# Midterm exam 1 (sample #2)

CS61a (sp11)

### Problem 1 (Higher order procedures)

In the twenty-one project, which of the following are higher-order procedures? **Check all**

**correct answers.**

\_\_\_ (best-value hand)

\_\_\_ (stop-at-17 my-hand dealer-card)

\_X\_ (play-n strategy n)

\_X\_ (stop-at n)

\_X\_ (majority strat1 strat2 strat3)

### Problem 2 (Scheme syntax).

(define (foo x)

(if x (foo #f) 5))

(define (baz x)

(and x (baz #f) 5))

What is the value of (foo 3)? \_\_\_\_\_\_\_\_5\_\_\_\_\_\_

What is the value of (baz 3)? \_\_\_\_\_\_\_#f\_\_\_\_\_\_

### Problem 3 (Recursive and iterative processes).

In question 2 above, one of the procedures foo and baz generates a recursive process; the

other generates an iterative process. Which is which, and in one English sentence,

explain why

ANSWER: foo procedure generates an iterative process cause it uses a tail recursion method - a recursive call would be (if the predicate is truthy) the last thing returned from the procedure. In a baz procedure and will wait until the recursive call finishes to “determine” what to do next.

### Problem 4 (Mutation).

Here is a transcript of a Scheme session. Fill in the blanks. (It will help if you draw a box

and pointer diagram first.)

> a

(1 2 (3 4 5) 6)

> b

(1 2 3 4 5)

> c

(1 2 (3 4 5) 6)

> (eq? (cddr b) (caddr a))

#T

> (eq? (caddr c) (caddr a))

#F

> (eq? (cdaddr c) (cdddr b))

#T

> (set-car! (caddr a) 7)

okay

> (set-car! (cdaddr a) 8)

okay

> b

1 2 7 8 5

> c

1 2 (3 8 5) 6

### Problem 5 (Object oriented programming).

Here is a class definition in OOP language:

(define-class (echo saved)

(instance-vars (count 0))

(default-method

(set! count (+ count 1))

(let ((result saved))

(set! saved message)

result)))

Write an equivalent program in ordinary Scheme. Don’t forget to include methods for the

messages saved and count! Here’s an example of how your program will be used:

> (define my-echo (make-echo ’hello))

MY-ECHO

> (my-echo ’foo)

HELLO

> (my-echo ’baz)

FOO

> (my-echo ’saved)

BAZ

> (my-echo ’garply)

BAZ

> (my-echo ’count)

3

We’ve given you the first line of the program; continue from there:

(define (make-echo saved)

(let ((count 0))

(lambda (msg)

(cond

((eq? msg 'count) count)

((eq? msg 'saved) saved)

(else

(set! count (+ count 1))

(let ((result saved))

(set! saved msg)

result))))))

### Problem 6 (Streams).

What are the first 20 elements of the stream mystery defined as follows:

(define mystery (cons-stream 1 (interleave integers mystery)))

Assume that integers is the stream of integers starting with 1

1 1 1 2 1 3 1 4 2 5 1 6 3 7 1 8 4 9 2 10

### Problem 7 (Metacircular evaluator).

Rewrite *one procedure* in the metacircular evaluator so that it will understand infix arithmetic operators. That is, if a compound expression has three subexpressions, of which the

second is a procedure but the first isn’t, then the procedure should be called with the first

and third subexpressions as arguments:

> (2 + 3)

5

> (+ 2 3)

5

You may write new helper procedures if needed

Changes in cond clause in eval procedure:

((application? exp)

(let ((left (eval (operator exp) env)))

(if (or (compound-procedure? left)

(primitive-procedure? left))

(apply left (list-of-values (operands exp) env))

(if (= (length exp) 3)

(apply (eval (cadr exp) env)

(list left (eval (caddr exp) env)))

(error "Wrong expression")))))

### Problem 8 (Logic programming).

Last year’s final asked students to invent a logic program that would multiply two nonnegative integers, with integers represented as lists of the appropriate length, so (a a a)

represents 3. We’re going to continue inventing arithmetic operations

**Don’t use lisp-value in your solutions.**

(a) Write a rule or rules to determine if one integer is less than another. For example, the

query

(less ?x (a a a))

should give the results:

(less () (a a a))

(less (a) (a a a))

(less (a a) (a a a))

N/A

(b) Suppose you are given logic rules for plus and times, so the query

(times (a a) ?what (a a a a a a))

gives the result

(times (a a) (a a a) (a a a a a a))

Your job is to write a divide logic rule or rules with places for the dividend, the divisor,

the quotient, and the remainder:

(divide (a a a a a a a) (a a a) ?quo ?rem)

should give the result

(divide (a a a a a a a) (a a a) (a a) (a))

indicating that 7 divided by 3 gives a quotient of 2 with remainder 1.

Note: Don’t write rules for plus or times; assume you are given those!

Hint: Part (a) will be useful

(rule (divide dividend? divisor? quotient? remainder?)

(and (times divisor? quotient? ?x)

(plus ?x remainder? dividend?)))

### Problem 9 (Environment diagrams).

(a) Draw the environment diagram that will result from the following sequence of Scheme

expressions:

(define x 3)

(define y 4)

(define foo ((lambda (x) (lambda (y) (+ x y))) (+ x y)))

(foo 10)



(b) What is the value of the expression (foo 10) above?

The value is 17

### Problem 10 (Deep lists).

Write a function named locate that takes two arguments: a value and a list structure

containing that value. It should find the position of the value in the structure (e.g., the

car of the cdr of the cdr) and should return a selector function to extract that position

from any similarly-shaped structure. For example:

> (define baz (locate 5 ’(1 2 (3 4 5) 6 7)))

BAZ

> (baz ’(a b (c d e) f g))

E

If the value is not found in the structure, locate should return #F. You may assume that

the value will not be found more than once in the structure.

(define (locate val lst)

(define (helper val lst)

(cond

((null? lst) #f)

((equal? val (car lst)) (list car))

((pair? (car lst))

(let ((result (helper val (car lst))))

(if result

(cons car result)

(helper val (cdr lst)))))

(else

(let ((result (helper val (cdr lst))))

(if result

(cons cdr result)

#f)))))

(define path-list (helper val lst))

(define (extract struct path-list)

(if (null? path-list)

struct

(extract ((car path-list) struct) (cdr path-list))))

(lambda (struct) (extract struct path-list)))