Homework 5
Seth Denney

1.Write a Scheme function named outside so that outside(I,J,K) returns true iff K is an integer K< I or K> J .

```
(define (outside I J K) (or (< K I) (> K J)))
```

2.Write a Scheme function named max so that max(L,M) returns true if M is the largest element in the list L.

```
(define (max1 L M) (and (memq M L) (max2 L M)))
(define (max2 L M) (or (null? L) (and (>= M (car L)) (max2 (cdr L) M))))
```

3.Write a Scheme function p such that p(X) returns true if X is a list consisting of n a's followed by n+1 b's, for any n >= 1.

```
(define (p X) (aList X 0))
(define (aList X NA1) (or
                        (and
                         (equal? (car X) #\a)
                         (and
                          (equal? (car (cdr X)) #\a)
                          (aList (cdr X) (- NA1 1))))
                        (and
                         (equal? (car X) #\a)
                         (and
                          (equal? (car (cdr X)) #\b)
                          (bList (cdr X) (- NA1 1))))))
(define (bList X NB1) (or
                        (and
                         (equal? (car X) #\b)
                         (and
                          (equal? (length X) 1)
                          (equal? NB1 0)))
                        (and
                         (equal? (car X) #\b)
                         (and
                          (not (null? (cdr X)))
                          (and
                           (equal? (car (cdr X)) #\b)
                           (bList (cdr X) (+ NB1 1)))))))
```

4.Consider the following grammer:

expression -> a | b | (list)
list -> list ; expression | expression

Define a Scheme function parse(L) which will return true iff L is a list of tokens representing a legal expression. Since semicolon and parenthesis will give a problem, it's OK to make them character constants. Rewrite the grammar if the left recursion gives you a problem.

parse('(#\(, #\(, a ,#\; , b , #\), #\; , a , #\)))
#t

```
(define (parse X) (step X 0 #f #t))
(define (AorB X) (or
                (equal? X #\a)
                (equal? X #\b)))
(define (step X NP1 SC AB) (or
                        (and
                         (null? (cdr X))
                         (and
                          (equal? (car X) #\))
                          (equal? NP1 1)))
                        (or
                         (and
                          (not (not AB))
                          (and
                           (null? (cdr X))
                           (AorB (car X))))
                        (or
                          (and
                           (not (not AB))
                           (and
                            (AorB (car X))
                            (step (cdr X) NP1 #t #f)))
                          (or
                          (and
                            (not (not SC))
                            (and
                             (equal? (car X) #\;)
                             (and
                              (> NP1 0)
                              (and
                               (> (length (cdr X)) NP1)
                               (step (cdr X) NP1 #f #t)))))
                           (or
                            (and
                             (>= NP1 0)
                             (and
                              (equal? (car X) #\()
```

```
                (and
                 (> (length (cdr X)) (+ NP1 1))
                 (step (cdr X) (+ NP1 1) #f #t))))
              (and
               (> NP1 0)
               (and
                (equal? (car X) #\))
                (and
                 (> (length (cdr X)) (- NP1 1))
                 (step (cdr X) (- NP1 1) #t #f)))))))))
```

5.Write a Scheme function that is tail recursive, and returns the last element of a list.

(define (last L) (if (null? (cdr L)) (car L) (last (cdr L))))

6.Write a Scheme function remove(n,x) that is tail recursive and returns a list with the n-th element of the list (1 <= n <= list-length) removed.

remove(4, ‘(1,2,5,3,4,6)) => (1,2,5,4,6)

```
(define (remove N L) (if (or (> N (length L)) (< N 1))
                 L
                 (if (equal? N 1)
                    (cdr L)
                    (append (list-tail (reverse L) (- (length L) 1)) (remove (- N 1) (cdr L))))))
```

7.Write a Scheme function that will add up the values of a list of numbers after applying the first parameter function to the list.

addem(f, x)

so for example if f is the function (abs x) ;built in absolute value

addem(abs ‘(-3,4,-22)) => 29

```
(define (addem F X) (sum (map F X) 0))
(define (sum L S) (if (null? (cdr L)) (+ S (car L)) (sum (cdr L) (+ S (car L)))))
```