CS 61A Midterm #3 — April 16, 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	/6
2	/5
3	/5
4	/7
5	/5
6	/7
total	/36

Question 1 (6 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 1 2 3)))
  (set-cdr! (cdr x) (caddr x))
  x)
```

```
(define a (list 2 3 5))
(define b (cons 1 a))
(set-car! (cdr a) (cadr b))
b
```

```
(let ((foo (list 0 3))
      (bar (list 7 10)))
  (set-car! (car foo) (car bar))
  foo)
```

Your name _____ login cs61a–_____

Question 2 (5 points):

Translate the following into OOP language using `define-class` (we've left out some details; the point is for you to identify the instance variables, class variables, instantiation variables, and methods and/or initializations):

```
(define make-horse
  (let ((population 0))
    (lambda (color)
      (let ((hunger 0))
        (define (dispatch message)
          (cond ((eq? message 'eat)
                 (lambda (grass)
                   (if (> hunger 0)
                       (set! hunger (- hunger grass))
                       (se color '(horse not hungry)))))
                ((eq? message 'color) (lambda () color))
                ((eq? message 'hunger) (lambda () hunger))
                ((eq? message 'population) (lambda () population)) ))
        (set! population (+ population 1))
        dispatch))))
```

Question 3 (5 points): Write the first 10 elements of the following stream:

```
(define mystery
  (cons-stream 1
    (cons-stream 2
      (interleave (stream-map (lambda (x) (* x 2)) mystery)
                  (stream-map (lambda (x) (+ x 1)) mystery)))))
```

_____ _____ _____ _____ _____

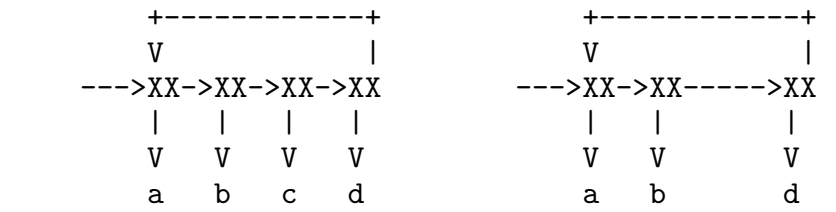
_____ _____ _____ _____ _____

Question 4 (7 points):

Write a procedure `josephus` that takes two arguments: a circular list of people's names, and a positive integer `k`, and repeats the following process until there is only one pair left:

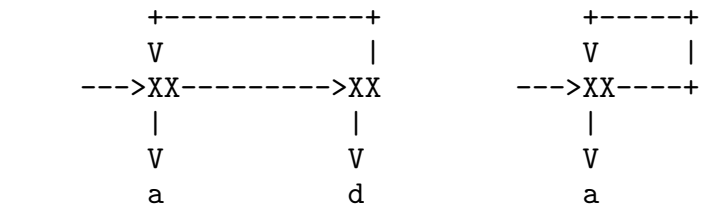
Starting with the first person, count up. The k-th pair is eliminated, and the process repeats, starting from what was originally the $k+1$st pair.

For example, suppose we have the list shown in the first diagram:



$k = 3$. We count 3 elements (`a`, `b`, `c`) and eliminate the `c` pair, giving the second diagram.

Then, we start at the `d` pair and repeat the process. Counting 3 elements forward (`d`, `a`, `b`), we remove the `b` pair.



Now, starting at the `d` pair, we count 3 elements forward (`d`, `a`, `d`) and remove the `d` pair. So, we finish with only the `a` pair left, and return that pair.

Do not allocate any new pairs!

Write your solution on the next page.

Your name _____ login cs61a–_____

Question 4 continued:

Question 5 (5 points):

The five TAs decide to go to lunch and are seated at a table. The food has just been served. Unfortunately, at the table there aren't enough utensils. There is one fork, one spoon, and one knife.

Evan needs a fork to eat some pasta.
Ramesh needs a fork and a spoon to eat curry.
Jerry needs a fork and a knife to eat steak.
Albert needs a spoon for some tasty soup.
Yaping needs a knife for bread and butter.

The TAs are all very hungry. Assume that none of the TAs are willing to let go of a utensil that they have until after finishing the meal. Also, assume that once a TA has the necessary utensils, he or she starts eating (i.e., nobody will pocket a utensil and laugh at the other TAs).

(a) Suppose that each TA finishes his or her food in five minutes. What is the shortest amount of time needed for everyone to finish? Give a possible ordering of people, showing clearly who eats in parallel, to finish in this amount of time.

(b) Can this situation lead to deadlock? If so, describe the sequence of events leading to deadlock. If not, change any one person's requirements so that deadlock is possible.

Your name _____ login cs61a–_____

Question 6 (7 points):

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

Write `diffs`, which takes a vector of numbers and returns a new vector with differences between consecutive elements. Assume the input vector has at least 2 elements.

Here are some examples:

```
> (diffs #(1 2 3 4))
#(1 1 1)
> (diffs #(1 8 2 2))
#(7 -6 0)
> (diffs #(1 2))
#(1)
```