# METACIRCULAR EVALUATOR 11

COMPUTER SCIENCE 61AS

## Metacircular Evaluator

1. How are expressions treated as in Metacircular Evaluator?

2. There are many types of procedures, such as special forms. How does MCE figure out the type of the expression?

3. How are primitive procedures represented? What about compound procedures?

## Operations on Environment

1. MCE is an extended version of Scheme-1. What is the most significant improvement from Scheme-1?

2. What is a frame? How is it represented?

3. How does MCE look up a variable?

## Practice with MCE

1. Recall that mceval.scm tests true or false using the true? and false? procedure:

```
(define (true? x) (not (eq? x false)))
(define (false? x) (eq? x false))
```

Suppose we type the following definition into MCE:

```
MCE> (define true false)
```

What would be returned by the following expression:

```
MCE> (if (= 2 2) 3 4)
```

_____

2. Suppose we type the following into mc-eval:

```
MCE> (define 'x (* x x))
```

This expression evaluates without error. What would be returned by the following expressions?

```
MCE> quote
```

_____

```
MCE> (quote 10)
```

_____

3. Suppose we just loaded mceval.scm into STk, and erroneously started MCE using (driver-loop) instead of (mce). What is the return value of the following sequence of expressions typed into MCE? If there will be an error, just write ERROR:

```
MCE> 2
```

_____

```
MCE> (+ 2 3)
```

_____

```
MCE> (define x 10)
```

_____

4. Rewrite one procedure in the metacircular evaluator so that it will understand infix arithmetic operators. That is, if a compound expression has three subexpressions, of which the second is a procedure but the first isn't, then the procedure should be called with the first and third subexpressions as arguments:

```
> (2 + 3)
5
> (+ 2 3)
5
```

5. Write the new special from `text-multiple`. It takes in as arguments a predicate of one argument and an arbitrary number of arguments. It tests the predicate on each argument in turn. As soon as one of them returns `#t`, it outputs true. If it has tested all of the arguments and none are true, it outputs false.

```
;;; M-Eval input:
(test-multiple (lambda (x) (equal? 'b x)) 'a 'e 'i 'o 'u)

;;; M-Eval output:
#f

;;; M-Eval input:
(test-multiple (lambda (x) (= x 0)) 3 (/ 1 0) 0)

;;; M-Eval output:
Error

;;; M-Eval input:
(test-multiple (lambda (x) (= x 0)) 0 1 (/ 1 0) 2)


;;; M-Eval output:
#t

;;; M-Eval input
(test-multiple (lambda (x) (= x 0)) 2 3 4)

;;; M-Eval output:
#f
```