SI 413 Fall 2012: Homework 3

Your name:

Due: Friday, 7 September, before class

Instructions: Review the course honor policy for written homeworks.

This cover sheet must be the front page of what you hand in. Fill out the left column in the table to the right after we go over each problem in class, according to the rubric below.

This rubric is also on the website, in more detail, under "Other Stuff"→"Grading Rubrics".

Make sure all problems are submitted IN ORDER.

- **5**: Solution is completely correct, concisely presented, and neatly written.
- **4**: The solution is mostly correct, but one or two minor details were missed, or the presentation could be more concise.
- **3**: The main idea is correct, but there are some significant mistakes. The presentation is somewhat sloppy or confused.
- **2**: A complete effort was made, but the result is mostly incorrect. There may be some basic misunderstandings of the topic or the problem.
- **1**: The beginning of an attempt was made, but the work is clearly incomplete.
- **0**: Not submitted.

Problem	Self-assessment	Final assessment
1		
2		
3		
4		
5		

Comments or suggestions about this homework:

Comments or suggestions about the course so far:

Citations (other students, websites, . . .):

Use a separate sheet of paper for your answers! Many of these exercises are programming exercises, but you do not need to submit them electronically. Everything should be turned in in one packet, all printed out for me to see.

1 Nested Lets

Write a Scheme expression that is equivalent to the following Java code, by using a series of 3 nested **let** expressions.

```
1 int x = 1;
2 x += 3;
3 x *= 12;
4 return x;
```

2 Homoiconicity

The Wikipedia page on homoiconicity claims that raw machine code can be considered homoiconic, just like Scheme.

Explain what this means in a few sentences of your own.

Then tell me what properties of most homoiconic languages (like Scheme) does machine code definitely *not* have.

3 Adder

Scheme has a built-in function `add1` which adds 1 to its argument.

But what if we wanted to create a similar function like `add2` or `add20`? Write a **function that produces a function** `make-adder` that takes an argument `n` and produces a lambda expression (function) that takes one argument and adds `n` to it.

So for example `((make-adder 5) 10)` produces `15`.

4 Expression Transformer

Write a Scheme function `letter` that takes a **quoted let** expression and turns it into an equivalent quoted **lambda** expression.

For example, calling

```
1 (letter '(let ((a 7)
2                (b 10))
3            (* a b)))
```

should produce the expression `((lambda (a b) (* a b)) 7 10)`. (And if you call **eval** on that expression from the interactions window in DrScheme, you should get the result, 70.)

5 The Doubler

a) Write a function `(double f)` which takes a 1-argument function `f` and produces a 1-argument function (as a lambda expression) that takes an argument `x` and applies `f` to it twice, like `(f (f x))`.

   So for example calling `((double sqrt) 16)` is the same as calling `(sqrt (sqrt 16))`, which is `2`.

b) Can we do `(double double)`? What does it do?