HIERARCHICAL DATA 5

COMPUTER SCIENCE 61AS

Concepts and Definitions

1. What is a Tree? How is it different from a Deep List? When would you use one over the other?

2. What is mutual recursion?

3. When working with trees, what is (typically) the input to each mutually recursive procedure?

4. What is a binary tree? Which selectors do we use for it?

5. What is car/cdr recursion?

Practice with Capital-T Trees

1. Assume `forest` is a forest (list of Trees) and `tree` is a Tree where each element is a number. For each of the following lines of code, say whether it is a data abstraction violation, a domain/range mismatch, or valid.

   a. `(car forest)`

b. `(car tree)`

c. `(children tree)`

d. `(children forest)`

e. `(null? (children tree))`

f. `(cdr (children tree))`

2. Write a procedure `add-child-lengths` which takes as input a tree, and produces as output a new tree in which each datum is now a list whose first element is the original datum, and whose second element is the number of children it has. (You can find the length of a list using the `length` procedure.) Use mutual recursion here.

3. Write `sum-tree`, which takes as input a tree of numbers and produces as output the sum of all the numbers in the tree. You should use sequence operations (`map, filter, accumulate`) to implement it. No helper procedures allowed! (This is a hard question to start with, so you may want to first do it with mutual recursion, which is easier.)

4. `listify-tree` takes as input a tree and produces as output a flat (not deep) list of all of the datums in the tree, in any order.

   a. Write `listify-tree` using higher order functions.

   b. Write `listify-tree` using mutual recursion.

5. Write `count-leaves`, which returns the number of leaves in the input tree. A leaf is any tree which has no children.

Practice with HOFs and Deep Lists

1. Louis Reasoner writes a procedure, `deep-squares`, that takes in a deep list and squares every number in it. He writes the following code.

```
(define (deep-squares lol)
    (cond ((null? lol) '())
          ((list? (car lol))
           (cons (map square (car lol))
                 (deep-squares (cdr lol)) ))
          (else (cons (square (car lol))
                      (deep-squares (cdr lol)) ))))
```

   a. Say whether the code above will error, work, or is a data abstraction violation.

```
(deep-squares '())
(deep-squares '(1 (2 3) (4 5))
(deep-squares '(1 (2 (3) 4) (((5))))
```

   b. Now fix Louis's code. Hint: it can be fixed with an extremely small change. Do NOT use deep-map in your solution.

2. Express the following function without using recursion. (Hint: use `map`).

```
(define (jelly-words lst)
    (if (null? lst)
        '()
        (cons (word 'jelly (car lst))
              (jelly-words (cdr lst)))))

> (jelly-words '(fish bean donut car cdr) )
(jellyfish jellybean jellydonut jellycar jellycdr)
```

3. Express the following function without using recursion.

```
(define (sqrt-positives nums)
    (cond ((null? nums) '())
          ((<= (car nums) 0) (sqrt-positives (cdr nums)))
          (else (cons (sqrt (car nums))
                      (sqrt-positives (cdr nums))))))

> (sqrt-positive '(1 -9 5 -4))
(1 25)
```

Practice with Car/cdr Recursion

1. Write the procedure `sum-binary-tree`, which sums the datum of a binary tree.

2. Now, write the procedure `sum-cons-structure`, which should work on any structure created by `cons`. Would `(sum-cons-structure bt)` evaluate to the same thing as `(sum-binary-tree bt)` (assuming that `bt` is a valid input to `sum-binary-tree`)? If yes, why don't we generally do this? If no, why not?