CONDITIONAL EXPRESSIONS

cond

Cond is the most general conditional provided by Lisp.

```
(cond
        ( <test1>          <consequents 1> )
        ( <test2>          <consequents 2> )
                  •
                  •
                  •
        ( <testm>          <consequents m> )
)
```

-- Each test is a predicate that returns NIL (false) or nonNIL (any nonNIL value acts like true).

-- Each consequent is zero or more Lisp forms (usually one).

-- The tests are tried sequentially. As soon as one succeeds, all of its consequent forms are evaluated, and the value of the last consequent form is returned by the cond.

-- If all the tests yield NIL, the cond returns NIL.

```
( cond
        ( ( plusp X ) 1 )
        ( ( minusp X ) -1 )
        ( ( zerop X ) 0 )
)


(cond
        ( ( atom X )  X )
        ( T       ( SETF  FLAG  T )  (car  X ) )
)


( cond
        ( ( null   L )   NIL )
        ( ( atom L )  ( f  L ) )
        ( T    (or   (f  (car L ) )  (f   ( cdr  L ) ) ) )
)


( cond
        ( ( match  X  ’noun ) (nprocess  X ) )
        ( ( match  X  ’verb  ) (vprocess  X ) )
        ( ( match  X  ’adjective ) ( adjprocess X ) )
        ( ( match  X  ’adverb ) ( advprocess X ) )
        ( ( match  X  ’article ) ( artprocess X ) )
        ( ( match  X  ’preposition ) ( pprocess X ) )
        ( T                       ( errormsg  X ) )
)
```

if

The Lisp if statement provides an if - then - else facility.

```
( if    <test>    <thenform>    <elseform> )
```

The if returns the value of <thenform> if the test is non NIL, otherwise <elseform>.

```
( if    ( >  X  10 )    ( -  X  10 )    0 )
```

when, unless

These forms may be used instead of if when either the <thenform> or the <elseform> is NIL.

```
( when    <test>    <thenform> )
          else NIL is understood
( unless    <test>    <elseform> )
          then  NIL  is  understood
```

There's also a case form.

CONDITIONALS IN FUNCTIONS

```
( defun  small  ( v )
      ( if    ( and
                  ( >  v  -10 )
                  ( <  v   10  ) )
                     'small     'large ) )



( defun  signum  ( X )
      ( cond
                ( ( plusp  X )   1 )
                ( ( minusp  X )           -1 )
                ( ( zerop  X )   0 ) ) )
```

COMMON LISP EQUALITY

EQ

EQ is for **symbol** equality.

(EQ X Y) returns

T if X and Y are identical symbols (are represented by the same chunk of computer memory).

NIL otherwise

```
( eq    'a          'a )            T

( eq    'b          'a)             NIL

( eq    '( a b )  '( a b ) )        NIL

( setf  x           'a )

( eq    x           'a )            T

( eq    ( float 2 ) ( sqrt 4 ) )    NIL
```

EQL

EQL is for testing numeric equality.

(EQL x y) returns

T if (EQ x y) or

if x and y are numbers of the same type and value.

NIL otherwise

```
( eql   ( float 2 )  ( sqrt 4 ) )    T
( eql   2            ( sqrt  4 ) )   NIL
( eql   '( a b )  '( a b ) )         NIL
```

EQUAL

EQUAL is the most general equality test in Lisp

(equal x y) returns

T	if	(EQL x y) or
	if	x and y are S-expressions whose components are equal.
NIL	otherwise.	

(equal '(a b) '(a b))	T
(equal '(a b) (list 'a 'b))	T
(equal '(a b) (cons 'a (cons 'b NIL)))	T

```
defun  equal  ( x y )
  ( cond
     ( ( atom x ) ( cond ( ( atom y ) ( eql x y ) )
                         ( T NIL ) ) )
     ( ( atom y ) NIL )
     ( ( equal ( car x ) ( car y ) )
               ( equal ( cdr x ) ( cdr y ) ) )
     ( T NIL ) ) )
```

```
( equal  7  7 )
( equal  7  8 )
( equal  7  '( a b ) )
( equal  '( a b )  7 )
( equal  '( a b )  '( a b ) )
( equal  ( cons 'a 'b ) ( cons 'a 'c ) )
( equal  '( 1 2 3 )  '( 4 5 ) )
```

List Membership

Instead of returning T or NIL, some predicates in Common Lisp return NIL for false and any nonNIL value for true.

member

Member determines if a value is an element of a list.

(member X L) returns

NIL	if	X is not a top-level element of list L
L’	if	X is a top-level element of L and L’ is the sublist starting with X and having all the remaining elements of L.

(member ’a ’(b c d a e))
returns (A E) .

MEMBER normally tests with EQL

To test with EQ or EQUAL or something else, use the keyword :TEST.

(member val list :test #’equal)

determines if val is in list using equal.

(member ’(a b) ’((e e) (a b) (c d))

:test #’equal)

returns

((A B) (C D))

More Utilities

length

(length L)	returns a count of the number of top-level elements of L.

(length	’(a b c))	3
(length	’((a b) (c d)))	2

reverse

(reverse L)	returns a list with the same elements as L, but in reverse order.

(reverse ’(a b c))	(C B A)
(reverse ’((a b) (c d)))	((C D) (A B))

TESTING FOR NIL

null

Null is the predicate that tests for NIL.

```
        ( null  x )       returns
            T       if   x   is   NIL
           NIL      if   x   is   nonNIL
```

endp

Endp does the same thing but sounds better when testing for end of a list.

```
( cond
   ( ( endp  L )        0 )
   (   T       ( 1+     ( length  ( cdr  L ) ) ) ) )
```

I/O

- PRINTING TO THE SCREEN

  (PRINT   S-exp)

- READING FROM THE KEYBOARD

  (READ)

```
( PRINT   ’(Please enter N ) )
( setf          N     (READ) )
( PRINT    ’(Now enter list L ) )
( setf          L     (READ) )
```

FILE I/O

```
( with-open-file
(  < stream name >
   < file specs >
  :direction       < :input  or   :output > )
   < sequence of forms >  )
```

READING

```
( with-open-file    ( fi    ”myinput.lsp”
     :direction    :input  )
     ( setf val   ( read fi ) )     )
```

WRITING

```
( with-open-file   ( fo    ”myoutput.lsp”
      :direction    :output )
   ( print   val    fo )       )
```