Sample Quiz 2

Problem 1 (15 points)
For each of the following sets of expressions, write down what the last one will return. If it returns an error, or doesn't return at all, say so. You can assume that each set of expressions is evaluated in a fresh Scheme buffer. Show your work to be eligible for partial credit.

Set 1:  (define f (lambda (y) (lambda (x) (set! x (+ x y)) x)))
       (define g (f 3))
       (g 4)
       (g 5)

Set 2:  (define f (lambda (x) (lambda (y) (set! x (+ x y)) x)))
       (define g (f 3))
       (g 4)
       (g 5)

Set 3:  (define x '(a b c))
       (define y '(d e f))
       (define z (append x y))
       (set-car! x 'foo)
       (set-car! y 'bar)
       z
Problem 2 (18 points)

Draw the box-and-pointer diagrams for r, s, x, and y that will result from evaluating the following expressions:

```
(define r (list 'a 'b))
(define s (list 'c 'd))
(define x (append r s))
(define y (append r s))
(set-car! (cdr x) 'e)
(set-cdr! (cdr (cdr y)) (list r))
```
Problem 3 (22 points)

On the last page of the quiz, you will find an incomplete environment diagram. Tear out that page and use it as a reference to provide values for all of the entries that appear in angle brackets. The value for each entry should be taken from the object titles to the upper left of each object (e.g., GE, E1, P1, etc.). Note that the sequencing of object titles has no relation to the order in which the objects were created.

Assume the statements below are evaluated in order. Fill in the blanks below to give values for the missing pieces of the environment diagram.

```
(define (f1 x)
  (let ((a (* x x)))
    (lambda (m) (+ a x m))))

(define f2 (f1 3))

(f2 4)

(f2 5)
```

<1>: ___________________________

<2>: ___________________________

<3>: ___________________________

<4>: ___________________________

<5>: ___________________________

<6>: ___________________________

<7>: ___________________________

<8>: ___________________________

<9>: ___________________________

<10>: ___________________________

<11>: ___________________________
Problem 4 (25 points)

Consider the following constructor for student records.

```
(define (make-record first last id major year)
  (list (list first last)
        id
        (cons major year)))
```

An example record would be

```
(define alyssa (make-record 'Alyssa 'Hacker 12345 'CS 2003)
```

a) Draw the box and pointer diagram that represents alyssa

b) The data selectors get-first-name, get-last-name, get-id, get-year, and get-major are to be used to extract data from a student’s record. For example, (get-year alyssa) would return the value 2003, Alyssa’s year. Write definitions for these five selectors.
Problem 4 (continued):

c) Define the procedure change-major! that takes a student’s record and a new-major as arguments. Evaluating (change-major! record new-major) should

- Return the id number in record
- Mutate the record to contain new-major
- Change no pointers to record

Do not break abstraction barriers – use your selectors from part b!
Problem 5 (20 points):

Below is a definition for make-inc, a function that creates an accumulator that uses message passing.

```
(define (make-inc init)
  (let ((value init))
    (define (inc-val x)
      (set! value (+ value x)))
    (define (dispatch m)
      (cond ((eq? m 'inc-val) inc-val)
            (else (error "Invalid message – MAKE-INC" m)))
    dispatch))
```

a) Change the code to add a new functionality. We’d like to be able to call the accumulator with the message `reset-val` to reset the counter to 0.

You do not need to rewrite all of the code above. If you want to insert code into the function above, write it here and clearly indicate where it should be put in the code above. (Space was left in the function definition to allow you to do this.)
Problem 5 (continued):

Code is repeated here for your convenience.

```
(define (make-inc init)
    (let ((value init))
        (define (inc-val x)
            (set! value (+ value x)))
        (define (dispatch m)
            (cond ((eq? m 'inc-val) inc-val)
                (else (error "Invalid message - MAKE-INC" m))))
        dispatch))
```

a) Now change the code to add another new functionality. We’d like to be able to call
the accumulator with the message 'set-val to reset the counter to a value passed in by
the user.

You do not need to rewrite all of the code above. If you want to insert code into the
function above, write it here and clearly indicate where it should be put in the code
above. (Space was left in the function definition to allow you to do this.)
TEAR OUT THIS PAGE FOR USE WITH PROBLEM 3.

```
P1
  p: m
  b: <10>

P2
  p: x
  b: (let ((a (* x x)))
       (lambda (m)
                  (+ a x m)))

E1
  x: <4>

E2
  a: <6>

E3
  m: 4

E4
  m: 5
```