Quiz 5a

1. (3 points) The following function operates on the Tree abstract data type as discussed in lecture:  
   (define (mystery tree)  
    (if (null? (children tree))  
    (make-tree (datum tree) '())  
    (make-tree (+ (datum tree) 100)  
    (map mystery (cdr (children tree))))))  
     
   Draw the result of calling mystery with the tree below as its argument:

1

/ | \

2 3 4

| / \ |

5 6 7 8

/ \

9 10

101

| \

103 104

|

7

This question concerns the Tree abstract data type, as discussed in lecture, implemented as follows:

(define make-tree cons)

(define datum car)

(define children cdr)

True or false: Every nonempty list can be interpreted as the representation of a Tree.

\_\_\_\_True \_\_X\_\_False

1. a) (2 points) The scheme function (map f l) returns a list made by applying the function f to each element of the list l. Write an analogous function (maptree f t) that returns a tree made by applying the function f to each entry of the binary tree t. Since the tree is a data abstraction, you may only use the following to operate on trees: (entry t), (right-branch t), (left-branch t), (make-tree entry left right), and (empty-tree? t). You may use the predefined constant the-empty-tree. Example:   
     
   > (define tree

(make-tree 10 (make-tree 5 the-empty-tree the-empty-tree)

(make-tree 12 the-empty-tree the-empty-tree)

tree

> tree

(10 (5 () ()) (12 () ()))

> (maptree square tree)

(100 (25 () ()) (144 () ()))

(define (maptree fn tree)

(if (equal? tree the-empty-tree)

the-empty-tree

(make-tree (fn (entry tree))

(map-tree fn (left-branch tree))

(map-tree fn (right-branch tree)))))

2 points – Perfect or minor mistake

1 point – close but not there

b) (1 point) A tree is a data abstraction that, like a list, can be used to store any type of data. Assume we have defined the variable sonias-tree that contains the complex numbers 3+4i, -1-4i, and 10i. Write a Scheme expression **WITHOUT USING DEFINE** that will add the complex number 3+3i to each entry in the tree sonias-tree. You may assume that the function +c is already defined. You may also assume that the constructor (make-complex real imag) is also defined.

(map-tree (lambda (x) (+c x (make-complex 3 3))) sonias-tree)

1 point all or nothing

1. (4 points) This question is about binary trees, as defined in SICP:   
   Constructor: (make-tree entry left right)   
   Selectors: (entry tree) (left-branch tree) (right-branch tree)   
   We'll call a binary tree full if all leaves are at the same depth, and there are no missing nodes. Your goal is to take a tree that is not full and fill it.  
   (a) Write make-filler, which takes a nonnegative integer as its argument. If the argument is zero, it should return the empty list.   
   Otherwise, it should return a full binary tree, with zeros in every entry, and with the number of levels specified by the argument. Some examples will help make this clear (showing pictures of trees rather than the representation as printed by Scheme):  
   > (make-filler 1)  
   0  
   > (make-filler 2)  
    0  
    / \  
   0 0  
     
     
   > (make-filler 3)  
    0  
    / \  
    0 0  
    / \ / \  
   0 0 0 0  
     
     
   (define (make-filler n)  
    (if (= n 0)  
    ‘()  
    (make-tree 0 (make-filler (- n 1)) (make-filler (- n 1)))))  
     
   (b) Now we can write fill, which takes in a tree and returns a filled version of that tree.   
   Someone has written depth for us; it returns the maximum depth of the tree. (The depth of a tree with only a root node is zero.) We've given you the procedure below as a starter. You write fill-help.  
     
   (define (fill tree)  
    (fill-help tree (depth tree)))

(define (fill-help tree d)

(if (null? tree)

(make-filler (+ 1 d))

(make-tree (entry tree)

(fill-help (left-branch tree) (- d 1))

(fill-help (right-branch tree) (- d 1))

1 point for make-filler

3 points for fill-help perfect

2 points for fill-help mostly correct

1 point for fill-help being on right track