignment was written by Prof. Tim Wahls, with minor changes by John MacCormick.

Notes: 1. None of the functions in this assignment is required to be tail recursive. The next assignment will ask you to write tail recursive functions. 2. In this assignment, please use only elementary list functions described in class, together with any additional functions mentioned in hints below. In fact, the assignment can be completed using only the following built-in functions and forms: `append`, `car`, `cdr`, `cons`, `define`, `empty`, `eqv?`, `if`, `lambda`, `letrec`, `let`, `list?`, `null?`, `-`, `>`. Specifically, do not use functions like `remove` and `filter`, which would obviously make the questions below too easy. If in any doubt about which Scheme functions are permitted, please ask the instructor.

1. (3 pts) Write a Scheme function `my-gcd` that takes two integers and returns their greatest common divisor (GCD). One elegant solution uses Euclid's method as follows:

   - if x = y, then the GCD of x and y is x (or y)
   - if x > y, then the GCD of x and y is the GCD of x - y and y
   - if x < y, then the GCD of x and y is the GCD of y - x and x

   You are not allowed to use the built-in `gcd` function.

2. (5 pts) Write a Scheme function `my-delete` that takes a list and an element `e`, and returns the list with all occurrences of `e` deleted. For example:

   ```
   (my-delete '(1 2 3 1 3 1) 1)
   ```

   returns `(list 2 3 3)`. (Hint: use the built-in function `eqv?` to compare for equality so that your function will work with arbitrary lists, not just lists of integers.)

3. (7 pts) Write a Scheme function `my-flatten` that takes a (possibly nested) list, and returns a simple (unnested) list. For example,

   ```
   (my-flatten '(1 (2 3 (3 4)) 5))
   ```

   returns `(list 1 2 3 3 4 5)`. (Hint: use the built-in function `list?` that tests whether its argument is a list, and the built-in function `append` that appends (concatenates) two lists.)

4. (6 pts) Write a Scheme function `my-del-if` that takes a list and a function as arguments. The function argument should itself take one argument and return boolean (true or false). The result of a call to `my-del-if` should be the argument list with all elements that make the function argument true removed. For example:

   ```
   (my-del-if '(1 4 3 0 6 2) (lambda (x) (< x 3)))
   ```

   returns `(list 4 3 6)`.

5. (6 pts) Write a Scheme function `my-exp` that takes an integer n and returns a one argument function. The function returned should compute its argument to the nth power. For example, `((my-exp 3) 2)` should return `8`. For full credit, your `my-exp` function should not call any other functions that you have written, or any system function that performs exponentiation (exception: you can call functions defined *inside* `my-exp`). You may assume the function arguments are non-negative.

Submit your solutions to Moodle as a single file. Your functions will be graded on correctness, compliance with the above guidelines, and coding style.