Common Lisp functions   First Draft 2/17/09 COSC 341

Numeric functions
```
*, +, - /          ; takes 1 to n arguments. returns arithmetic result
(mod x y)          ; 2 args. Returns x%y
1+ x               ; returns x++
1- x               ; returns x--

sqrt               ; returns x^(1/2)

(expt x y)         ; 2 arguments, returns x^y

min                ; 1 to n arguments
max                ; 1 to n arguments

abs                ; 1 argument
round              ; 1 argument

floor              ; 1 arg
ceiling            ; 1 arg

sin                ; 1 arg. x in radians
cos                ; 1 arg.
tan                ; 1 arg.
```

List functions
```
car                ; returns first element of list
cdr                ; returns list without first element
cadr, ... cdddr
first, second, ... tenth
last
(nth num l)        ; 2 args. Returns num-th element in list l
                   ; (initial element is zeroth)
(cons e l)         ;return list with e as first element and l as cdr
(append l1 l2)     ;return list whose elements are the
                   ; elements of l1 and l2
(list a0 a1 ... an); return list with elements a0, a1, ... an

(length l)         ; returns number of elements in l
(reverse l)        ; returns list with elements reversed
```

Testing Functions (Predicates)
```
and, or            ; short circuits
not

atom
null
numberp
symbolp
listp
endp               ;test for end of list
(typep x y)        ;is x the type of y?
eq                 ;compare objects
eql                ;numbers and characters are eql to themselves,
equal              ;compare structures
member x l);       ;if x is a member of l (using eql),
                   ; returns tail portion of l starting at x
```

```
>, >=, =, <= <
zerop
minusp
plusp
evenp
oddp
```

Flow of control

```
(cond
      (test1 action1)
      (test2 action2)
      )                       ;first test that is true causes
                              ; corresponding action to execute

(do
      ((var1 init1 update1)   ; local variables var1, var2
       (var2 init2 update2)
      )
   (s-expression)             ; test for continuation
  )

 ;; example of do
 (do
    ((x '(1 2 3) (cdr x))     ; local x, initialize and iterate
     (sum 0 (1+ sum)))        ; local sum, initialize and iterate
    ((null x) sum))           ; test for completion and return vaue


(prog
      ((var1 init1)           ; local variables
       (var2 init2)
       (var3 init3)
       )
  label                       ; optional target of go
  s-expression1
  s-expression2
  (go label)                  ; loop back to label

;; example of prog
 (prog
    ((sum 0)                  ; local var sum
     (l '(a b c)))            ; local var l
    again                     ; target of go
    (cond ((atom l)           ;test for completion,
           (return sum)))     ; if done return sum
    (setq sum (1+ sum))       ; body of loop
    (setq l (cdr l))          ; more body
    (go again))               ;  a goto statement!


mapcar            ; takes n arguments. First is a function,
                  ; remaining arguments are lists of arguments
                  ;  to that function
                  ; apply function to arguments taken from lists and
                  ; returns list of resulting values
;; example of mapcar
```

```
(mapcar '1+ '(3 5 7))    --> (4 6 8)
(mapcar '+ '(1 1 1 1) '(4 5 6 7) '(-2 -2 -2 -2))    --->  (3 4 5 6)
(mapcar 'atom '(a b (x y) nil (a b) x y)) ---> (T T NIL T NIL T T)
```

Evaluation control

```
apply                   ; 2 arguments. First is a function,
                        ; second is a list of actual arguments to
                        ;  that function
;; example of apply
(apply 'cons '(a (b c)))      ---> (a b c)
(apply 'cdr '(a b c))   ---> error because actually doing (cdr a b c)
(apply 'cdr '((a b c)))       ---> (b c)
(apply '+ '(1 2))             ---> 3

funcall           ; first argument is a function, remaining arguments
                  ; are arguments that are passed
                  ; to that function.  I.e., like apply, except that
                  ; arguments passed to function
                  ; are not in a list
;; example of funcall
(funcall 'cdr '(b c))   --> (c)
(funcall '+ 1 2 3)      --> 6

quote             ; return argument without evaluation of it
      (quote a)  ---> a

eval              ; evaluates argument, then result is evaluated
                  ; (two evaluations all together)
      (eval (+ 1 2 3))  ---> 6
;; example of eval
(cons '+ '(2 3))  --> (+ 2 3)
(eval (cons '+ '(2 3)))       ---> 5
```