CS 565 Out: January 15, 1998
Jens Palsberg Due: January 29, 1998

Homework 1:
Scheme Programming

Define and test the procedures outlined below. In doing the assignment, you may assume that the inputs to your functions have the correct type. This means that you need not check the inputs for validity. However, this also means that your function must behave gracefully (that is, not bomb) on all such inputs.

Your solution should consist of the required function definitions, including comments and any necessary auxiliary functions, and a suite of tests adequate to convince the grader that your solutions work correctly on all valid inputs.

You should turn both your solution and a transcript of your tests. The command `(transcript-on` *filename*`)` will make a transcript of your session on the given file. `(transcript-off)` ends transcription and closes the transcript file.

Note: You can "warm up" to doing the exercises by doing some of the Exercises 2.2.7 and 2.2.8 in *EOPL* (pp. 50–54). You can find solutions to these exercises on the homepage for CS 565, Fall 1996.

1. Write a Scheme function `rotate-n` that takes as arguments a list of integers l and an integer n, and returns a list like the original, except that the elements have been rotated n times. A single rotation removes the first element of the list and puts it at the end of the list. For example:

```
(rotate-n '(4 6 3 -8 3 4) 0)  ==> (4 6 3 -8 3 4)
(rotate-n '(4 6 3 -8 3 4) 1)  ==> (6 3 -8 3 4 4)
(rotate-n '(5 6 7) 4)         ==> (6 7 5)
```

2. Write a Scheme procedure `split` that takes a list of integers l, and returns a list with three elements, all of which are lists. The first element-list contains all negative numbers in l, in the same order they appear in l. The second element-list contains all 0s in l. The third element-list contains all strictly positive numbers in l, in the same order they appear in l. For example:

```
(split '(5 3 -2 4 -6))      ==> ((-2 -6) () (5 3 4))
(split '(1 2 3 4))          ==> (() () (1 2 3 4))
(split '(1 -1 0 -2 0 3 -3)) ==> ((-1 -2 -3) (0 0) (1 3))
```

3. Write a Scheme procedure `sum` that takes as argument a list of integers l, and returns the sum of all elements that have an odd-numbered index. Recall that the elements of a list are numbered 0, 1, etc. Convention: the sum of *no* integers is 0. For example:

```
(sum '(5 3 -2 4 -6))    ==> 7
(sum '(1 2 3 4))        ==> 6
(sum '(1 -1 0 -2 3 -3)) ==> -6
(sum '())               ==> 0
```

4. Write a Scheme procedure `majority?` which takes as argument a list of integers l, and returns either `#t` or `#f` depending on the following condition. Suppose l contains n elements. If there is an element in l which occurs in l strictly more than $n/2$ times, then `(majority?` l`)` returns `#t`, otherwise `#f`. For example:

```
(majority? '())          ==> #f
(majority? '(2))         ==> #t
(majority? '(3 4))       ==> #f
(majority? '(2 3 2))     ==> #t
(majority? '(2 3 4))     ==> #f
(majority? '(2 3 2 3))   ==> #f
(majority? '(2 3 2 4 2)) ==> #t
```

5. Exercise 2.2.9.2 in *EOPL*, p.54. Write the program in Scheme.

6. In this problem we consider 2×2 matrices where the elements are integers. Let us represent a matrix

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

as the list $(a_{11}\ \ a_{12}\ \ a_{21}\ \ a_{22})$. Write a Scheme procedure `matrix-multiply` which takes as arguments such two matrix-representations, and returns a representation of the matrix-product of those two matrices.

For example:

```
(matrix-multiply '(1 2 0 1) '(1 0 1 1)) ==> (3 2 1 1)
(matrix-multiply '(1 0 0 1) '(2 3 4 5)) ==> (2 3 4 5)
```

7. Write a Scheme procedure `extensional-eq?` which takes as arguments two procedures and an integer n. Each procedure-argument takes one integer argument and it returns an integer. The call

`(extensional-eq?` $f\ \ g\ \ n$`)`

returns `#t` if f and g agree on the inputs $0, 1, \ldots, n$, and it returns `#f` otherwise.

For example:

```
(extensional-eq?
  (lambda (x) (+ x 1))
  (lambda (y) (- (+ y 2) 1)) 5)  ==> #t
(extensional-eq?
  (lambda (x) (* x x))
  (lambda (x) x) 1)              ==> #t
(extensional-eq?
  (lambda (x) (* x x))
  (lambda (x) x) 2)              ==> #f
```

8. Write a Scheme procedure `acc` which takes as argument a list of integers $l = (s_0 \ s_1 \ \ldots \ s_{n-1})$, and returns a list $m = (t_0 \ t_1 \ \ldots \ t_{n-1})$, where, for each $i \in \{0, 1, \ldots, n-1\}$,

$$t_i = s_0 + s_1 + \ldots + s_i \ .$$

For example,

```
(acc '(1 2 3)) ==> (1 3 6)
(acc '(2 0 3)) ==> (2 2 5)
(acc '())      ==> ()
```