

## Deep Recursion

**Summary:** In this laboratory, you will further explore issues of deep recursion introduced in the reading on pairs and pair structures and continued in the reading on deep recursion.

### Contents:

- Exercises
  - Exercise 0: Preparation
  - Exercise 1: Number Trees
  - Exercise 2: A Number Tree Predicate
  - Exercise 3: Summing Number Trees
  - Exercise 4: Preconditions
  - Exercise 5: Counting Cons Cells
- For Those with Extra Time
  - Extra 1: Cons Cells vs. Values
  - Extra 2: Searching for Symbols
- Notes
  - Notes on Exercise 3
  - Notes on Exercise 5

## Exercises

### Exercise 0: Preparation

- a. Make sure that you have the reading on pairs and pair structures and the reading on deep recursion open in separate tabs and windows.
- b. Make sure that you have a piece of paper and writing instrument handy.

### Exercise 1: Number Trees

Recall that a list is a data structure defined recursively as follows:

- The empty list is a list.
- Cons of a value and a list is a list.

In the reading on pairs and pair structures, the section entitled “Recursion with Pairs” includes a procedure that works on “number trees”, nested structures built with the `pair` procedure.

Write a recursive definition for *number trees*, trees built from only numbers and cons cells, similar to that for lists.

## Exercise 2: A Number Tree Predicate

Using your recursive definition of number trees from the previous problem, write a procedure, `(number-tree? val)` that returns true if `val` is a number tree and false otherwise.

## Exercise 3: Summing Number Trees

Consider again the `sum-of-number-tree` procedure from the reading, which you can find repeated at the end of this lab.

a. Verify that it works as advertised on the first example.

```
(sum-of-number-tree (cons (cons (cons 0 1)
                               (cons 2 3))
                          (cons (cons 4 5)
                                (cons 6 7))))
```

b. What do you expect `sum-of-number-tree` to return when given `(cons 10 11)` as a parameter? Verify your answer experimentally.

c. Verify that it works as advertised on a single number.

d. Verify that it works as advertised on a pair of numbers.

e. What do you expect `sum-of-number-tree` to return when given the empty list as a parameter? Verify your answer experimentally.

f. What do you expect `sum-of-number-tree` to return when given `(list 1 2 3 4 5)` as a parameter? Verify your answer experimentally.

## Exercise 4: Preconditions

a. What preconditions should `sum-of-number-tree` have?

b. Use the `number-tree?` predicate from earlier to rewrite `sum-of-number-tree` so that it reports an appropriate error if its preconditions are not met.

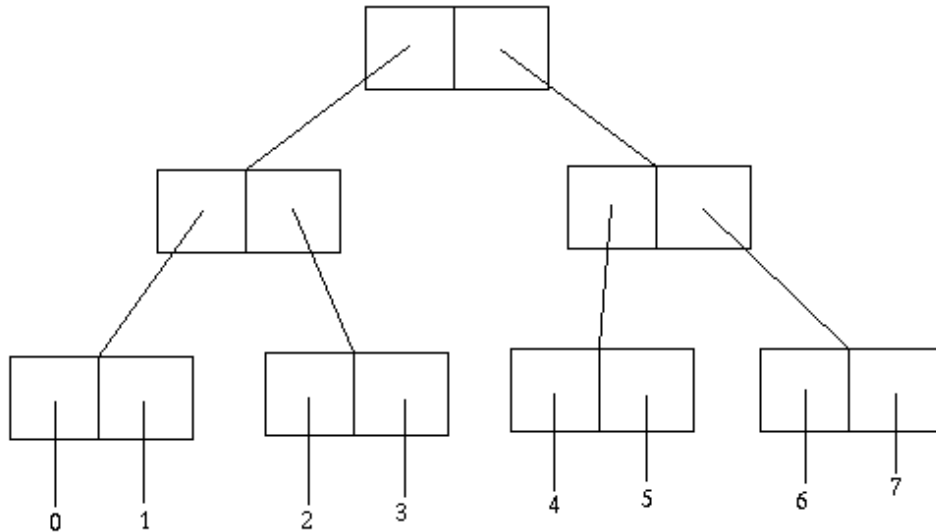
c. Some programmers consider it inappropriate to scan a tree twice, once to make sure that it's valid and once to compute a value based on the tree. Rewrite `sum-of-number-tree` so that it checks for and reports errors only when it is at one of the non-pair values.

## Exercise 5: Counting Cons Cells

a. Define and test a procedure named `cons-cell-count` that takes any Scheme value and determines how many boxes would appear in its box-and-pointer diagram. (The data structure that is represented by such a box, or the region of a computer's memory in which such a structure is stored is called a "*cons cell*". Every time the `cons` procedure is used, explicitly or implicitly, in the construction of a Scheme value, a new `cons cell` is allocated, to store information about the `car` and the `cdr`. Thus

`cons-cell-count` also tallies the number of times `cons` was invoked during the construction of its argument.)

For example, the structure in the following box-and-pointer diagram contains seven cons-cells, so when you apply `cons-cell-count` to that structure, it should return 7. On the other hand, the string "sample" contains no cons-cells, so the value of `(cons-cell-count "sample")` is 0.



In answering this question, you should consider whether each value, in turn, is a pair using the `pair?` predicate.

b. Use `cons-cell-count` to find out how many cons cells are needed to construct the list

```
(0 (1 (2 (3 (4))))))
```

See the notes at the end of the lab if you have trouble creating that list.

c. Draw a box-and-pointer diagram of this list to check the answer.

## For Those with Extra Time

If you find that you have extra time, you might want to attempt one or more of the following problems.

## Extra 1: Cons Cells vs. Values

As you may recall, a tree is either (a) a non-pair value or (b) the cons of two trees. In the reading, you saw a procedure that counted the number of values in a tree. In this lab, you wrote a procedure that counted the number of cons cells (pairs) in a tree. What is the relationship between the numbers returned by those two procedures?

## Extra 2: Searching for Symbols

Write a procedure, `(tree-member? sym tree-of-symbols)`, that determines whether `sym` appears anywhere in `tree-of-symbols`.

## Notes

### Notes on Exercise 3

In case you don't want to switch documents, here is the code for `sum-of-number-tree`.

```
;;; Procedure:
;;;   sum-of-number-tree
;;; Parameters:
;;;   ntree, a number tree
;;; Purpose:
;;;   Sums all the numbers in ntree.
;;; Produces:
;;;   sum, a number
;;; Preconditions:
;;;   ntree is a number tree. That is, it consists only of numbers
;;;   and cons cells.
;;; Postconditions:
;;;   sum is the sum of all numbers in ntree.
(define sum-of-number-tree
  (lambda (ntree)
    (if (pair? ntree)
        (+ (sum-of-number-tree (car ntree))
           (sum-of-number-tree (cdr ntree)))
        ntree)))
```

### Notes on Exercise 5

If, for some reason, you are having trouble creating the list

```
(0 (1 (2 (3 (4)))))
```

try

```
(list 0 (list 1 (list 2 (list 3 (list 4)))))
```

---

Copyright © 2007 Samuel A. Rebelsky. This work is licensed under a Creative Commons

Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.