CS 61A Midterm #3 — November 12, 2008

Your name ____

login: cs61a–_______

Discussion section number _______

TA's name ____

This exam is worth 40 points, or about 13% of your total course grade. It includes two parts: The individual exam (this part) is worth 36 points, and the group exam (the other part you probably just finished) is worth 4 points. The individual part contains six 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 seven 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, open notes 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.

If you want to use procedures defined in the book or reader as part of your solution to a programming problem, you must cite the page number on which it is defined so we know what you think it does.

READ AND SIGN THIS:

I certify that my answers to this exam are all my own work, and that I have not discussed the exam questions or answers with anyone prior to taking this exam.

If I am taking this exam early, I certify that I shall not discuss the exam questions or answers with anyone until after the scheduled exam time.

____

0	/1
1	/4
2	/6
3	/6
4	/8
5–6	/11
total	/36

Question 1 (4 points):

What will the Scheme interpreter print in response to **the last expression** in each of the following sequences of expressions? Also, **draw a "box and pointer" diagram for the final result of each sequence of expressions.** If any expression results in an error, **circle the expression that gives the error message** and just write "ERROR"; you don't have to give the precise message. Hint: It'll be a lot easier if you draw the box and pointer diagram *first*!

```
(let ((x (list 'a 'b 'c)))
  (set! (car x) (caddr x))
  (set-car! x (caddr x))
  x)
```

```
(define a (list (list 3) 5))
(define b (append a a))
(set-car! (cdr b) (caddr b))
(set-car! a (cons 3 4))
b
```

Question 2 (6 points):

Fill the blank with a mutation instruction so that the final result is as shown. (Hint: Draw a box and pointer diagram.)

Use mutation only. Do not allocate any new pairs!

```
> (define foo (list 1 2 (list 3 4)))
> (define bar (list (list ’a ’b) ’c ’d))

> ______________________________________________________________
> bar
((a b (3 4)) c d)
```

```
> (define foo (list 1 2 (list 3 4)))
> (define bar (list (list ’a ’b) ’c ’d))

> ______________________________________________________________
> bar
(((2 (3 4)) b) c d)
```

Question 3 (6 points):

Here is an OOP class definition:

```
(define-class (person name)
  (class-vars (people '()))
  (instance-vars (size 3))
  (initialize (set! people (cons self people)))
  (method (eat)
    (set! size (+ size 1)))
  (method (greet other)
    (print (se "Hi," (ask other 'name) "My name is" name))))
```

and here is a partially-written non-OOP-language equivalent:

```
(define make-person
  ; THIS IS POINT A.
  (lambda (name)
    ; THIS IS POINT B.
    (let ((size 3))
      (define (dispatch msg)
        ; THIS IS POINT C.
        (cond ((eq? msg 'people) (lambda () people))
              ((eq? msg 'size) (lambda () size))
              ((eq? msg 'name) (lambda () name))
              ((eq? msg 'eat)
               (lambda () (set! size (+ size 1))))
              ((eq? msg 'greet)
               (lambda (other)
                 (print (se "Hi,"
                            ; THIS IS POINT D.
                            "My name is" name))))
              (else (error "Bad message" msg))))
      ; THIS IS POINT E.
      dispatch)))
```

This question continues on the next page.

Your name ____ login cs61a–_______

Question 3 continued:

(a) To implement the class variable, we're going to put the line

```
(let ((people '()))
```

at which point in the program?

____Point A ____Point B ____Point C

(b) To implement the initialization, we're going to put the line

```
(set! people (cons ______________ people))
```

somewhere. What goes in the blank?

____`self` ____`make-person` ____`dispatch`

(c) Where does that initialization line go?

____Point B ____Point C ____Point E

(d) What goes in point D?

____`other`

____`(ask other 'name)`

____`(other 'name)`

____`((other 'name))`

Question 4 (8 points):

Do not use `vector->list` or `list->vector` in this problem. Use no data aggregates other than vectors.

Sort a vector in place, **allocating no new storage**, using the following (insertion sort) algorithm. Suppose you are asked to sort the vector

```
#(7 2 14 5)
```

We are going to insert elements starting at the right end of the vector. First we are going to insert `14` into the subvector containing just `5`. The result will be that the rightmost two elements are in order:

`#(7 2` **`5 14`**`)`

(Doing this involved moving the `5` leftward one slot, and inserting the `14` into the rightmost slot.)

Now insert the `2` into the subvector of `5` and `14`. Since 2 is less than 5, this doesn't require you to move anything:

`#(7` **`2 5 14`**`)`

Finally, insert the `7`, which involves moving the `2` and the `5`:

`#(`**`2 5 7 14`**`)`

This question continues on the next page.

Question 4 continued:

(a) Write a helper procedure `insert!` that takes four arguments: a vector, a number to be inserted, and the *vector element position numbers (indices)* of the first and last elements of the already-sorted subvector. It should insert the new number where it belongs in the subvector; the resulting larger sorted subvector starts one element to the left of the given starting element number. So, the last call to `insert!` for the example above will be equivalent to

```
(insert! '#(7 2 5 14) 7 1 3)
```

although, of course, your program won't have these particular values in it! Assume that the arguments are valid: the starting index is at least 1, the ending index is less than the length of the vector, and the elements between the two index values are already in order.

(b) Now write `sort!`, which takes a vector as its only argument, and sorts the vector in place using the algorithm described earlier. You may assume that the vector has at least two elements, to avoid special cases.

Question 5 (5 points): Write the first 5 elements of the following stream:

```
(define mystery
  (cons-stream 1
               (stream-map + ones (stream-map + ints mystery))))
```

________ ________ ________ ________ ________

Question 6 (6 points):

(a) What are the possible values of `x` after the following call to `parallel-execute`? (Note: Read the code carefully! We didn't make a mistake; this is what we intended to write.)

```
(define x 16)
(parallel-execute (lambda () (if (even? x) (+ x 5) (- x 5)))
                  (lambda () (set! x (* x 2)))
                  (lambda () (set! x (+ x 1))))
```

(b) Which of those possible values are correct?