CS 61A, Fall 2001
Midterm #1

Question #1 (7 points):

What will Scheme print in response to the following expressions? If an expression produces an error message or runs forever without producing a result, you may just say "error"; you don't have to provide the exact text of the message. If the value of an expression is a procedure, just say "procedure"; you don't have to show the form in which Scheme prints procedures.

```
(last (se (se '61a 'is) (se 'really 'fun)) )
```

```
(keep (lambda (wd) #f) '(a b c d e))
;KEEP is definied in the lecture notes, p. 128 of the reader
```

```
((lambda (a)
   (lambda (b) (se a b)) 'over)) 'under))
```

```
(if (not (lambda () 10))
    (/ 1 0)
    'scheme)
```

```
(let ((a 5)
    (b (+ a 3)))
    (* a b))
```

```
((lambda (a b) ((if (< b a) + *) b a)) 4 6)
```

```
(first '(word 'a 'b 'c))
```

Problem #2 (10 points):

Write a procedure named funtion-combiner that takes a list of functions as an argument Function-combiner will return a funtion of one argument that applies the first function to the result of applying the second function to ... applying the last function to the argument.
Here are some sample interactions:

```
> (define (square x) (* x x))
> (define (1+ x) (+ x 1))
> (define square-1+ (function-combiner (list square 1+)))
> (square-1+ 10) ;;returns (square (1+ 10))
121
```

```
> ((function-combiner (list list first bf bl)) '(first second third fourth))
(second)

> ((function-combiner '()) 'hello)
hello

> ((function-combiner (list cader)) '(here is a list))
list
```

Fill in the definitions:

```
(define (function-combiner
```

(b) What is the result of ((function-combiner '(car)) '(this is a list))? why ?

Problem #3 (8 points):

```
(Define (garply n)   (if (< n 20)
   n
   (+ (foo n)
      (garply (- n 1)) )) )
```

Assuming foo is defined somewhere, please circle TRUE or FALSE. and in one sentence explain your choice.

True or False: We have enough information to determine the order of growth of garply.

True or False: NO matter how foo is defined, garply will always have an order of growth greater than or equal to THETA(n)

True or False: garply has an order of growth in Theta(n^2) if foo is defined as follows:

```
(define (foo n)   (if (< n 100)      121      (+ (* n 100) (foo (- n 1)))))
```

True or False: garply generates a iterative process.

Problem #4 (8 points):

This question concerns the twenty-one game used in the first programming project.
(Assume version without JOkERS)

We wish to determine wheter a certainstrategy depends on the dealer's up-card. For example, the stop-at-17 startegy does not depend on dealer's up car. Conversly, a strategy that hits only if your hand is valued at less than 17 and the dealer shows a 7 or higher (lets call this strategy vegas) does depend on the dealer's card.

the general problem is hard to solve. We'll solve a simpler problem For a given hand, does the stragey depend on the dealer's hand? we are going to write a procedure dealer-sensative? that takes a strategy and a hand as arguments and returns #t if the strategy depdons on the dealer's hand and #f if it doesnt depend on the dealer's hand.

```
> (dealer-sensative? stop-at-17 '(ah 5c 10d))
#f
> (dealer-sensative? vegas '(ah 5c 10d))
#t
```

(a) we first write it using recursion.

```
(define (dealer-sensative? strat hand)

  (define (helper ace-result possible-dealer-cards)

    (cond ((null? possible-dealer-cards) __________________)

        ((equal? ___________  ____________)

        ______________)

        (else _____________________)

   (helper (strat hand 'ah) _________________))
```

(b) Now we'll write dealer-sensative? using higher order procedures. In this problem we assume the function all-same? is defined for you. All-same? takes a list as an argument and returns true if all elements in the list are the same or false otherwise. For example:

```
> (all-same? '(a a a a))
#t

>(all-same? '(a b a a a))
#f
```

```
(define (dealer-sensative? strat hand)

    (let ((results ( ______________________   _______________

         '(ah 2h 3h 4h 5h 6h 7h 8h 9h 10h))))

    ____________________)
```

Don't Define additional helper procedures!

Problem #5 (points):

The registar in Sproul Hall wants to keep a database of students and the courses in which they're enrolled. two abstract data types are used in this program: a course consists of a department such as CS and a number such as 61A. A student consists of a student ID number and a list of courses.

```
(define (make-course dept num)      ; constructor for COURSE   (list dept num))

(define (course-dept course)      ; selector for COURSE   (car course))
```

```
(define (course-num course)      ; selector for COURSE   (cadr course))

(define (make-student sid courses)      ; constructor for STUDENT   (list sid courses))
```

(a) Write the selectors for the student abstract data type

(b) A "geek" is a student enrolled in one or more computer science courses. Write the procedure geek? that takes a student as its argument and returns true if the student is a geek false other wise. RESPECT the data abstractions.

(c)We now change the constructor for student to the following:

```
(define (make-student sid courses)
   (cons courses sid))       ;; two changes on this line!
```

Make all necessary changes so that geek? still works