CS 61A, Spring 1999
Midterm #2
Professor Brian Harvey

Question 1 (4 points):

What will Scheme print in response to the following expressions? If an expression produces an error message, 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. **Also, <u>draw a box and pointer diagram</u> of the value produced by each expression.**

```
(cons '(a b) '(( c d)))
```

```
(cdadr '((1 2 3) (4 5 6) (7 8 9)))
```

```
(cons (list '(a) '(b)) (list 'c 'd))
```

```
(cons '(a b) 'c)
```

Question 2 (5 points):

A *three-tree* is a tree-like structure in which each node contains *two* values, called the **left-datum** and the **right-datum**, and up to three children, called the **left-branch**, the **middle-branch**, and the **right-branch**. (Any of these can be an empty list instead of a three-tree.)

(a) Write the constructor (make-3 lft-dat rt-dat lft-br mid-br rt-br) and the appropriate selectors so that a three-tree node looks like this:

```
    ___ ___     ___ ___     ___ ___     ___ ___
   |   |   |   |   |   |   |   |   |   |   |  /|
-->| | | --|-->| | | --|-->| | | --|-->| | | / |
   |_|_|___|   |_|_|___|   |_|_|___|   |_|_|/__|
     |           |           |           |
    \|/         left        middle      right
    _|_ ___     branch      branch      branch
   |   |   |
   | | | --|-->right
   |_|_|___|   datum
     |
    left
    datum
```

(b) An *LR-three-tree* is a three-tree in which the data are numbers, the left-datum is less than the right-datum, and all of the (nonempty) children are LR-three-trees.

Write the predicate **LR-three-tree?** that takes as its argument and returns true if and only if it's an LR-three-tree.

Respect the data abstraction.

Question 3 (5 points):

Ben Bitdiddle, who used to program in C++, is upset that to select the element at a particular position in a sequence, a Scheme programmer has to type (list-ref seq 5), whereas a C++ programmer using an array to represent the sequence has merely to type seq[5].

Alyssa offers to help him out by creating an array abstract type, implemented using message passing. It'll work like this:

```
> (define a1 (make-array '(lucy in the sky with diamonds)))
> (a1 3)
SKY
> (a1 'length)
6
> (a1 'with)
4
> (a1 'friends)
#F
```

If given the word **length** as a message, an array will return the number of elements it has. If given a number as a message, it will return the element at that position, counting from zero. If given any other message, if the message is an element of the array, it returns the position at which the element is found; otherwise it returns false.

Write **make-array**.