3.
```
(let ((x (list 1 2 3 4)))
     (set-car! (cddr x) x)
     x)
```

4.
```
(let ((x (list 1 2 3)))
     (set-car! x (cdr x))
     (set-cdr! (car x) 5)
     x)
```

5.
```
(let ((x (list 1 2 3)))
     (set-car! x (list 'a 'b 'c))
     (set-car! (cdar x) 'd)
     x)
```

Environment Diagram

1. Question 6 of Final, Fall 1997

```
(define (kons a b)
  (lambda (m)
    (if (eq? m 'kar) a b)))
(define p (kons (kons 1 2) 3))
```

2. Question 9 of Final, Fall 1998

```
(define x 3)
(define y 4)
(define foo
    ((lambda (x) (lambda (y) (+ x y)))
     (+ x y)))
(foo 10)
```

List Mutations

1. Question 2 of MT3, Fall 1994

   Write `make-alist!`, a procedure that takes as its argument a list of alternating keys and values, like this:

   ```
   (color orange zip 94720 name wakko)
   ```

   and changes it, by mutation, into an association list, like this:

   ```
   ((color . orange) (zip . 94720) (name . wakko))
   ```

   You may assume that the argument list has an even number of elements. The result of your procedure requires exactly as many pairs as the argument, so you will work by rearranging the pairs in the argument itself. Do not allocate any new pairs in your solution!

2. Question 2 of MT3, Spring 1996

   Write `list-rotate!` which takes two arguments, a nonnegative integer n and a list seq. It returns a mutated version of the argument list, in which the first n elements are moved to the end of the list, like this:

   ```
   > (list-rotate! 3 (list 'a 'b 'c 'd 'e 'f 'g))
   (d e f g a b c)
   ```

   You may assume that $0 \leq$ `n < (length seq)` without error checking.

   Note: Do not allocate any new pairs in your solution. Rearrange the existing pairs.

Vector

1. You've seen vectors. You've seen tables. We want to implement tables with vectors. To make things easier, we're going to assume that there are two vector tables that manage keys and values. Assume that the corresponding key/value pair has the same index.

   Write `vector-lookup` for vector-tables. It acts just like lookup in tables. It takes a key and returns the corresponding value if the key is in the table and `#f` otherwise.

2. Write `vector-reverse!` that does the obvious thing. Do not create new vectors!

   Hint: You might want to write a helper called `vector-swap!` that takes in a vector and two indices and swap the elements at those indices.

Concurrency

1. What are the possible values of x after the following is executed:

```
(define x 10)
(parallel-execute (lambda() (set! x (+ 5 x)) (set! x (* x 3)))
                  (lambda() (if (> x 16)
                                (set! x 100)
                                (set! x (- x 20)))))
```

2. Question 13 of Final, Spring 2003

   Given the following definitions:

```
(define s (make-serializer))
(define t (make-serializer))
(define x 10)
(define (f) (set! x (+ x 3)))
(define (g) (set! x (* x 2)))
```

   Can the following expressions produce an incorrect result, a deadlock, or neither? (By "incorrect result" we mean a result that is not consistent with some sequential ordering of the processes.)

   (a) `(parallel-execute (s f) (t g))`

   (b) `(parallel-execute (s f) (s g))`

   (c) `(parallel-execute (s (t f)) (t g))`

   (d) `(parallel-execute (s (t f)) (s g))`

   (e) `(parallel-execute (s (t f)) (t (s g)))`