CS 60A Midterm #2 — March 9, 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 Scheme print in response to the following expressions? Also, draw a "box and pointer" diagram for the result of each expression:

```
(cons (cdr '(x)) 3)
```

```
(list 2 '(3 4))
```

```
((lambda (s) (cons (car s) (cddr s))) '(w x y z))
```

```
(cons (list 2 3) '(a))
```

Your name _____ login c60a–_____

Question 2 (5 points):

Write a function `uniq` (named after a Unix utility program that does more or less the same thing) that takes a list as its argument. It should return a list that has the same elements, except that if an element is repeated several times in a row, it only appears once in the result list:

```
> (uniq '(a x b b b c c c c x d))
(a x b c x d)

> (uniq '(3 3 3 3 3 3 3 4 4 4 7 hello hello hello))
(3 4 7 hello)
```

Question 3 (5 points):

Write a function `add-numbers` that takes as its argument an arbitrary hierarchical list (that is, a list of lists) and returns the sum of all the numbers anywhere in the structure:

```
> (add-numbers '((4 calling birds) (5 gold rings) 10 (20 (30) 40)))
109
```

Your name _____ login c60a–_____

Question 4 (5 points):

This two-part question is about data-directed programming. We are going to use a list to represent a *finite state machine* (FSM) like this:

The numbered circles represent the *states* of the machine. At any moment the machine is in a particular state. The machine reads words, one letter at a time. If the machine is in some state, and sees a certain letter, it follows the arrow from its state with that letter as its label, and ends up in a new state. For example, if the machine above is in state 1 and it sees the letter B, it goes into state 3.

We'll represent a FSM by a list of all its transition arrows. The machine above has six such arrows:

```
((1 A 2) (1 B 3) (1 C 4) (2 A 1) (3 B 1) (4 C 1))
```

If the machine reads a letter for which there is no applicable arrow, it should enter state number 0. In effect, there are "invisible" arrows like `(2 C 0)` that needn't be represented explicitly in the list.

This question continues on the next page.

Question 4 continued.

(a) Write a function `transition` that takes three arguments: a FSM list, a current state number, and a letter. The function should return the new state. For example, if `fsm` represents the list above, we should be able to do this:

```
> (transition fsm 1 'C)
4
> (transition fsm 2 'C)
0
```

(b) We want an FSM to process words, not just single letters. The machine "reads" the word one letter at a time. Our sample FSM, starting in state 1, should process the word AACCAAB by going through states 2, 1, 4, 1, 2, 1, and 3. The final state, 3, is the result of processing the word.

Write a function `process` that takes as its arguments a FSM, a starting state number, and a word. It should return the ending state:

```
> (process fsm 1 'AACCAAB)
3
> (process fsm 1 'AAAC)
0
```