CS 60A Midterm #3 — April 20, 1992

Your name _____

login c60a–_____

Discussion section number _____

TA's name _____

This exam is worth 20 points, or 12.5% 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)))
  (set-car! x (cdr x))
  x)
```

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

Your name _____ login c60a–_____

Question 2 (5 points):

We want to generate the stream of all *composite* numbers — that is, all positive integers that are *not* prime. The book gives us definitions for the streams `integers` and `primes`. We want to be able to say

```
(define composites (set-difference integers primes))
```

Write the function `set-difference`. It should take two (possibly infinite) ordered streams as arguments. (In other words, the elements of each stream are numbers in increasing order.) It should return the stream of all the numbers that are elements of the first argument but not elements of the second argument.

Question 3 (5 points):

We are going to add magic wands to the adventure game. A magic wand is a kind of `thing` that you can possess. When you wave the wand, it transports you instantly to some particular place. For example:

```
> (define wand-1 (instantiate wand 'silver shin-shin))
> (ask evans 'appear wand-1)
> (ask brian 'go 'down)
BRIAN MOVED FROM CSUA-OFFICE TO EVANS
> (ask brian 'take wand-1)
BRIAN TOOK SILVER-WAND
TAKEN
> (ask brian 'wave wand-1)
BRIAN MOVED FROM EVANS TO SHIN-SHIN
```

We need to invent the `wand` class and to add a `wave` method to the `person` class. Here is the `wave` method:

```
(define-class (person name place)
  ...
  (method (wave wand)
    (if (member wand possessions)
        (ask wand 'magic)
        (error "You don't own that wand!") ))
  ...)
```

Wands must understand a `magic` message; when a wand gets this message, it asks its possessor to `go-directly-to` its predetermined place.

Write the `wand` class definition.

Your name _____ login c60a–_____

Question 4 (5 points):

One of the problems in the object system is that an object can't make direct use of its parent's instance variables. For example:

```
(define-class (counter)
  (instance-vars (count 0))
  (method (next)
    (set! count (1+ count))
    count) )

(define-class (incrementer)
  (parent (counter))
  (method (next amount)
    (set! count (+ count amount))       ; This won't work!
    count) )
```

On the next page is an environment diagram showing the (simplified) result of defining these classes, and creating an instance

```
(define upper (instantiate incrementer))
```

(a) Extend the diagram on the next page to show the result of

```
((upper 'next) 3)                       ; Note two invocations.
```

[This expression is equivalent to the OOP `(ask upper 'next 3)`.]

(b) Briefly explain, with reference to the diagram, why the `set!` in the `incrementer` class doesn't do what you want.