**CS 330 – Programming Languages**

**HW5Scheme 1**

Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Key\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Grade \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part I. Warming up** - tell the result from evaluating the following scheme expressions. You could use DrRacket (Lang = Pretty Big or similar*)* to verify your answers for this part. (2x20=40 points)

(a) (cdr (car '((a b) d (c d)))) 🡪 \_\_\_\_a\_\_\_\_\_\_\_

(b) (car (car (cdr '(cdr ((a b) (c d) e f))))) 🡪 \_\_\_\_(a b)\_\_\_\_\_

(c) (cadadr ' ((a b) (c d) (e f))) 🡪 \_\_\_\_\_d\_\_\_\_\_\_

(d) (cdr '(car (cdr (cdr ((a b) (c d) e f))))) 🡪 \_\_\_\_((cdr (cdr ((a b) (c d) e f))))\_\_\_\_\_\_\_\_

(e) '(car '(car (cdr (cdr ((a b) (c d) e f))))) 🡪 \_(car '(car (cdr (cdr ((a b) (c d) e f))))) \_\_\_\_\_\_\_\_

(f) (cons ‘a ' (b c d)) 🡪 \_\_\_\_(a b c d)\_\_\_\_\_

(g) (append '(a b) ‘(c d)) 🡪 \_\_\_\_(a b c d)\_\_\_\_\_\_\_\_

(h) (list '(a b) '(c d)) 🡪 \_\_\_\_((a b) (c d)\_\_\_\_\_\_\_\_

(i) (member 'a '(ba b a c)) 🡪 \_\_\_\_(a c)\_\_\_\_\_\_\_\_

(j) (list '(b c d) (list 'a)) 🡪 \_\_\_\_((b c d) (a))\_\_\_\_\_\_\_\_

(k) ((lambda x x) 1 2 3 ) 🡪 \_\_\_\_(1 2 3)\_\_\_\_\_\_\_\_

(l) (symbol? 'a) 🡪 \_\_\_\_\_#t\_\_\_\_\_\_\_

(m) (null? ' ()) 🡪 \_\_\_\_\_#t\_\_\_\_\_\_\_

(n) (reverse ' (a (b c) d)) 🡪 \_\_\_\_\_(d (b c) a)\_\_\_\_\_\_\_

(o) (length ' (a (b c) (d) e)) 🡪 \_\_\_\_\_\_4\_\_\_\_\_\_

(p) (display “Hello World!”) 🡪 \_\_\_\_Hello World!\_\_\_\_\_\_\_\_

(q) (write “Hello World!”) 🡪 \_\_\_\_”Hello World!”\_\_\_\_\_\_\_\_

(r) (let ((a 2)) (set! a (read)) a) 🡪 \_\_\_\_\_Hello\_\_\_\_\_\_\_

(input is *Hello World!)*

(s) (append ' (b c d) (list 'a)) 🡪 \_\_\_\_(b c d a)\_\_\_\_\_\_\_\_

(t) (reverse (cdr (reverse '(x y z)))) 🡪 \_\_\_\_(x y)\_\_\_\_\_\_\_\_

**Part II. Working with numbers on Leetcode –** define racket scheme functions for the following leetcode problems. Note that leetcode heading include a contract def using -> which should not affect your solution. The easiest way to implement your scheme solutions in racket and then simply initiate a call from the leetcode interface function to your scheme function. Submit screenshot of your source code and leetcode acceptance for this part. (10X3=30 points)

1. Problem 136 single number

Hint:

a. call your function (ex., sn) with a sorted list (asc),

b. check to see if car is same as cadr, if yes, recursive call on cdr, if no, return car

What is the base case here?

;; Modified interface – same function name and argument, removed contract notation

(define single-number

(lambda (L)

(sn (sort L <))))

(define sn

(lambda (L)

(cond ((null? (cdr L)) (car L))

((eq? (car L) (cadr L)) (sn (cddr L)))

(else (car L))

))

)

1. Problem 137 single number 2

Hint: extending from 136 appears to be cake

;; this is original Leetcode interface

(define/contract (single-number nums)

  (-> (listof exact-integer?) exact-integer?)

  (sn2 (sort nums <))

  )

(lambda (L)

(cond ((null? (cdr L)) (car L))

((eq? (car L) (cadr L)) (sn2 (cdddr L)))

(else (car L))

))

)

1. Problem 268 missing number

Hint:

1. call your main function with the list sorted in DSC order.
2. At reverse order, car should equal list length
3. How do we know the list is missing a 0?

(define/contract (missing-number nums)

(-> (listof exact-integer?) exact-integer?)

(mn (sort nums >))

)

(define mn

(lambda (nums)

(cond ((null? nums) 0)

((eq? (car nums) (length nums))

(mn (cdr nums)))

(else (length nums))

)

)

)