CSF
Hwk06.5
Re-Curse You Red Baron

Your mission, should you choose to accept it, is to implement these recursive functions in LISP. Your head may end up hurting. Don't worry – it won't be permanent.

First... you must learn LISP!

Basics

Starting clisp: From the ADA terminal, type: clisp
Exiting clisp: (quit)
To get out of the break level: Ctrl-D

To load a lisp text file: (load 'filename.lisp)

Comments: The comment symbol in lisp is: ;
Anything on a line that follows a semicolon will be ignored.

Useful: At the top of your file, define a function that allows you to quickly load in your text file: For example:
(defun l () (load 'lab65.lisp))
Now, you can just use (l) to load in the file.

Defining Functions

```
(defun <function-name> (<var> <var> …)
<code>
 )
```

Manipulating Lists

```
Breaking: (first '(a b c)) --> a      Also: (car '(a b c)) --> a
          (rest '(a b c)) --> (b c)     Also:  (cdr '(a b c)) --> (b c)
Making:
          (append '(a b) '(c d)) --> (a b c d)
          (cons 'a '(b c d)) --> (a b c d)
          (list 'a 'b) --> (a b)
```

Predicates

(not NIL) --> t, (not t) --> NIL
(equal '(a) '(a)) --> t, (equal '(a) '(b)) --> NIL, (equal 3 (+ 2 1)) --> t

(null '()) --> t, (null '(a)) --> NIL
(atom 'a) --> t, (atom '(a)) --> NIL
(listp 'a) --> NIL, (listp '(a)) --> t

Conditional

```
(cond (<test> <result>)
      (<test1> <result1>)
      ....)  --> returns first result where test is true.
(cond (t 3)) --> 3, (cond  ((equal 2 3) 4) ((equal 5 5) 6)) --> 6
```

Basic math operation

```
(+ 4 5) --> 9
(- 10 3) --> 7
(* 5 6) --> 30
(/ 4.5 2.3) --> 1.9565217
(mod 11 3) --> 2
```

Variables

In LISP, it is often the case that you can get by with few or no variables (other than parameters). This is because everything returns a value and you can use functional composition. If you want to make local variables, you can use the let mechanism. This will create a number of local variables and also create a scope in which they are visible and alive. Only use variables if needed (which is rarely).

```
(let (x y)
  <code>
)
```

If you want to set a local variable, use the setq function
(setq a 3) --> Sets a to 3 and returns 3

Misc

(print lst) --> prints out the list lst (useful for debugging).

(trace <function>)   : Turns on tracing of the given function. This shows when you go in and out of a function.
(untrace <function>) : Turns tracing off for this function.

THE LAB

The bunny slope:
Write a function countelements(lst) which takes a list and counts the number of elements that is has.
Ex:
(countelements '(a b c d (e f) g)) --> 6
(countelements '()) --> 0

The green circle:
Write a function myreverse(lst) which takes a list and returns the list lst reversed at the top level.
Ex:
(myreverse '(a b a b a a c d a)) --> (A D C A A B A B A)
(myreverse '(a (b a) b a (a) c (d e) a)) --> (A (D E) C (A) A B (B A) A)
(myreverse '(the cat)) --> (CAT THE)

The blue square:
Write a function myremove(atm lst) which takes an atom and a list and returns the list lst with all elements equal to atm removed at the top level.

Ex:
(myremove 'a '(a b a b a a c d a)) --> (B B C D)
(myremove 'a '(a (b a) b a (a) c (d e) a)) --> ((B A) B (A) C (D E))
(myremove 'the '()) --> ()

The black diamond:
Write a function reccalc(lst) which acts as a prefix notation calculator with integers and + - * and /.
Ex:
(reccalc '(+ (/ (+ 2 8) 2) (- 10 4))) --> 11
Note: Each operator takes exactly 2 arguments, e.g. you can't have (+ 2 3 4).

The mega-challenge:
Write a function megareverse(lst) which takes a list lst and REALLY reverses it.
Ex:
(megareverse '(the (cat (in the hat)) went for (a walk)))
--> ((walk a) for went ((hat the in) cat) the)

The double dare-ya mega-challenge:
Write a function megaremove(atm lst) which takes an atom and a list and returns the list lst with all atoms equal to atm removed at ALL levels.
Ex:
(megaremove 'a (a b a b a a c d a)) --> (B B C D)
(megaremove 'a '(a (b a) b (a (b a b) c) (a) c (d e))) --> ((B) B ((B B) C) NIL C (D E))

Extra Credit I:
The super double dare-ya mega-challenge with cherry on top:
Write a function sandr(targetlst replst) which takes a target list targetlst and a list replst that contains pairs of search and replace atoms. It should return targetlst with ALL items appropriately replaced. You may want to use a helper function(s) if you find it useful.
Ex:

```
(sandr '(a (b a) b (a  (b a b) c) (a) c (d e)) ‘((a x) (c y)))
        --> (x (b x) b (x  (b x b) y) (x) y (d e))
```

Extra Credit II:
The Victory Lap:
Write a function flatten(lst) which takes a list and basically removes all inner parentheses.
Ex:

```
(flatten '(the (cat (in the hat)) went for (a walk) )
        --> (the cat in the hat went for a walk)
```

The super double dare-ya mega-challenge with cherry on top:
Write a function sandr(targetlst replst) which takes a target list targetlst and a list replst that contains pairs of search and