Parameter-Passing Mechanisms

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.

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.

Each of these mechanisms has its advantages. call-by-value is the easiest to understand, and is the most commonly used mechanism. It is used by Scheme, Java, C, and Python.

Call-by-reference allows a procedure to change its argument, and this can be a useful thing.

Call-by-name has the advantage of delaying the evaluation of an argument until it is used. Sometimes a function uses one of its arguments very rarely, but the evaluation of the argument is expensive in terms of time or memory. Call-by-name is useful in such a situation.

Implementing each of these mechanisms is easy.

Your MiniScheme interpreter implements call-by-value.

To change it to call-by-reference we do 2 things. First, 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. 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.

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.