from last time: set-car!, set-cdr!

trees
For the box & pointer diagram:
(a) Write what Scheme prints out for the structure (if it can)
(b) Write a Scheme expression that makes the structure (if an error, describe it)
(c) Draw the structure that results from the mutation, and its printed representation.

1. a. \( x \Rightarrow \)

b. Scheme expression:

\[ \text{c. mutation: (set-cdr! (car x) '(8))} \]
more set-car! and set-cdr! problems

For the box & pointer diagram:
(a) Write what Scheme prints out for the structure (if it can)
(b) Write a Scheme expression that makes the structure (if an error, describe it)
(c) Draw the structure that results from the mutation, and its printed representation.

2. a.    x  =>

b. Scheme expression:

c. mutation: (set-cdr! (cddr x) (caar x))
trees

- A tree is a nested list; each node is a list of the children of that node
- A child is either another tree or a leaf node
  - A child that is a tree is called a *subtree*
  - A leaf node is anything that is not a pair (i.e., a symbol or a self-evaluating value).
1. Draw a box-and-pointer structure for the following tree. How does the interpreter print this structure?
2a. Draw the interpretation of this list as a tree structure: (((1 2) 3) (4 (5 6) 7 (8 9 10)))

2b. Draw the box-and-pointer diagram.
counting leaves

(define (countleaves tree)
  (cond ((null? tree) 0)
        ((leaf? tree) 1)
        (else (+ (countleaves (car tree))
                 (countleaves (cdr tree))))))

(define (leaf? x)
  (not (pair? x)))

(1 ((2 3) 4) 5 6)
doubling a tree: version 1

(define (countleaves tree)
  (cond ((null? tree) 0)
       ((leaf? tree) 1)
       (else (+ (countleaves (car tree))
                (countleaves (cdr tree))))))

(define (leaf? x)
  (not (pair? x)))

(define (double-tree tree)
  (cond ((null? tree) '())
       ((leaf? tree) )
       (else ( (car tree))
              ( (cdr tree))))

(1 ((2 3) 4) 5 6) => (2 ((4 6) 8) 10 12)
doubling a tree: version 2, map

v. 1 (define (double-tree tree)
  (cond ((null? tree) '())
        ((leaf? tree) (leaf? tree))
        (else ( (car tree))
              (cadr tree))))

v. 2 (define (double-tree tree)
  (if (leaf? tree)
       (leaf? tree))
  (else ( (car tree))
        (cadr tree)))))

(1 ((2 3) 4) 5 6) => (2 ((4 6) 8) 10 12)
doubling a tree: version 3, map-tree

v. 1 (define (double-tree tree)
  (cond ((null? tree) `())
        ((leaf? tree) )
        (else ( (car tree))
               (cdr tree))))

(define (map-tree proc tree)
  (if (leaf? tree)
      (proc tree)
      (map-tree proc tree)))

v. 2 (define (double-tree tree)
  (if (leaf? tree)
      (double (car tree))
      (double-tree (cdr tree))))

(define (double x)
  (* 2 x))

(define (double-tree tree) )
A binary tree is one in which each node is represented by an entry and a link.

The “left” link points to elements smaller than node entry.

The “right” link points to elements larger than node entry.

To check where an element is in a set:
- compare x with an entry
- if x is less than entry, search left subtree; if greater, search right subtree

Two trees that represent the set \{1, 3, 5, 7, 9, 11\}: