Quiz 9a

1. (3 points) What will the Scheme interpreter print in response to each of the following expressions? Also, draw a “box and pointer'' diagram for the result of each printed expression. If any expression results in an error, just write “ERROR''; you don't have to give the precise message. Hint: It'll be a lot easier if you draw the box and pointer diagram *first*!

(let ((x (list 1 2 3))

(y (list 4 (list 5) 6)))

(set-cdr! (cadr y) x)

(set-car! x (caadr y))

y)

(let ((x (list 'cs 'is 'cool))

(y (list 'x)))

(set-cdr! (cdr y) x)

y)

2. (4 points) Implement a procedure lists->assoc! that takes a list of keys and a list of values as its arguments and, using mutation (set-car! and set-cdr!), creates a single association list.

> (define L1 (list 1 2 3))

L1

> (define L2 (list 'a 'b 'c))

L2

> (lists->assoc! L1 L2)

okay ;; return value is ignored

> L1

((1 . a) (2 . b) (3 . c))

The pairs of the first argument should become the spine pairs of the result; the second argument need not be preserved. Your implementation may **not** create any new pairs! You may assume the two arguments are always lists of equal length.

(define (lists->assoc! keys values)

3. (3 points) You are given a binary tree, in which the nodes are represented in the form

indicated by these selectors:

(define datum car)

(define left-branch cadr)

(define right-branch cddr)

The empty tree is represented by the empty list.

Write traverse!, a procedure that takes a binary tree as its argument, and rearranges the pairs to form an inorder traversal --- a linear sequence of the data from the tree, in left-to-right order. (If the tree is a binary search tree, for example, then the result will be a sorted sequence.)

A binary tree with *N* nodes contains *2N* pairs. You will string together the pairs containing the data, and discard the pairs containing the branch pointers (after you've collected the data from those branches).

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

Note: In this problem you are changing a structure's abstract data type, from tree to sequence. In such situations, data abstraction doesn't make much sense; just use car, set-cdr!, etc.