CS 61A Midterm #3 — April 11, 1994

Your name _____

login cs61a–_____

Discussion section number _____

TA's name _____

This exam is worth 20 points, or 11.5% of your total course grade. The exam contains four substantive questions, plus the following:

Question 0 (1 point): Fill out this front page correctly and put your name and login correctly at the top of each of the following pages.

This booklet contains six 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.

Our expectation is that many of you will not complete one or two of these questions. If you find one question especially difficult, leave it for later; start with the ones you find easier.

0	/1
1	/4
2	/5
3	/5
4	/5
total	/20

Question 1 (4 points):

What will the Scheme interpreter print in response to each of the following expressions? Also, draw a "box and pointer" diagram for the result of each expression. Hint: It'll be a lot easier if you draw the box and pointer diagram *first*!

```
(let ((x (list 1 2 3 4)))
  (set-cdr! (cdr x) (cdddr x))
  x)
```

```
(let ((x (list 1 2 3 4)))
  (set-car! (cdr x) (cadr x))
  x)
```

```
(let ((x (list 1 2 3 4)))
  (set-cdr! (cdr x) (car x))
  x)
```

```
(let ((x (list 1 (2 3) 4)))
  (set-car! x (cadr x))
  x)
```

Your name _____ login cs61a–_____

Question 2 (5 points):

Write `deep-subst!`, a procedure that takes three arguments, two of which are words and the third of which is any list structure (anything made of pairs). It should mutate the list structure so that any occurrence of the first argument, however deep in sublists, is replaced by the second argument. Examples:

```
> (deep-subst! 'foo 'baz (list (cons 'hello 'goodbye) (cons 'moby 'foo)))
((hello . goodbye) (moby . baz))

> (deep-subst! 'a 'x (list (list 'a 'b 'c) (list 'b 'a 'd)
                           (list 'f 'a 'b)))
((x b c) (b x d) (f x b))
```

Do not allocate any new pairs in your solution!

Question 3 (5 points):

We are simulating a read-only memory (ROM) in the OOP system:

```
(define-class (rom values)
  (default-method
    (if (and (number? message) (< message (length values)))
        (nth message values)
        (error "Bad ROM address"))))

> (define sample-rom (instantiate rom '(4 5 9 23))
> (ask sample-rom 2)
9
> (ask sample-rom 7)
ERROR -- Bad ROM address
```

Now we want to make a programmable ROM (PROM). Unlike a standard ROM, a PROM starts with nothing stored in it, although it does have a fixed size that is set when the PROM is built. You can put a value into each address, but only once—you can't change the value later. We want to implement the prom class so that it takes the size as its instantiation variable, but inherits from the rom class:

```
(define (prom size)
  (parent (rom ((repeated (lambda (vals) (cons '() vals)) size)
                '())))
  ...)
```

It should accept a SET message that takes an address (a number) and a value (anything) as arguments. If the prom is big enough to have such an address, but the value in that address is the empty list, then the new value should be put in that address.

```
> (define this-prom (instantiate prom 6))
> (ask this-prom 3)
()
> (ask this-prom 'set 3 'foo)
> (ask this-prom 3)
FOO
> (ask this-prom 'set 9 'baz)
ERROR -- Bad ROM address
> (ask this-prom 'set 3 'garply)
ERROR -- PROM address already has value
```

Complete the definition of the prom class. If possible, do it without modifying the definition of the rom class. If you must modify the rom definition, explain why.

Write your answer on the next page!

Your name _____ login cs61a–_____

Answer question 3 here:

Question 4 (5 points):

Quite a while ago you saw this procedure to get from one row of Pascal's triangle (represented as a sentence of numbers) to the next:

```
(define (next-row row)
  (define (iter old)
    (if (empty? (bf old))
        '(1)
        (se (+ (first old) (first (bf old)))
            (iter (bf old)))))
  (se 1 (iter row)))

> (next-row '(1 3 3 1))
(1 4 6 4 1)
```

(A) Rewrite `next-row` so that its argument and return value are (finite) streams instead of sentences.

(B) Using your stream version of `next-row`, create the (infinite) stream of all the rows of Pascal's triangle. (The first row of the triangle just contains a single number 1.)