

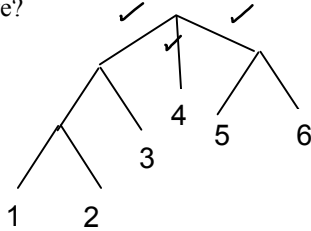
**Tree Problem Solutions**

Dr. Kimberle Koile

**Trees as Nested Lists**

A conventional representation of trees is achieved using a nested list structure. Each node in the tree is represented as a list of the children of that node, where a child may be either another tree or a leaf node. A child node that is a tree is called a subtree. A leaf node is anything that is not a pair (e.g., a symbol or a self-evaluating value).

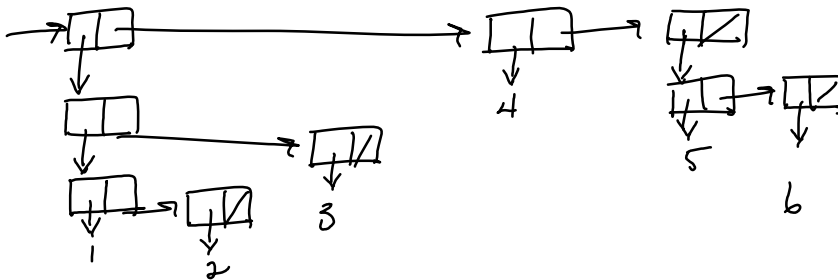
1. Draw a box-and-pointer structure for the following tree using this convention. How does the interpreter print this structure?



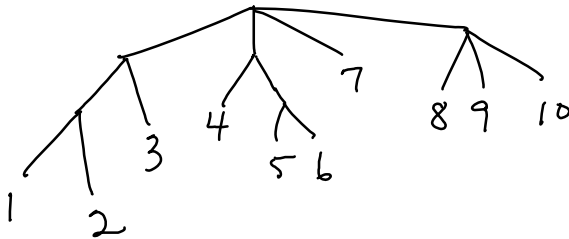
Printed representation:

$(( (1 2) 3) 4 (5 6))$

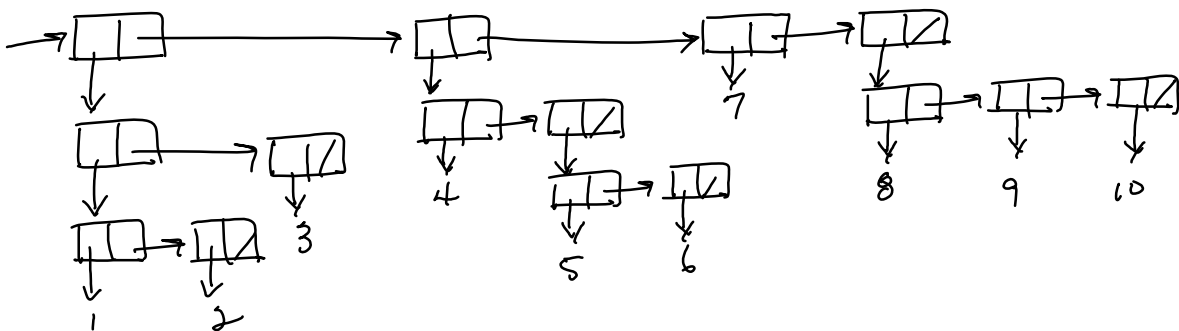
Box-and-pointer diagram:



2. Draw the interpretation of this list as a tree structure:  $(( (1 2) 3) (4 (5 6)) 7 (8 9 10))$



Draw the box-and-pointer diagram:



3. Fill in the procedure for `double-tree` that returns a new tree (in the list representation) with double the value of all leaf nodes. Recall that you check for a leaf node with this procedure:

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

(define (double-tree tree)
  (cond ((null? tree) nil)
        ((leaf? tree) (* 2 tree))
        (else (cons (double-tree (car tree))
                     (double-tree (cdr tree))))))
```

4. An advantage of representing trees as lists is that we can use list procedures. Write the `double-tree` procedure using the `map` procedure.

```
(define (double-tree tree)
  (if (leaf? tree)
      (* 2 tree)
      (map double-tree tree)))
```

Note that in this case we don't have to check for null, since `map` is already doing so.

5. Recall the `tree-map` procedure, which will perform some operation on all the leaf nodes of a tree, e.g. `(tree-map double mytree)`.

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

Why can't we just use the procedure `tree-map` as the second argument to `map`?

We can't just pass `maptree`, since we would have an argument mismatch (the procedure passed to `map` must take only one argument). So we have to use an anonymous procedure that has the `proc` variable buried inside it.

Alternate `tree-map`:

```
(define (tree-map proc tree)
  (if (leaf? tree)
      (proc tree)
      (cons (tree-map proc (car tree))
            (tree-map proc (cdr tree))))))
```