CS4121 Project 4: Scheme Programming

The goal of this project is to help you learn how to use the Scheme programming language to solve problems. In this project, you are required to use the Dr. Racket, which has been installed in the lab, to finish the problems in this project.
The slack day policy can not be applied to this assignment.
In each of the following problems you can assume the following data definitions:

$$\begin{aligned} M &\rightarrow \ '() \mid (A\ .\ M) \mid (M\ .\ M) \\ L &\rightarrow \ '() \mid (A\ .\ L) \\ A &\rightarrow \ \text{atom} \end{aligned}$$

You may assume that all numbers are integers greater than or equal to 0 and that all data fits the specified format unless specified otherwise. Each problem is worth 5 points.

Give a Scheme solution to the following function specifications:

1. `(last a L)` returns the zero-based index of the last occurrence of the atom `a` in the flat list `L`, or -1 if there is no occurrence of `a` in `L`.

```
> (last 3 '(2 1 3 4))
2
> (last 3 '(2 1 3 3 4))
3
```

2. `(wrap M)` returns `M` with a pair of parentheses wrapped around each atom occuring in M.

```
> (wrap '(1 ()))
((1) (()))
> (wrap '(1 (2) 3))
((1) ((2)) (3))
```

3. `(count-parens-all M)` counts the number of opening and closing parentheses in `M`.

```
> (count-parens-all '())
2
> (count-parens-all '((a b) c))
4
```

4. `(insert-right-all new old M)` builds a list obtained by inserting the item `new` to the right of all occurrences of the item `old` in the list `M`.

```
> (insert-right-all 'z 'a '(a b (a c (a))))
(a z b (a z c (a z)))
> (insert-right-all 'dog 'cat '(my dog is fun))
(my dog is fun)
```

5. `(invert M)`, where M is a list that is a list of pairs (lists of length two), returns a list with each pair reversed.

```
> (invert '((a 1) (a 2) (b 1) (b 2)))
  ((1 a) (2 a) (1 b) (2 b))
```

6. `(filter-out pred L)` returns a flat list of those elements in `L` that do not satisfy the predicate `pred`.

```
> (filter-out number? '(a 2 #f b 7))
'(a #f b)
> (filter-out symbol? '(a 2 #f b 7))
'(2 #f 7)
```

7. (`summatrices M1 M2`) returns the sum of matrices M1 and M2. A matrix is represented as a list of rows, eachof which is a flat list of numbers.

```
> (summatrices '((1 2 3)) '((4 5 6)))
'((5 7 9))
> (summatrices '((1 2 3) (4 5 6)) '((10 10 10) (20 20 20)))
'((11 12 13) (24 25 26))
```

8. (`swapper a1 a2 M`) returns `M` with all occurrences of `a1` replaced by `a2` and all occurrences of `a2` replaced by `a1`.

```
> (swapper 'a 'd '(a b c d))
'(d b c a)
> (swapper 'x 'y '((x) y (z (x))))
'((y) x (z (y)))
```

9. (`flatten M`) returns `M` with all inner parentheses removed. `flatten` turns a list into a flat list.

```
> (flatten '(a b c))
'(a b c)
> (flatten '((a) () (b ()) () (((c)))))
'(a b c)
```

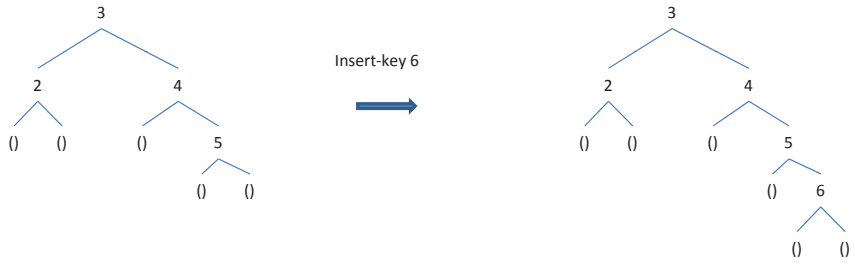

Figure 1: Binary Tree Insertion

10. Consider the following grammar for a binary tree of numbers using Scheme lists where non-terminals are denoted with capital letters:

```
T →  (T N T)
     | ()
N →  number
```

Assuming that T is a binary search tree, define (`binary-tree-insert T n`) that takes T and an integer `n`, and inserts `n` into `T` while maintaining its binary search tree property. (Figure 1 illustrates the tree structures before and after insertion.)

```
(binary-tree-insert '((() 2 ()) 3 (() 4 (() 5 ()))) 6)
--> ((() 2 ()) 3 (() 4 (() 5 (() 6 ()))))
```

11. Create a functional abstraction of the following two Scheme functions and then redefine each function in terms of the abstraction.

```
(define (rember* a M)
    (cond
      [(null? M) '()]
      [(not (pair? (first M))) (if (eq? a (first M))
                                   (rember* a (rest M))
                                   (cons (first M) (rember* a (rest M))))]
      [else (cons (rember* a (first M))
                  (rember* a (rest M)))]))

(define (depth M)
    (cond
      [(null? M) 1]
      [(not (pair? (first M))) (depth (rest M))]
      [else (max (add1 (depth (first M)))
                 (depth (rest M)))]))
```

12. **For this problem you must use the "Lazy Racket" language in Dr. Racket.** Write an infinite list `f-list` that contains all solutions to the following recurrence relation:

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 2 \\ f(2) &= 3 \\ f(n) &= 3f(n-1) + 2f(n-3) \quad n \geq 3 \end{aligned}$$

What to turn in: For problems 1 through 11, put your code in a file named `program5.rkt`. For problem 12, put your solution in a file named `lazy.rkt`. Turn in both files via Canvas.