Week 7 – 04-03-04
Midterm 2 Review

Topics to Cover:

1. Scheme
2. List Recursion
3. Deep List Recursion
4. Abstract Data Types
5. Trees
6. Data Directed Programming
7. Message Passing
8. Scheme-0 & Scheme-1

Scheme Questions

What will Scheme print in response to the following expressions? If an expression produces an error or runs forever without producing a result, just say "error"; If the value of an expression is a procedure, just say "procedure."

1. (let ((y '(fame)))
   (cons (append y (list y)) y))

2. (caddar (list (cons 4 (list 7 5)) 89 42 (list 3)))

3. (map car (list 'the 'matrix 'has 'you))

4. (car '(word 1 2 3 4 5))

5. (map - (filter number? '(the 1 after 999)))

6. (car ((lambda (x y z) (list (y z z) (x z z))) '- '+ 4))

7. (cons '(a b) (list '(c d) 'e)) *also draw the box and pointer*

8. (define best-class-at-cal
   (list (cons '(c) '(s)) (append '(6) '(1)) 'a))

   What does Scheme return when best-class-at-cal is typed into the interpreter?

   Draw the box-and-pointer diagram.

   Define best-class-at-cal again, only this time using call to cons.

9. We'd like to write the LIST? predicate. It's a function that takes a single argument which can be of *any* Scheme type and returns #t if it is a list, #f otherwise:

   STk> (list? '(a b c))

```
#t
STk> (list? (cons 2 3))                ;; (2 . 3)
#f
STk> (list? (lambda (x) (* x x)))      ;; a procedure is not a list
#f
```

Here is our first attempt:

```
(define (list? thing)
   (or (null? thing)
       (list? (cdr thing))))
```

It has a bug. Find it and fix it.

10. Given,

```
(define (mystery x . args)
   (if (list? x)
      args
      (apply mystery (map list args))))
```

What does (mystery 'a 'b 'c 'd) return?

List Recursion

11. Write the function DIAGONAL, which takes a list of lists. It should return the first element of the first sublist, the second element of the second sublist, and so on:

```
STk> (diagonal '((a b c d) (1 2 3) (aa bb cc dd ee ff)))
(a 2 cc)
```

Do not define any helpers.

12. My friend was trying to write DUPLS-REMOVED, which takes a list and returns a list with all the duplicate words removed, using FILTER. It should work like this:

```
STk> (dupls-removed '(foo bar baz bar foo 2))
(foo bar baz 2)
```

Here is what he's got:

```
(define (dupls-removed lst)
    (filter (lambda (wd) (not (member wd lst))) lst))
```

Does it work? YES NO
Explain.

13.Write the function FLATTEN-ONCE that takes a list. It should return a list with all the toplevel sublists flattened by one, and any non-pair toplevel elements removed:

```
STk> (flatten-once '((a (b) c) (1 2 3) a b))
(a (b) c 1 2 3)
STk> (flatten-once '(hello (there) fred))
(there)
STk> (flatten-once '(a b c d))
()
STk> (flatten-once '(one (two) ((three)) (((four)))))
(two (three) ((four)))
```

Do not write any helper procedures, and do not use APPLY.

Write a version using recursion, and one using higher-order functions (MAP, ACCUMULATE/REDUCE, FILTER) and no recursion.

Deep List Recursion

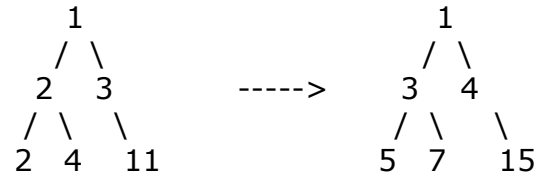
14.Define the depth of an element as the number of car it takes to get the element out. For example each element, x, below has a depth of x.

(a (b (c (d) e) f) g)
*where a & g both have depth of 1 and b & f have depth of 2, etc..

Write a procedure (depth N L) where N is a number and L is a deep list. depth returns either the depth of N (if N is in the list), or #f if N is not in the list. You may assume N appears at most once.

Trees

15.Using the datum children representation, write cost. Cost takes a tree and returns a new tree where every node in the new tree is the result of adding all of the node's parent's datum in the old tree...



Write it both with a helper and one without.

Abstract Data Types

16.An **athlete** contains 3 pieces of information: **name** (a word), **country** (a word), and **events** (a sentence containing the names of all the events the athlete participates in). The constructor:

```
(define (make-athlete name counter events)
    (append (list name country) (cons events '())))
```

Write the appropriate selectors for the athlete ADT

```
(define all-athletes
    (list
        (make-athlete 'Picabo-Street 'USA (se 'slalom 'downhill))
        (make-athlete 'Todd 'USA (se 'slalom 'super-monkey-ball))
        (make-athlete 'Johnny-Moseley 'USA (se 'moguls 'video-games))
  …more athletes… ))
```

Complete the definition given so that **get-participants** returns a sentence containing the names of all the athletes who participate in that event.

> (get-participants 'slalom all-athletes) --> (Picabo-Street Todd)

```
(define (get-participants event list-of-athletes)
    (cond ((_____ list-of-athletes) '())
    ((member? event (_____ (_____ list-of-athletes)))
     (_____ (_____ (_____ list-of-athletes))
                (get-participants event (_____ list-of-athletes))))
    (else (get-participants event (_____ list-of-athletes)))) )
```

17. Complete diagonal, which takes a list of sentences, returns the words on the diagonal.

```
(define (diagonal lstsents)
    (if (_____ lstsents)
        '()
        (_____ (_____ (_____ lstsents))
                (diagonal (chop (_____ lstsents))))))

(define (chop lstsents) ;; removes the first word from each sentence
    (if (_____ lstsents)
        '()
        (_____ (_____ (_____ lstsents))
                (chop (_____ lstsents)))))
```