CS 3 (Clancy) Exam 1
March 1, 1993

THIS COPY HAS ANSWERS DISPLAYED.
Read and fill in this page now.
Do NOT turn the page until you are told to do so.

Your name:
(please print last name, first name)
Your lab section day and time:
Your lab t.a.:
Name of the person sitting to your left:
Name of the person sitting to your right:

Problem 0 Total: /30
Problem 1
Problem 2
Problem 3

This is an open-book test. You have approximately fifty minutes to complete it. You may consult any books, notes, or other inanimate objects available to you. To avoid confusion, read the problems carefully. If you find it hard to understand a problem, ask us to explain it. If you have a question during the test, please come to the front or the side of the room to ask it. This exam comprises 15% of the points on which your final grade will be based. Partial credit will be given for wrong answers.

You are not to use setf within functions. Any other Lisp construct described in Touretzky chapters 1-6, in part I of the *Difference Between Dates* case study, or in *Integer Division and its Uses* is ok, though. You need not rewrite a function that appears in Touretzky, in any of the case studies, or in any of the CS 3 handouts; merely cite the page in Touretzky, the case study and page number or appendix, or the handout in which the function appears.

Your exam should contain 4 problems (numbered 0 through 3) on 7 pages. Please write your answers in the spaces provided in the test; in particular, we will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there. Relax this exam is not worth having heart failure about.

Problem 0 (2 points; 1 minute)
Put your name on each page. Also make sure you have provided the information requested on the first page.

Problem 1 (10 points; 15 minutes)
For the first two parts of this problem, you are to write a function called two-added-at-end that takes three inputs, the first of which is a list, and returns the result of adding the second and third inputs as the next-to-last and last elements of the list. For example, if given the list (A B) and the two atoms C and D as inputs, two-added-at-end should return the list (A B C D).

```
Part a
Write a version of two-added-at-end that uses append.

(defun two-added-at-end (L a b)
        (append L (list a b)) )

2 points: -1 each error
```

Part b
Write a version of two-added-at-end that does not use append.

```
(defun two-added-at-end (L a b)
        (reverse (cons b (cons a (reverse L)))) )
```

3 points: -1 each error (e.g. each occurrence of forgetting
a reverse, using the wrong function, misparenthesizing)

Part c
Provide a call to two-added-at-end that returns the list
(4 5 (A B) (1 2)).

```
(two-added-at-end '(4 5) '(A B) '(1 2))
```

2 points: -1 each error.

Part d

Provide an expression that contains a pair of calls to two-added-at- end, one nested within the other, so that the expression s value is the list (3 4 5 (A B) (1 2)).

```
(two-added-at-end
        (two-added-at-end '(3) 4 5)
        '(A B) '(1 2))
```

3 points: -1 each error

Problem 2 (10 points, 20 minutes)
Background
A certain commuter airline flies in a loop from Sacramento to Fresno to San Jose to San Francisco back to Sacramento again, as shown in the diagram on the right. The arrows indicate the direction of flight; the numbers indicate the distance flown between one city and the next.

The distances from one city to another, flying along the airline's route, are as shown in the table below.

finish start	Sacramento	Fresno	San Jose	San Francisco
Sacramento	0	170	320	370
Fresno	290	0	150	200
San Jose	140	310	0	50
San Francisco	90	260	410	0

Part a
Supply the body of the function f in the code below, so that the distance-flown function returns the result specified in its comments. A solution that earns full credit will involve fewer than four function calls. You

may supply a setf for a table if necessary; place it outside your definition of f.

```
(setf city-table '(sacramento fresno san-jose san-francisco))
(defun precedes? (city1 city2)
        (member city2 (member city1 city-table)) )
; Start-city and finish-city are each either sacramento,
; fresno, san-jose, or san-francisco. Return the distance
; that the commuter airline would fly to go along its route
; starting at start-city and ending at finish-city.
(defun distance-flown (start-city finish-city)
        (if (precedes? start-city finish-city)
                (- (f finish-city) (f start-city))
                (+ 460 (- (f finish-city) (f start-city))) ) )
(defun f (city)

(second (assoc city dist-from-sacto-table))
)
(setf dist-from-sacto-table
        '((sacramento 0) (fresno 170)
          (san-jose 320) (san-francisco 370) ) )

5 points total: 3 for the table, 2 for accessing it
correctly.

-1 for a four-way cond, each condition of which involves an
assoc.

-1 for the first quoting error; no deduction for quoting
errors after the first.

-1 for the first parenthesis error; no deduction for
parenthesis errors after the first.
```

Part b
Provide a name for f that sufficiently describes the result it returns.

distance-from-sacramento

2 points: 1 for mentioning distance, the other for mentioning the base point from which the distance is measured.

Part c
Write a function next-stop that, given the name of a city as input, returns the name of the city that is next in the loop. For full credit, use no more than two function calls. You may also supply a setf for a list or table if necessary; place it outside your definition of next-stop.

```
(defun next-stop (city)
        (second (member city '(sacramento fresno
                san-jose san-francisco sacramento) )) )
```

```
(defun next-stop (city)
        (if (equal city 'san-francisco) 'sacramento)
                (second (member city city-table)) ) )
```

They can code a version with assoc also.

3 points for the first solution, 2 points for the second or for a large cond.

-1 for the first quoting error; no deduction for quoting errors after the first.

-1 for the first parenthesis error; no deduction for parenthesis errors after the first.

Problem 3 (8 points, 14 minutes)
In ancient Rome, the 13th of each month was called the ides.* Complete, in the space below, the definition of a function called ides? that returns a true value (i.e. a value that's not nil) if its input represents the ides of some month, and nil otherwise. Examples of what ides? should return appear in the table below.

input to ides?	value to return
clancy	nil
(clancy)	nil
(march)	nil
(march 3)	nil
(march 13 x)	nil
(march 13)	some true value

```
(setf all-month-names '(january february march april may june
        july august september october november december)
(defun ides? (date)

        (and
                (listp date)
                (= (length date) 2)
                (member (first date) all-month-names)
                (equal (second date) 13) ) )
```

2 pts for checking for listhood, 2 pts for checking for the correct length, 2 points for checking for a correct month, and 2 points for checking for the correct date in the month.

-1 for a parenthesis error, -2 for multiple parenthesis errors.

* The definition has been simplified for the purposes of this problem.