Assignment 15, Math 240, Fall 2005

Due: 2:45pm, December 6. Value: 30 pts.

As with all assignments, you may submit your solution hand-written or typed. You must submit it on paper, though.

Scheme is installed on the eight Macintosh computers nearest the door in MCRey 314. To access it, open the Applications folder on the hard drive, go into the PLT Scheme folder, and double-click DrScheme. You can also install Scheme on your own computer (Linux, Mac, or Windows) via the drscheme.org *Web site; I recommend configuring it to use the intermediate language level (not beginner).*

Problem A. Write a Scheme function `list-sum` that takes an list as a parameter and counts the number of elements in it. Your function must recurse through the list itself; it must not use the built-in `sum` operation.

```
> (list-sum '(2 3 5 7 11))
28
```

The `first`, `rest`, and `+` operations are all useful in completing this.

Problem B. Write a Scheme function `tree-sum` that takes a list and counts the number of overall elements in it.

```
> (tree-sum '((2 4 6 8) (3 9) 5 7))
44
```

In addition to the operations from the previous problem, the `list?` operation is also useful.

Problem C. Write a Scheme function `my-eval` that accepts an arithmetic expression written in Scheme and evaluates it. (Your function must itself process the list; it must not use Scheme's built-in `eval` function.)

```
> (my-eval '(+ 2 (* 3 5)))
17
```

The only operations you need to support are `+` and `*`, and you can assume that these operations always have exactly two arguments.

In addition to the operations used in the previous problem, the `eq?`, `second`, and `third` operations are also useful.

Problem D. Write a Scheme function `my-eval-2` that accepts an arithmetic expression written in Scheme as well as a function for evaluating identifier values, and which returns the evaluation of the expression based on the identifier values given in the function. (Again, you only need to support `+` and `*`, and you can assume they always work on two arguments.)

For example, to create a function that specifies that `x` is 2, `y` is 3, and all other identifiers are 0, we could write the following.

```
(define (x2y3 ident)
  (if (eq? ident 'x) 2
      (if (eq? ident 'y) 3
          0)))
```

Then we could write the following to evaluate the expression $x^2 + 2y$ using your function.

```
> (my-eval-2 '(+ (* x x) (* 2 y)) x2y3)
10
```

In addition to the operations used in the previous problem, the `number?` operation is also useful.