Problem I.
(i) Implement, document (i.e. provide specifications), and test the following functions in Scheme (Racket is also allowed). An association list is a list of bindings of names to values: ((name1 val1) ... (name t value t)). This data structure can be used to implement a symbol table. An environment can be represented by a list of association lists (i.e. a list of symbol tables), where the first element in the list is nearest scope, the second the next surrounding scope, and the last the outermost scope.

All programming targets were achieved to the 3rd degree. Below is proof of program correctness.

(a) Write recursive scheme function (lookup name assoc_list) that returns the binding (pair) whose name equals the given name. If no such binding is found return the null list.
Below is a screenshot that shows our implementation of the (lookup) function.

```
;1 i A
;assoc_list is a list of bindings.
;Goal: find binding (pair) whose name equals the given name
(define (lookup name assoc_list)
  (cond
    ;if assoc_list is null return a null list
    ((null? assoc_list) '())
    ;if the first element of the first binding is equal to name, return the binding
    ((eq? name (car (car assoc_list))) (car assoc_list))
    ;otherwise recursion, call the function again with the rest of the list of bindings
    (else (lookup name (cdr assoc_list)))))

(define assoc_list1 '( (dennis "george") (stephen "hansen") (shivanshi "nagar")))
(lookup 'dennis assoc_list1)
(lookup 'ben assoc_list1)
```

Running the above code warrants the following output:

```
tux3: problem1 > racket problem1i.rkt
'(dennis "george")
'()
```

This proves that the function is working as intended.

(b) Write a recursive function (lookup-env name environment), which returns the binding with the specified name in an environment (i.e. list of association lists) and null if no such binding is found.

Hint: Review the data structures of the implementation of the Metacircular Evaluator, presented in Week 6 Part 1 file.
Below is a screenshot that shows our implementation of the (lookup-env) function.

```
;Goal: find binding with the specified name in an environment
(define (lookup-env name environment)
  ;if environment is null
  (if (null? environment) '()
      ;look for each name using function lookup in each association list
      (let ((each (lookup name (car environment))))
        ;if found return the name
        (if (not (null? each)) each
            ;if not call the function with the rest of the environment
            (lookup-env name (cdr environment))))))

(define env1 '( (sixers "philly") (lakers "la") (blazers "portland")))
(define env2 '( (chicken "chick fil a") (mint "chocolate") (subway "sandwiches") ))
(define env (list env1 env2) )
(lookup-env 'sixers env)
(lookup-env 'mint env)
(lookup-env 'hoagie env)
```

Running the above code warrants the following output:

```
'(sixers "philly")
'(mint "chocolate")
'()
```

This proves that the function is working as intended.

(ii) Work out solutions in Scheme (Racket is also allowed) of the following SICP programming problems: 4.4, 4.9 of section 4.1.2 and 4.11 of section 4.1.3. Submit your solutions together with the data demonstrating that your code works properly. You may directly adapt the code available at http://community.schemewiki.org/.

4.4 Install and and or as new special forms for the evaluator by defining appropriate syntax procedures and evaluation procedures eval-and and eval-or. Alternatively, show how to implement and and or as derived expressions.

The appropriate special forms were added to the “ch4-mceval.scm” file. This screenshot below shows that they were successfully implemented.

```
tux1: problem1 > mit-scheme
MIT/GNU Scheme running under GNU/Linux
Type `^C' (control-C) followed by `H' to obtain information

Copyright (C) 2019 Massachusetts Institute of Technology
This is free software; see the source for copying conditions

Image saved on Thursday September 5, 2019 at 11:51:46 AM
  Release 10.1.10 || Microcode 15.3 || Runtime 15.7 || SF 4.

1 ]=> (load "ch4-mceval.scm")

;Loading "ch4-mceval.scm"... done
;Value: metacircular-evaluator-loaded

1 ]=> (define the-global-environment (setup-environment))

;Value: the-global-environment

1 ]=> (driver-loop)


;;; M-Eval input:
(and true false)

;;; M-Eval value:
#f

;;; M-Eval input:
(or true false)

;;; M-Eval value:
#t

;;; M-Eval input:
```

4.9 Many languages support a variety of iteration constructs, such as do, for, while, and until. In Scheme, iterative processes can be expressed in terms of ordinary procedure calls, so special iteration constructs provide no essential gain in computational power. On the other hand, such constructs are often convenient. Design some iteration constructs, give examples of their use, and show how to implement them as derived expressions.

```
1 ]=> (while->combination '(while (< i 100) (display i) (set! i (+ i 1))))

;Value: ((lambda () (define (while-procedure) (if (< i 100) (begin (display i) (set! i (+ i 1)) (while-procedure)) 0)) (while-procedure)))
```

4.11 Instead of representing a frame as a pair of lists, we can represent a frame as a list of bindings, where each binding is a name-value pair. Rewrite the environment operations to use this alternative representation.

```
1 ]=> (load "ch4-mceval.scm")

;Loading "ch4-mceval.scm"... done
;Value: metacircular-evaluator-loaded

1 ]=> (make-frame '(a b c) '(1 2 3))

;Value: (table (a . 1) (b . 2) (c . 3))

1 ]=> (define test-frame (make-frame '(a b c) '(1 2 3)))

;Value: test-frame

1 ]=> (frame-pairs test-frame)

;Value: ((a . 1) (b . 2) (c . 3))

1 ]=> (add-binding-to-frame! 'd 4 test-frame)

;Unspecified return value

1 ]=> (frame-pairs test-frame)

;Value: ((d . 4) (a . 1) (b . 2) (c . 3))

1 ]=> (define the-global-environment (setup-environment))

;Value: the-global-environment
```

```
1 ]=> (define-variable! 'foo 'bar the-global-environment)

;Unspecified return value

1 ]=> (lookup-variable-value 'foo the-global-environment)

;Value: bar

1 ]=> (set-variable-value! 'foo 'baz the-global-environment)

;Unspecified return value

1 ]=> (lookup-variable-value 'foo the-global-environment)

;Value: baz
```