CS4121 Project 5: Scheme Programming

Due Date: Friday, April 18, 2025 at *11:59 pm*

In each of the following problems you can assume the following data definitions:

$M \rightarrow$ '() | $(A\ .\ M)$ | $(M\ .\ M)$
$L \rightarrow$ '() | $(A\ .\ L)$
$A \rightarrow$ atom

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. Some of you may find the built-in function `append` helpful for a problem or two. For example, `(append '(a b) '(c d))` results in `'(a b c d)`.

Give a Scheme solution to the following function specifications:

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

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

2. `(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
```

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

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

4. `(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))
```

5. `(last a L)` returns the one-based index of the last occurrence of the atom `a` in the flat list `L`, or 0 if there is no occurrence of `a` in `L`. (Hint: You may need to define and use a local function.)

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

6. `(select pred L)` returns a flat list of those elements in `L` that satisfy the predicate `pred`.

```
> (select number? '(a 2 #f b 7))
'(2 7)
> (select symbol? '(a 2 #f b 7))
'(a b)
```

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

   Your answer must define and use a local function `sumrow` which adds two flat lists of numbers. For example, `(sumrow '(1 2 3) '(4 5 6))` results in `'(5 7 9)`.

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)
```

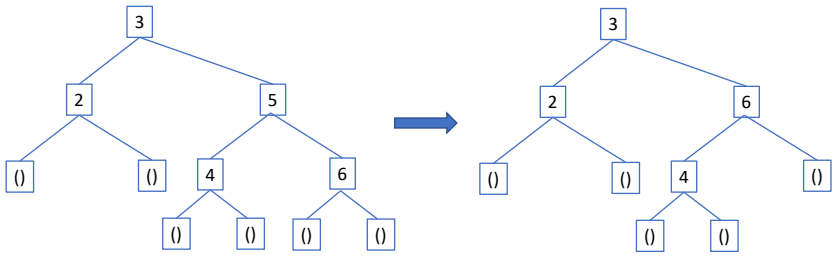

Figure 1: Binary Tree Deletion

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-delete T n)` that takes T and an integer `n`, and delete `n` from `T` while maintaining its binary search tree property. Figure 1 illustrates the tree structures before and after deleting 5. You can refer to the attached CS4321 lecture notes for the binary search tree algorithms. The deletion algorithm is in Slide 13.

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

(binary-tree-delete '((() 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 (count* a M)
    (cond
      [(null? M) 0]
      [(not (pair? (first M))) (if (eq? a (first M))
                                   (add1 (count* a (rest M)))
                                   (count* a (rest M)))]
      [else (+ (count* a (first M))
               (count* a (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:

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

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.