Implementign Call-By-Name

There are three commonly used mechanisms for linking the parameters of a procedure to arguments:

- call-by-value. The arguments are evaluated in the caller's environment and their values are bound to the parameters of the function.
- call-by-reference. Here the arguments must be variables. Their addresses are bound to the parameters of the procedure.
- call-by-name.   Here the arguments are not evaluated in the caller's environment. The text of the arguments is passed to the procedure and replaces the parameters in the procedure body.

Here is an example that shows the difference between call-by-value and call-by-name.

```
(let ([x 0])
     (let ([f (lambda (y)
               (begin
                    (set! x (+ x 1))
                    y))])
          (f (+ x 5))))
```

With call-by-value f is called with argument 5; it sets x to 1 but then returns its argument 5.

With call-by-name the text of the function body becomes

```
(begin  (set! x (+ x 1))
        (+ x 5))
```

and this evaluates to 6.

To change MiniScheme to using call-by-name, we don't evaluate the arguments at all. In (apply-proc p args) we rebuild the tree for the body of p by substituting each argument for the corresponding parameter, then we evaluate this tree in the procedure's environment.

This takes us through a series of steps.

a) On an app-exp eval-exp calls apply-proc as usual, but with *unevaluated* arguments:

```
        (apply-proc (eval-exp op env) args env))
```

b) apply-proc calls a function substitute to replace parameters in the closure-body with the argument expressions:

```
        (eval-exp (substitute args params bod) c-env))]
```

c) substitute does one substitution and recurses on the rest:

```
(define substitute  (lambda (args params exp)
      (cond
            [(null? args) exp]
            [else (substitute (cdr args) (cdr params)
                              (substitute-one (car args) (car params) exp))])))
```

d) Finally, substitute-one is where the work get done.

```
(define substitute-one
       (lambda (arg param exp)
                      (cond
                              [(lit-exp? exp) exp]
                               [(varref-exp? exp)
                                        (let ([variable (varref-sym exp)])
                                                (if (eq? variable param) arg exp))]
                              [(if-exp? exp) (let ([test (if-test-exp exp)]
                                                                 [then-part (if-then-exp exp)]
                                                                  [else-part (if-else-exp exp)])
                                     (new-if-exp (substitute-one arg param test)
                                                            (substitute-one arg param then-part)
                                                            (substitute-one arg param else-part))))]
```

And so forth. substitute-one recurses into every subexpression of every MiniScheme expression, replacing param with arg.