CSci 4651 Fall 2008
Problem Set 2: Functional programming (Scheme). Due Wedn. Sept. 17 at 1pm (electronically)

Problem 1 (30 points). Scheme allows a programmer to write very general functions that 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 (creating a new list that contains only the elements of the given list that satisfy a certain condition), 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 an 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)
```

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.

You may define other functions to solve the problem. Submit your code, including all test data, electronically.

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: `flatten` must preserve the order of the elements.

Problem 2 (5 points). Exercise 3.1 p. 40.
Problem 3 (6 points). Exercise 3.2 p. 40-41, parts a,b.