CS 61A Final - December 16, 1998

Your name

login cs61a-

This exam is worth 30 points, or about 21% of your total course grade. It contains ten questions.

This booklet contains eight numbered pages including the cover page. Put all answers on these pages, please; don't hand in stray pieces of paper. This is an open book exam.

When writing procedures, don't put in error checks. Assume that you will be given arguments of the correct type.

Question 1 (2 points):

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) ___ (play-n strategy n) ___ (stop-at n) ___ (majority strat1 strat2 strat3)

Question 2 (2 points):

(define (foo x) (if x (foo #f) 5))

(define (baz x) (and x (baz #f) 5))

What is the value of `(foo 3)`?

What is the value of `(baz 3)`?

Question 3 (2 points):

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.

Your name login cs61a-

Question 4 (2 points):

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

> c

Question 5 (3 points):

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)

Your name login cs61a-

Question 6 (3 points):

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.

Question 7 (4 points):

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.

Question 8 (4 points):

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))

(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.

Your name login cs61a-

Question 9 (4 points):

(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?

Question 10 (4 points):

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.