# Midterm exam 1 (sample #2)

CS61a (sp11)

### Problem 1 (box and pointer)

(let ((x (list 1 2 3)))

(set-cdr! (car x) 4)

x)

=> Error

The expression (set-cdr! (car x) 4) will become (set-cdr! 1 4) and will throw an error cause 1 is not a pair, thus it’s a bad argument to set-cdr!.

(let ((x (list 1 2 3)))

(set-cdr! x 4)

x)

=> (1 . 4)



(let ((x (list 1 2 3)))

(set-car! (cdr x) x)

x)

=> #0=(1 #0# 3)



(define a ((lambda (z) (cons z z)) (list 'a)))

(set-cdr! (car a) '(b))

a

=> ((a b) a b)



### Problem 2 (Assignment, State, and Environments)

(define (make-player room-number)

(lambda (direction)

(set! room-number (next-room room-number direction))

room-number))

### Problem 3 (Drawing environment diagrams).

> (define foo

(lambda (x f)

(if f

(f 7)

(foo 5 (lambda (y) (+ x y))))))

> (foo 3 #f)

=> 10

E0 - is the environment created when we invoked (foo 3 #f)

E1 - is the environment created when we invoked (foo 5 (lambda (y) (+ x y))) inside the outer foo call

E2 - is the environment created when we invoked (f 7) inside inner foo call

The result is **10**



### Problem 4 (List mutation)

(define (merge! l1 l2)

(cond

((null? l1) l2)

((null? l2) l1)

((<= (car l1) (car l2))

(set-cdr! l1 (merge! (cdr l1) l2))

l1)

(else

(set-cdr! l2 (merge! l1 (cdr l2)))

l2)))

### Problem 5 (Vectors)

(define (rotate! v)

(define (loop i)

(if (= i 0)

v

(let ((current (vector-ref v i))

(prev (vector-ref v (- i 1))))

(vector-set! v i prev)

(vector-set! v (- i 1) current)

(loop (- i 1)))))

(loop (- (vector-length v) 1)))

### Problem 6 (Concurrency)

(a) Since both critical sections are serialized then we will have only correct answers (the same answers if we execute those assignments consecutively):

1. 10 - divide first (10/2=5) then addition (5+5=10)
2. 10 - addition first (10+10=20) then divide (20/2=10)

So, the value of baz always will be - 10, no matter in which order those operations will be executed.

(b) Now suppose that we change the example to leave out the serializer, as follows:

> (define baz 10)

> (parallel-execute (lambda () (set! baz (/ baz 2)))

(lambda () (set! baz (+ baz baz))))

What are all of the possible values of baz this time?

We still can have correct values:

1. 10 - divide first (10/2=5) then addition (5+5=10)
2. 10 - addition first (10+10=20) then divide (20/2=10)

But, in addition, now we also can have race conditions, thus incorrect values:

1. 5 - both read value 10 of baz, the division will assign its result AFTER the addition finishes
2. 20 - both read value 10 of baz, the addition will assign its result AFTER the division finishes
3. 15 - the addition reads value 10 for the first baz, then division finishes, then addition reads value 5 for the second baz

### Problem 7 (Streams)

(define stream-car car)

(define origin-stream-cdr stream-cdr)

(define (stream-cdr s)

(if (stream? s)

(origin-stream-cdr s)

(cdr s)))