Sample midterm 3 #1

Problem 1 (box and pointer).

What will the Scheme interpreter print in response to **the last expression** in each of the following sequences of expressions? Also, draw a "box and pointer" diagram for the result of each printed expression. If any expression results in an error, **circle the expression that gives the error message.** Hint: It'll be a lot easier if you draw the box and pointer diagram *first*!

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

```
(define x 3)
(define m (list x 4 5))
(set! x 6)
m
```

```
(define x (list 1 '(2 3) 4))
(define y (cdr x))
(set-car! y 5)
x
```

```
(let ((x (list 1 2 3)))
  (set-cdr! (cdr x) x)
  x)
```

Problem 2 (Assignment, State, and Environments).

Suppose we want to write a procedure `prev` that takes as its argument a procedure `proc` of one argument. `Prev` returns a new procedure that returns the value returned by *the previous call to* `proc`. The new procedure should return `#f` the first time it is called. For example:

```
> (define slow-square (prev square))
> (slow-square 3)
#f
> (slow-square 4)
9
> (slow-square 5)
16
```

Which of the following definitions implements `prev` correctly? **Pick only one.**

```
______ (define (prev proc)
         (let ((old-result #f))
           (lambda (x)
             (let ((return-value old-result))
               (set! old-result (proc x))
               return-value))))
```

```
______ (define prev
         (let ((old-result #f))
           (lambda (proc)
             (lambda (x)
               (let ((return-value old-result))
                 (set! old-result (proc x))
                 return-value)))))
```

```
______ (define (prev proc)
         (lambda (x)
           (let ((old-result #f))
             (let ((return-value old-result))
               (set! old-result (proc x))
               return-value))))
```

```
______ (define (prev)
         (let ((old-result #f))
           (lambda (proc)
             (lambda (x)
               (let ((return-value old-result))
                 (set! old-result (proc x))
                 return-value)))))
```

Problem 3 (Drawing environment diagrams).

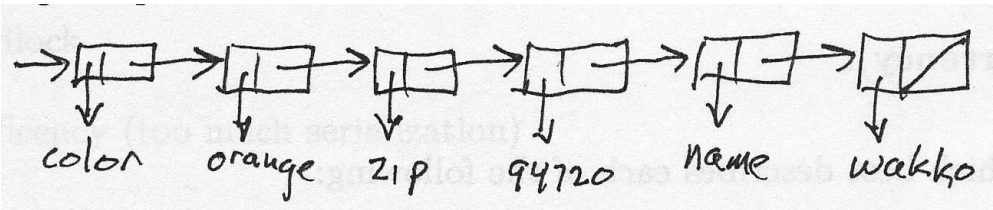
Draw the environment diagram resulting from evaluating the following expressions, and show the result printed by the last expression where indicated.

```
> (define x 4)

> (define (baz x)
    (define (* a b) (+ a b))
    (lambda (y) (* x y)))

> (define foo (baz (* 3 10)))

> (foo (* 2 x))
```

Problem 4 (List mutation).

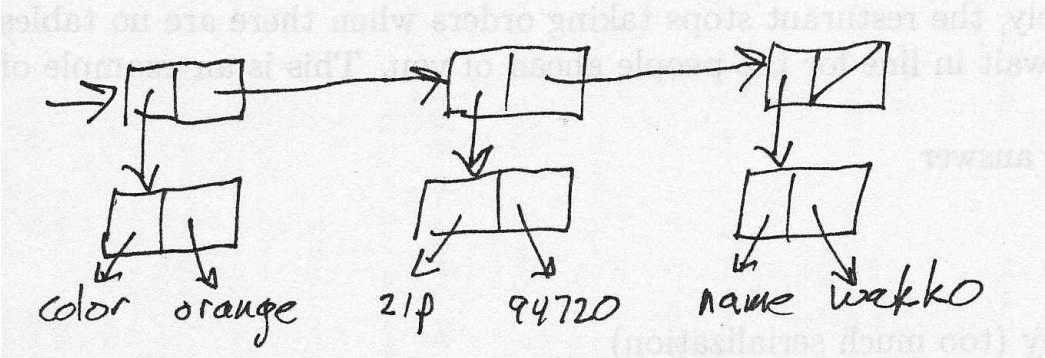
Write `make-alist!`, a procedure that takes as its argument a list of alternating keys and values, like this:

```
(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!**

Problem 5 (Vectors).

Suppose there are N students taking a midterm. Suppose we have a vector of size N, and each element of the vector represents one student's score on the midterm. Write a procedure `(histogram scores)` that takes this vector of midterm scores and computes a histogram vector. That is, the resulting vector should be of size M+1, where M is the maximum score on the midterm (it's M+1 because scores of zero are possible), and element number I of the resulting vector is the number of students who got score I on the midterm.

For example:

```
> (histogram (vector 3 2 2 3 2))
#(0 0 3 2) ;; no students got 0 points, no students got 1 point,
           ;; 3 students got 2 points, and 2 students got 3 points.
> (histogram (vector 0 1 0 2))
#(2 1 1) ;; 2 students got 0 points, 1 student got 1 point,
         ;; and 1 student got 2 points.
```

Do not use `list->vector` or `vector->list`.

Note: You may assume that you have a procedure `vector-max` that takes a vector of numbers as argument, and returns the largest number in the vector.

Problem 6 (Concurrency).

Choose the answer which best describes each of the following:

(a) You and a friend decide to have lunch at a rather popular Berkeley resturant. Since there is a long line at the service counter, each group of people entering the restaurant decide to have someone grab a table while someone else waits in the line to order the food. This sounds like a good idea, so your friend sits down at the last free table while you get in line. Unfortunately, the resturant stops taking orders when there are no tables available, and you have to wait in line for the people ahead of you. This is an example of

_____incorrect answer

_____deadlock

_____inefficency (too much serialization)

_____unfairness

_____none of the above (correct parallelism)

(b) After you finally get to eat lunch, you and your friend decide to go to the library to work on a joint paper. The library has a policy that students who enter the library must open their backpacks to show that they are not bringing food into the library. They have several employees doing backpack inspection, so there are several lines for people waiting to be inspected. However, today there was a bomb threat, and so the inspectors also use a handheld metal detector to examine the backpacks. Although there are several inspectors, the library only has one metal detector. This is an example of

_____incorrect answer

_____deadlock

_____inefficency (too much serialization)

_____unfairness

_____none of the above (correct parallelism)

(c)While you are working on the paper, your friend decides to do some research for your paper and leaves for a few hours while you continue writing. This is an example of

_____incorrect answer

_____deadlock

_____inefficency (too much serialization)

_____unfairness

_____none of the above (correct parallelism)

Problem 7 (Streams).

In weaving, a horizontal thread is pulled through a bunch of vertical threads. The horizontal thread passes over some of the vertical ones, and under others. The choice of over or under determines the pattern of the weave.

We will represent a pattern as a list of the words `OVER` and `UNDER`, repeated as needed. Here's an example:

```
(OVER OVER UNDER OVER UNDER UNDER)
```

The pattern may be of any length (it depends on the desired width of the woven cloth), but it must contain at least one `OVER` and at least one `UNDER`.

Write a Scheme expression to compute the (infinite) stream of all possible patterns.