CSci 4651 Fall 2003
Problem Set 2: Functional programming (Scheme)

Problem 1. Scheme allows a programmer to write very general functions which can be instantiated to perform a variety of tasks. Below is a function `traverse` that allows working with lists in a very general way. `traverse` returns a **function** that traverses a list and performs a specified task. The task depends on the parameters passed to traverse.

Given appropriate parameters, `traverse` can generate a mapping function (a function that modifies all elements of a list in a certain way), a filter (removing elements that don't satisfy a certain condition from a list), and functions for many other tasks on lists. Below is definition of `traverse`:

```
(define traverse (lambda (combine do seed)
  (lambda (x)
    (cond ((eq?  x '()) seed)
      (#t (combine (do (car x))
                   ((traverse combine do seed) (cdr x))))))))
```

The three parameters of `traverse` are as follows:

- `combine` is a function that combines the result for one element with the result for the rest of the list,
- `do` is a function that performs the specified action on the element, and
- `seed` is the result for an empty list.

Example: the function `mapsquare` below is defined via `traverse`. Given a list of integers, it creates a list of squares of these integers:

```
> (define mapsquare (traverse cons (lambda (x) (* x x)) '()))
> (mapsquare '(1 -2 3))
(1 4 9)
```

You should study this example in detail before attempting to solve the problems below.

Note: examples in this problem set use `cond` instead of `if` and `'()` instead of `nil` or `null` for an empty list. Please feel free to use other notations if they are more familiar to you. All the code given here has been tested under DrScheme.

Question 1. Using `traverse`, define and test the following functions:

1. `sumlist` to compute the sum of all the elements of an integer list.
2. `count` to count the number of elements in a list (make sure to test this function on a list of non-integers).
3. `remove5` to remove all 5s from a list of integers.
4. `reverse` to reverse a list.

5. `min` to find a minimum element in a list of integers (What would be the seed for this function? Make an assumption about the largest number that may appear on a list)

6. **Extra Credit.** `bettermin` – a function that finds a minimum element in a non-empty the list and returns $\#f$ for an empty list. Your function should work for arbitrary large numbers.

You may define other functions to solve the problem. Submit a printout of your interactions with scheme (File $->$ Print Interactions, then print after viewing the preview).

Question 2. Write a function `deeptraverse` which is analogous to `traverse`, but works on lists of lists (of arbitrary level of nesting). For instance, you should be able to use deeptraverse like this:

```
> (define deepmapsquare (deeptraverse cons (lambda (x) (* x x)) '()))
> (deepmapsquare '(1 () (3 (-2 5))))
(1 () (9 (4 25)))
```

The function `list?` which returns $\#t$ if the argument is a list and $\#f$ otherwise might be helpful for this task.

Test your solution carefully to make sure that it works for various kinds of nested lists.

Question 3. Using `deeptraverse` from Question 2, define the following functions:

1. `deepsumlist` to compute the sum of all the elements of a list of lists.

2. `deepreverse` to reverse every list in a list of lists. For example,

   ```
   > (deepreverse '(1 () (3 (-2 5))))
   (((5 -2) 3) () 1)
   ```

3. `flatten` to "flatten" a list of lists, i.e. to put all of its elements in a single list. For instance:

   ```
   > (flatten '(1 () (3 (-2 5))))
   (1 3 -2 5)
   ```

   Note: your function should preserve the order of the elements.

Problem 2. Exercise 3.1 p. 40, parts b,c.
Problem 3. Exercise 3.2 p. 40-41, parts a,b,c.
Problem 4. Exercise 3.6 p. 44-45.