CSCI 4150 Introduction to Artificial Intelligence, Fall 1999
Assignment 1: out Tuesday August 31, due Tuesday September 7

This assignment is to get you started programming in Scheme. You shouldn't have any trouble with the algorithmic content of these problems, although there will be some adjustment if you are not accustomed to functional programming.

The first thing you must do is to find a way to run MIT Scheme, either on your own computer or on RCS IBM or SGI (UNIX) workstations. See the course home page:

`http://www.cs.rpi.edu/courses/fall99/ai`

for more information on running Scheme.

Next, you should familiarize yourself with the following Scheme functions; we will cover most of these in class during the first week.

- variable/function definition: `define, let, let*`
- conditionals: `if, cond, case, and, or, not`
- list functions: `length, car, cdr, list-ref, list-tail, cons, list, append, member, reverse`
- mathematical functions: `+, -, *, /, quotient, max, min, truncate, round, floor, ceiling, sqrt, abs, remainder, modulo, gcd, lcm, expt, exp, log, sin, cos, tan, asin, acos, atan`
- mathematical predicates: `=, <, >, <=, >=, zero?, positive?, negative?, even?, odd?, number?, real?, integer?, complex?, rational?`
- non-mathematical predicates: `equal?, list?, null?, symbol?`

This assignment can be done only using the above functions, but you may also use MIT Scheme extensions in these categories. I encourage you to stick to standard Scheme as much as possible. You MAY NOT use "advanced" features of Scheme at this point, in particular: `map`, `apply`, `assoc`, `set!`, iterative functions, and any of their variants.

Note that several of the problems are optional; do them if you need the practice with Scheme. DO NOT TURN IN SOLUTIONS FOR THE OPTIONAL PROBLEMS. We will not grade the optional problems; you can check your answers against the solutions.

YOUR ASSIGNMENT IS TO BE TURNED IN ELECTRONICALLY. See the course home page for details.

1. (optional) Write functions to perform the following computations:
   - compute the hypotenuse of a right triangle given the two sides
   - compute the distance between two points in the plane
   - add 8% sales tax to a price given in dollars, rounding to the nearest cent
   - compute the function:
   $$y(x) = \begin{cases} x & -1 \leq x \leq 1 \\ x^3 & x < -1, x > 1 \end{cases}$$
2. (10 points) Under the Gregorian calendar, a year is a leap year if either:
   - the year is divisible by 4 but not by 100
   - the year is divisible by 400

   Write a predicate `(leap-year? year)` that determines whether `year` is a leap year.

3. (15 points) The Ackermann function is defined by:

$$\begin{array}{rcll} A(1,j) & = & 2^j, & j \geq 1 \\ A(i,1) & = & A(i-1,2), & i \geq 2 \\ A(i,j) & = & A(i-1, A(i,j-1)) & \end{array}$$

Write a function `(ackermann i j)` that computes values of the Ackermann function using recursion. Note that it would take a *very long* time to compute values for $i \geq 3$, so you should limit your testing to small values of i (and j).

4. (optional) What are the values of the following expressions:

```
(cadr '((a b) (c d) e f))
(caar '((a b) (c d) e f))
(cdadr '((a b) (c d) e f))
(caadr '((a b) (c d) e f))
(cdaadr '((g (h i) (j k)) (((l))) (m n)))
(caddar '((g (h i) (j k)) (((l))) (m n)))
(caaadr '((g (h i) (j k)) (((l))) (m n)))
```

5. (optional) Assume that the following definitions have been made:

```
(define numbers '(2 4 6))
(define letters '(q e d))
(define deep-list '(((13))))
```

Using only these three variables and the functions `cons`, `list`, and `append`, write expressions that will return the following lists:

```
(2 4 6 q e d ((13)))
((2 4 6) (q e d) ((13))
(2 4 6 (q e d) ((13)))
((2 4 6) (q e d) (((13))))
((2 4 6) q e d ((13)))
(((2 4 6) q e d) ((13)))
((2 4 6) ((q e d) ((13))))
```

6. (20 points) Write the following functions:

   (a) `(wacko x)` which performs the following:
   - if `x` is a list, returns the list reversed
   - if `x` is a symbol, returns the symbol
   - if `x` is an integer, returns 0 if it is even, 1 if it is odd
   - if `x` is a positive real number, returns the natural log of `x`
   - and otherwise returns the symbol `wacko`

   (b) `(days-in-month month is-leap-year)` which, given a month and whether or not this is a leap year (`#t` or `#f`) returns the number of days in that month. Assume `month` is of the symbols: `jan`, `feb`, `mar`, `apr`, `may`, `jun`, `jul`, `aug`, `sep`, `oct`, `nov`, `dec`.

7. (25 points) Write a function `(list-sums lst)` which takes a list `lst` and returns a list where the k^{th} element is the sum of the first k elements of `lst`. For example:

```
(list-sums '(1 4 8 2 5 3))    ==>    (1 5 13 15 20 23)
```