The purpose of this homework assignment is to review the Scheme language. You are encouraged to work together and ask lots of questions. The problems range from straightforward to medium difficult, but hopefully you find them interesting and learn how Scheme can be used to solve problems.

While there are no specific style guidelines for this course, good programming practices are expected including, but not limited to, meaningful names, comments, and proper indenting. Also, **avoid iterative constructs and assignment where possible**, and expect to use lots of recursion.

Your solutions should be saved in a file called `hwk1.scm`. In addition, you should create a file `hwk1sol.txt` of the interaction window demonstrating your solutions running on various test cases. You may edit out any errors due to typos, and you may add (Scheme-style) comments to this file, but you may not edit any of the results. (The instructor is going to run your solutions in any case. Edited output will result in a 0 for the assignment.)

The two files should be tar'd together: `tar -xvf NameHwk1.tar hwk1.scm hwk1sol.txt` and email the tarfile as an attachment to the instructor at `hwang@evansville.edu`.

1. List recursion. Write the following functions:

   a. `(last-element l)` returns the last element of list `l`. (Not using any built-in functions.)

   b. `(all-but-last l)` returns a list of all but the last element of list `l`

   c. `(middle l)` returns the middle element of list `l`, assuming an odd number of elements. Do not use any numeric value (such as a list's length) to implement `middle`.

2. Powerset. A standard mathematical set function is `powerset`, which computes the set of all subsets of a set. (Sets are represented as lists, so the powerset is a list of lists.) For example,

   ```scheme
   > (powerset '(a b))
   > (() (a) (b) (a b))
   ```
Implement `powerset` as a recursive function. It may require writing other subsidiary functions. The output may be in a different order than shown above. The key is to consider the relationship between the powerset of a set `s` and the powerset of `(cdr s)`.

3. Symbolic differentiation. One important AI application area is symbolic mathematics, particularly calculus. For this problem, construct a function `deriv` that differentiates a simple, single-variable mathematical expression. The function takes two arguments. The first is a mathematical expression in standard Lisp syntax, containing numbers, symbols (representing constants and variables) and the binary functions `+`, `-`, `*`, `/`, and `expt` (exponentiation). The second is a symbol that is the name of the variable with respect to which to differentiate. Other symbols in the expression are treated as constants. The rules of differentiation are as follows (where `u` and `v` are arbitrary expressions):

- `u = constant`, implies `du/dx = 0`
- `dx/dx = 1`
- `d(u+v)/dx = du/dx + dv/dx`
- `d(u-v)/dx = du/dx - dv/dx`
- `d(uv)/dx = udv/dx + vdu/dx`
- `d(u/v)/dx = (vdu/dx - udv/dx)/v^2`
- `v = constant`, implies `d(u^v)/dx = (du/dx)v^v-1` (`u^v` is written as `(expt u v)` in Scheme)

Basically, you want to move recursively through the mathematical expression constructing the derivative expression. Each case can be clause in a single cond statement.

Test your function on some interesting inputs. Add more rules (e.g., for trig functions or `e` ((exp x) in Scheme) if you wish. Note that the results do not need to be simplified. For example,

```
> (deriv '(* x x) 'x)    ;; u = x, v = x, du/dx = 1, dv/dx = 1
(+ (* x 1) (* x 1))
```

If you wish to explore, the use of predicates like `symbol?` and `number?` can be used along with `eval` and `quasiquote` (i.e. backquote) to actually compute the value of expressions with only constants or identity expressions like `(* x 1)` into just `x`.

4. Data types. Recall the `define-struct` macro is used to write structure type definitions. The example given in class was for 2-D points: `(define-struct point x y)`. that defined constructor `make-point`, predicate `point?`, accessors `point-x` and `point-y`, and mutators `set-point-x!` and `set-point-y!`.

01/16/07
a. Using the point definition above, write a function \((\text{distance } \textit{p1} \textit{ p2})\) that returns the distance between two points. The formula \(d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\) computes the distance between two points \((x_1, y_1)\) and \((x_2, y_2)\).

b. Write a \texttt{define-struct} definition for a line segment that consists of two (end)points.

c. Write a function \((\text{midpoint } \textit{ls})\) that returns the midpoint of a line segment.