CS 61A Summer 2010 Week 7A Lab
Monday 8/2 Afternoon

This lab will introduce you to the idea of continuations. Feel free to work on the project after understanding them.

Suppose we have the following definition:

```
(define (square x cont)
  (cont (* x x)))
```

Here `x` is the number we want to square, and `cont` is the procedure to which we want to pass the result. Now try these experiments:

```
> (square 5 (lambda (x) x))

> (square 5 (lambda (x) (+ x 2)))

> (square 5 (lambda (x) (square x (lambda (x) x))))

> (square 5 display)

> (define foo 3)
> (square 5 (lambda (x) (set! foo x)))
> foo
```

Don't just type them in – make sure you understand why they work! The nondeterministic evaluator works by evalutating every expression with *two* continuations, one used if the computation succeeds, and one used if it fails.

```
(define (reciprocal x yes no)
  (if (= x 0)
      (no x)
      (yes (/ 1 x))))

> (reciprocal 3 (lambda (x) x) (lambda (x) (se x '(cannot reciprocate))))

> (reciprocal 0 (lambda (x) x) (lambda (x) (se x '(cannot reciprocate))))
```

1. Copy the file ∼`cs61a/lib/ambeval-step2.scm` to your directory. This file contains the incomplete Non-deterministic evaluator where `if` and `choose` has been implemented.

Implement the special form `or` into our current non-deterministic evaluator. Note that we do not yet support procedure call, so you have to test your code with atoms and `choose` statements inside the `or`.

2. Once you feel comfortable with the ∼`cs61a/lib/ambeval-step2.scm` code. Take a look at ∼`cs61a/lib/ambeval-final.scm`, ignore the special form handling for now and try to understand the code for handling procedure calls. The procedure `eval-args` looks quite tricky but isn't that hard to understand. Just keep in mind that `mc-eval` (and hence `eval-args`) needs to pass the return value to the success continuation rather than returning it directly.

3. Now, consider the `set!` code. The only tricky part here is that we need to save the old value of the variable so we can set it back when we backtrack. Do 4.51 from Abelson and Sussman and you can think about how we can use this feature to not repeatly return the same value when we `choose-another`.