CS61A – Lab Assignment 6.1 **Kurt Meinz**
University of California, Berkeley Summer 2005

1. List all the procedures in the metacircular evaluator that call `mc-eval`.

2. List all the procedures in the metacircular evaluator that call `mc-apply`.

3. Abelson and Sussman, exercise 4.1

4. In this exercise, we will begin to write a postfix version of the metacircular evaluator. For now, do not worry about special forms; we will only provide postfix support for procedure calls, self-evaluating expressions, etc.

a. Make a new copy of `mceval.scm` and call it `postfix-mceval.scm` (we don't want these changes to conflict with future exercises). Make all changes for this exercise in `postfix-mceval.scm`.

b. Our postfix expressions will look the same as our prefix expressions, except that the operator will go on the right. Examples:

```
(2 3 +)
(((3 1 +) 8 *) 9 -)
('(one no trump) car)
```

To make this somewhat efficient (by only looking at each subexpression once), we will maintain a `stack` in `mc-eval`. The purpose of this stack is to keep track of evaluated subexpressions until we get to the operator.

Example: We want to evaluate `(2 (3 4 +) +)`

```
-> mc-eval called with exp = (2 (3 4 +) +)  stack = ()  ;; not at operator yet
   -> mc-eval called with exp = 2  stack = ()           ;; .. so evaluate first arg
-> mc-eval called with exp = ((3 4 +) +)  stack = (2)   ;; not at operator yet -> evaluate 2nd arg
   -> mc-eval called with exp = (3 4 +)  stack = ()     ;; .. second arg is a nested exp
      -> mc-eval called with exp = 3 stack = ()         ;; .... evaluate 1st arg of nested exp
   -> mc-eval called with exp = (4 +)  stack = (3)      ;; .. not at operator yet
      -> mc-eval called with exp = 4 stack = ()         ;; .... evaluate 2nd arg of nested exp
   -> mc-eval called with exp = (+)  stack = (3 4)      ;; .. found the operator
      -> mc-eval called with exp = + stack = ()         ;; .... so evaluate op and perform operation
-> mc-eval called with exp = (+)  stack = (2 7)         ;; found the operator
                                                       ;; .. so evaluate op and perform operation
```

As you can see, everytime we start evaluating a new expression, we start with an empty stack. At the top of `mceval.scm`, give a definition of `the-empty-stack`, which will be used to represent such an empty stack.

c. Modify the code for `mc-eval` so that it now takes in an additional argument: the `stack`

d. Modify the `application?` clause of mc-eval so that it deals with postfix expressions and exhibits the behavior in the example in step b. Do not worry about data abstraction.

e. Modify the code so that all calls to mc-eval have an appropriate stack.

f. Try it out!

g. Given the current modifications, what types of expressions will not work in postfix notation. What additional modifications are necessary?