In this homework, you are asked to write Scheme functions. Please submit a single file named “homework6.ss” that contains definitions of all these functions. Do not include the #lang racket line. We have to be able to load your file into the racket command-line interpreter.

Problem 1

Write Scheme programs that generate the following lists as output using only cons as the list building operator:

1. ’(a (b c) d ((e f) (g)))
2. ’(* a 4) such that ((car ’(* a 4)) 5 3) evaluates to 15.

Problem 2

Write the following functions on lists in Scheme. The semantics of the functions is described through examples.

1. (define flatten
   (lambda (l
             ...
           ))
   ...
   (flatten ’(a ((b) (c d) (((e))))) → ’(a b c d e)

2. (define rev
    (lambda (l
              ...
            ))
    ...
    (rev ’(a((b)(c d)(((e)))))) → ’(((e)))(d c)(b))a

   Note: Do not use the Scheme build-in function ”reverse”.

3. (define double
    (lambda (l
              ...
            ))
...  
  (double (a((b)(c d)(((e)))))) -->(a a((b b)(c c d d)(((e e))))))

4.  (define delete  
    (lambda (atom 1)  
        ...))
...  
  (delete 'c (a((b)(c d)(((e)))))) -->(a(b)(d)(((e))))))
  (delete 'f (a((b)(c d)(((e)))))) -->(a((b)(c d)(((e))))))

Problem 3

Implement a symbol table data type that supports the following operations:

1. NewTable() : returns an empty table value;

2. InsertIntoTable((variable value), table) : inserts a variable/value pair into the table;

3. LookupTable(variable, table) : finds entry for variable and returns its value. If no variable is found, the empty list is returned. If more than one entry for a variable, the most recently entered value for that variable will be returned.

(define NewTable  
    (lambda () ... ))
(define InsertIntoTable  
    (lambda (entry table) // entry is a list of a variable and a value  
        ... ))
(define LookupTable  
    (lambda (variable table)  
        ... ))

(define table  
    (InsertIntoTable '(b (2 4 5)) (InsertIntoTable '(a 7) (NewTable))))

(LookupTable 'a table) -->(7
(LookupTable 'b table) -->(2 4 5)
(LookupTable 'c table) -->()
Problem 4

Use the map and reduce functions defined as

(define map
  (lambda (f l)
    (if (null? l)
      ()
      (cons (f (car l)) (map f (cdr l))))))

(define reduce
  (lambda (op l id)
    (if (null? l)
      id
      (op (car l) (reduce op (cdr l) id))))

to implement functions minSquareVal and maxSquareVal that determine the minimal square
value and maximal square value, respectively, of a list of integer numbers. Example

(define minSquareVal
  (lambda (l)
    ... ))

... (minSquareVal '(5 3 -7 10 -11 8 7)) -> 9

(define maxSquareVal
  (lambda (l)
    ... ))

... (maxSquareVal '(5 3 -7 10 -11 8 7)) -> 121