Quiz 9b Rubric

1. (3 points) Say what Scheme prints, and draw a box-and-pointer diagram. In the case of an error, say “error” and don’t draw the box-and-pointer diagram. If there is a cycle, which would make it print infinitely, write “cycle” (and still draw the box-and-pointer diagram).

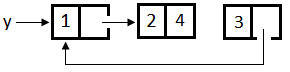
(define x (cons (list 1) 2))

(set-cdr! (car x) (cons 3 (cadar x)))

(set-cdr! x 4)

x

Error (0.5 points, all or nothing. Reason for error: (cadar x) doesn’t exist)



(define y (list 1 2 3))

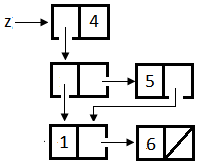
(set-cdr! (cddr y) y)

(set-cdr! (cdr y) 4)

y

(1 2 . 4) (0.5 points for printing, 0.5 points for box-and-pointer)

The last box doesn’t need to be shown (since it may be garbage collected). However, if they do draw it, and it is wrong, don’t give them the 0.5 points for the box-and-pointer.



(define z (cons (cons (cons 1 2) 3) 4))

(set-cdr! (car z) (cons 5 (caar z)))

(set-cdr! (caar z) (cons 6 ‘()))

z

(((1 6) 5 1 6) . 4)

0.5 points for printing, 1 point for box-and-pointer diagram. Partial credit of 0.5 points for box-and-pointer if it has exactly 5 boxes.

1. (4 points) Somehow, all of your data structures have gotten corrupted! All the pairs have been swapped – what should have been in the car is in the cdr instead, and vice versa. Write a procedure fix-data! that takes in some arbitrary structure created by cons, and swaps the car and the cdr for all of the pairs in that structure, **without** creating any new pairs. It must only modify existing pairs. (Hint: Since we are treating the car and cdr similarly here, it is convenient to think of the structure as a tree, so that you make two recursive calls, as in lesson 5.) Space is provided on the back.

> (define x (cons (cons (cons ‘() 5) 4) (cons ‘() 3)))

> (fix-data! x)

> x

((3) 4 5)

(define (fix-data! struct)

(if (not (pair? struct))

struct

(let ((orig-car (car struct)))

(set-car! struct (fix-data! (cdr struct)))

(set-cdr! struct (fix-data! orig-car))

struct)))

–4 points for no mutation

–3 points for no recursion

–3 points for creating new pairs, but still using mutation in some way

–1 point for not having enough recursion (eg. recursing only on the cdr)

–1 point for not saving the car, or something similar

–1 point for forgetting to return the new structure, **if that is necessary for the recursion**

–1 point for not switching the car and cdr

–1 point for missing/inadequate base case (for example, if base case is (null? struct) )

If they get less than a point using that rubric, then use this one instead:

Score of 1 if they have some mutation that clearly attempts to switch the car with the cdr, **or** vice versa, **and** they have at least one recursive call

Score of 0.5 if they have any call to set-car! or set-cdr!

1. (3 points) Write the procedure split-vector, which takes as input a vector vec and an index i. It returns two new vectors – one with the elements from index 0 to i-1, and one with the elements from index i+1 onwards. (Notice that the element at index i does not appear in the result.) At this point you should be saying “I can’t return more than one thing!” You’re absolutely right. So, you should instead return a pair of vectors, i.e. something like (cons left-vec right-vec). Helper procedures are okay.

Warning: This question is not going to be nearly as elegant as other procedures you have written. Expect to write more code than usual.

> (define pair (split-vector (vector ‘h ‘e ‘l ‘l ‘o ‘- ‘w ‘o ‘r ‘l ‘d) 5))

> pair

(#(h e l l o) . #(w o r l d))

(define (split-vector vec i)

(let\* ((left-vec (make-vector i))

(right-length (- (- (vector-length vec) i) 1))

(right-vec (make-vector right-length)))

(define (fill-left pos)

(if (< pos i)

(begin (vector-set! left-vec pos (vector-ref vec pos))

(fill-left (+ pos 1)))))

(define (fill-right pos)

(if (< pos right-length)

(begin (vector-set! right-vec pos

(vector-ref vec (+ 1 (+ i pos))))

(fill-right (+ pos 1)))))

(fill-left 0)

(fill-right 0)

(cons left-vec right-vec)))

–0.5 points for each distinct off-by-one error, up to –1

–1 point for any other indexing error

–0.5 points for missing begin, unless it’s not necessary (eg. If they used a cond clause)

–0.5 points if the forget to return the vectors, or return value is not a pair of vectors

–1 point for only filling one of the two vectors

If they get less than a point using that rubric, use this one instead:

Score of 1 if they try to create two vectors, **and** try to modify at least one using recursion

Score of 0.5 if they create two vectors **or** try to modify one using recursion