Logic Programming

1. Write a rule for car of list. For example, (car (1 2 3 4) ?x) would have ?x bound to 1.

2. Write a rule for cdr of list. For example, (cdr (1 2 3) ?y) would have ?y bound to (2 3).

3. Define a rule for member, so that (member 4 (1 2 3 4 5)) would be satisfied, and (member 3 (4 5 6)) would not, and (member 3 (1 2 (3 4) 5)) would not.

4. Define a rule for deep-member, so that (deep-member 3 (1 2 (3 4) 5)) would be satisfied as well.

5. Define a rule for reverse, so that (reverse (1 2 3) (3 2 1)) would be satisfied. Assume you have a rule for append

6. Write the rule interleave so that (interleave (1 2 3) (a b c d) ?what) would bind ?what to (1 a 2 b 3 c d).
Final Review

1. Write a higher-order function, all-pairs, that takes two lists and return a list of all possible pairs of elements from the two argument lists. DO NOT use recursion. For example:

```lisp
> (all-pairs '(1 2) '(2 3 4 5))
((1 2) (1 3) (1 4) (1 5) (2 2) (2 3) (2 4) (2 5))
```

2. Define the stream of even powers of 2, without defining procedures (no lambdas). You may use the stream ones as a building block.

3. Write a procedure, (smallest-containing-tree tree x y) that takes in a general tree and two elements x and y, and returns the smallest subtree of tree containing both x and y. If tree does not contain x and y, return false. You can use tree-member?.

4. Draw the environment diagram and figure out what the last expression returns

```lisp
(define p
 (let ((x #f))
   (set! x 1)
   (lambda (n)
     (lambda ()
       (set! n (+ n 1))
       (set! x (+ x n))
       x))))
(define m 3)
(define p1 (p m))
(define p2 (p m))
(p1)
(p2)
(p1)
```
5. Analyze the following code and write down all possible answers:

```scheme
(define x 10)
(parallel-execute
  (lambda() (set! x (+ 5 x)) (set! x (* x 3)))
  (lambda() (if (> x 16)
    (set! x 100)
    (set! x (- x 20))))))
```

6. Write a procedure, (num-sum exp) that takes in a valid Scheme expression, and returns the sum of all numbers that occurs in that expression. For example,

```scheme
(num-sum (if (= 2 3) (lambda(x) (+ x 3)) 10)) ==> 18
```

7. Define a procedure (trimmed ls) that takes in a list and returns the same list without the first and last element. So,

```scheme
(trimmed (1 2 3 4 5 6)) ==> (2 3 4 5)
(trimmed ()) ==> ()
(trimmed (1)) ==> ()
(trimmed (1 2)) ==> ()
```
8. Define a procedure (coolize ls) that does this:

STk> (coolize (1 2 3 4 5))
(1 (2 (3) 4) 5)
STk> (coolize (a b c d e f))
(a (b (c () d) e) f)

You may assume you are given a procedure (last ls) that returns the last element of the list. If you want even more practice, define this procedure. You can also use trimmed.