Implementing Call-By-Reference

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-reference.

```
(let ([x 0])
     (let ([f (lambda (y) (set! y 34))])
          (begin
               (f x)
               x)))
```

With call-by-value f is called with argument 0; f changes the value of its parameter y from 0 to 34, but this has nothing to do with variable x. The overall expression returns 0.

With call-by-reference f is called with the address of variable x, so the set! actually changes x. This expression returns 34.

Your MiniScheme interpreter implements call-by-value.

To change it to call-by-reference we do 3 things.

a) First, we go back to the original version of extended-env:

```
(define extended-env (lambda (syms vals old-env)
        (list 'extended-env syms vals old-env)))
```

This means that you have to map box around any values you actually put into an extended environment

b) Next, when evaluating the arguments in the caller's environment, we get the boxes the arguments are bound to (remember, the arguments must be variables). We don't unbox them.

This means that instead of calling apply-proc with the evaluate args:

```
    ... (apply-proc p (map (lambda (t) (eval-exp t env)) args))
```

We call it with the boxes:

```
.....  (apply-proc p (map (lambda (t) (lookup env (varref-sym t))) args))
```

Here varref-sym is my name for the getter of a var-ref (which is what the arguments must be)

b) Second, when extending the function's environment in order to evaluate its body, we don't box the argument values; we just bind the argument boxes to the parameters

c) Finally, when extending the function's environment in apply-proc in order to evaluate its body, we don't box the argument values; we just bind the argument boxes to the parameters:

Instead of
(eval-exp body (extended-env params (map box args) cenv))

We do
(eval-exp body (extended-env params args cenv))

Here params body and cenv are variables holding the closure parameters, body, and environment.

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.