LISP

- **List Processing language**

- One of the oldest of computer languages.

- Close to the functional paradigm

- Pure LISP

- Common Lisp.

- XLISP
Getting Started 1.

maxwell:JPEH:26 pwd
/macslocal/xlisp21gbc
maxwell:JPEH:27 xisp
XLISP-PLUS version 2.1g
Portions Copyright (c) 1988, by David Betz.
Modified by Thomas Almy and others.
UNIX version
> (+ 2 3)
5
> (list 1 2 3 4 5)
(1 2 3 4 5)
> (car (list 1 2 3 4 5))
1
> (cons 1 (list 2 3 4 5))
(1 2 3 4 5)
> (append (list 1 2 3 4) (list 5 6 7))
(1 2 3 4 5 6 7)
> (exit)
maxwell:JPEH:28
Getting Started 2

- Add `/macslocal/xlisp21gbc` to your **PATH**. Do this by editing your `.login` file.

- Learn a Unix editor – moving Word files to Unix is a bad idea, in part because word doesn’t understand code.
S-expressions

LISP programs manipulate S-expressions.

An S-expression is either:

• an atom or,

• a dotted pair \((S_1.S_2)\) where \(S_1\) and \(S_2\) are S-expressions

Atoms are sequences of characters eg: \(x, f, foo, +, \geq\)

There are two special atoms \(t\) and \(nil\).
Lists

A list is the atom \textbf{nil} (the empty list) or a dotted pair \((S_1.S_2)\).

Lists of the form

\((S_1.(S_2.(S_3\ldots(S_n . \textbf{nil}) \ldots))))\)

are called \textit{linear lists} and are written as

\[(S_1 \ S_2 \ S_3 \ \ldots \ S_n)\]
Variables and the Environment

LISP atoms can be *bound* to S-expressions in the *environment*. These atoms are called variables.

This is done using

\[(\text{setq } x \ S)\]

If \(S\) is an S-expression representing a LISP function then

\[(S \ S_1 \ \ldots \ S_n)\]

is normally evaluated by

1. First evaluate \(S_1, \ldots, S_n\) (in that order)
2. Then *apply* \(S\) to the resulting values.
Logical Terms

- ATOM

- NULL

- NUMBERP

- ZEROP

- <

- =, EQUAL

- AND, OR, NOT
List Constructors

- CAR
- CDR
- CONS

\[(\text{member } 3\ (\text{list } 1\ 2\ 3\ 4\ 5))\]
\[(3\ 4\ 5)\]
\[>\]

- \((\text{eq } S1\ S2)\) evaluates to \(T\) if
  1. \(S1\) and \(S2\) evaluate the the same atom, or
  2. \(S1\) and \(S2\) evaluate to pointers to the same list cell
Conditionals

(COND (test1 value1) (test2 value2) .... (testn valuen))

(IF form1 form2 form3)
Lambda Functions

(LAMBDA var-list term)

((LAMBDA (X Y) (+ X Y)) 1 9 )

function arguments

This is perhaps the MOST important diagram in the course.
Defining Functions

(DEFUN function-name variable-list term)

>(defun plus (X Y) (+ X Y))
PLUS
>(plus 4 5)
9
Loading code

- This is not part of pure LISP but it is helpful to know how to do this.

  (load "file.lsp")

- Comments are lines beginning with ‘;’

- To quit

  (exit)