Introduction to LISP II.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

LISP language

LISP: LISt Processing language
• An AI language developed in 1958 (J. McCarthy at MIT)
• Special focus on symbolic processing and symbol manipulation
  – Linked list structures
  – Also programs, functions are represented as lists
LISP tutorial: data types

Basic data types:
- **Symbols**
  - a
  - john
  - 34
- **Lists**
  - ()
  - (a)
  - (a john 34)
  - (lambda (arg) (* arg arg))

LISP tutorial

Lists represent function calls as well as basic data structures

```lisp
> (factorial 3)
6
> (+ 2 4)
6
```

```lisp
> (setq a '(john peter 34))
(john peter 34)
> (setq a `((john 1) (peter 2)))
((john 1) (peter 2))
```
### LISP tutorial: lists

**List representation:**
- A singly linked list

![Diagram of a singly linked list]

```lisp
> (setq a '(john peter))
  (john peter)
> (car a)
  john
> (cdr a)
  (peter)
```

### LISP tutorial

**Useful list functions:**

```lisp
> (reverse '(1 2 3)) ;reverse the elements of a list
  (3 2 1)
> (member 'a '(b d a c)) ;set membership -- returns the first tail
  (A C) ;whose car is the desired element
> (find 'a '(b d a c)) ;another way to do set membership
  A
> (find '(a b) '((a d) (a d e) (a b d e) ())) :test #'subsetp
  (A B D E) ;find is more flexible though
> (subsetp '(a b) '(a d e)) ;set containment
  NIL
```
LISP tutorial: equality

4 equality predicates: =, equal, eq, eql

> (= 2 4/2) ;; used for numerical values only
  T
> (setq a '(1 2 3 4))
  (1 2 3 4)
>(setq b '(1 2 3 4))
  (1 2 3 4)
>(setq c b)
  (1 2 3 4)
> (equal a b) ;; equal is true if the two objects are isomorphic
  T
> (equal c b)
  T

LISP tutorial: equalities

>(eq a b) ;; eq is true if the two arguments point to the same object
  NIL
>(eq b c)
  T
LISP tutorial: nil

Nil represents False and an empty list

> (null nil)
  T
> (null ( ))
  T
> (null ‘(a b))
  NIL
> (not ‘(a b))
  NIL

LISP tutorial: functions

Logical operators: and, or

> (and NIL T)
  NIL
> (and T 2 3)
  3
> (or nil (= 5 4))
  NIL
> (or nil 5)
  5
LISP tutorial: recursion

Recursive function definitions are typical for LISP

> (defun factorial (num)
   (cond ((<= num 0) 1)
       (t (* (factorial (- num 1)) num)))
   )

FACTORIAL
> (factorial 4)
24

LISP tutorial: local and global variables

> (setq a 12)
12
> (defun foo (n)
   (setq a 14)
   (+ n 2))

FOO
> a
12
> (foo 3)
5
> a
14
LISP tutorial: local variables

Defining local variables with let
> (setq a 7) ;store a number as the value of a symbol
  7
> a ;take the value of a symbol
  7
> (let ((a 1)) a) ;bind the value of a symbol temporarily to 6
  1
> a ;the value is 7 again once the let is finished
  7
> b ;try to take the value of a symbol which has no value
Error: Attempt to take the value of the unbound symbol B

LISP tutorial: local variables

Defining local variables with let and let*
> (let ((a 5) ;; binds vars to values locally (in parallel)
            (b 4))
        (+ a b))
  9
> (let* ((a 5) ;; binds vars sequentially
            (b (+ a 2))
        (+ a b))
  12
LISP tutorial: functions revisited

Standard function – all parameters defined

(defun fact (x)
  (if (> x 0)
    (* x (fact (- x 1)))
    1))

But it is possible to define functions:
• with variable number of parameters,
• optional parameters and
• keyword-based parameters

LISP tutorial: functions revisited

Functions with optional parameters
> (defun bar (x &optional y) (if y x 0))
BAR
> (defun baaz (&optional (x 3) (z 10)) (+ x z))
BAAZ
> (bar 5)
0
> (bar 5 t)
5
> (baaz)
13
> (baaz 5 6)
11
> (baaz 5)
15
LISP tutorial: functions revisited

Functions with variable number of parameters

> (defun foo (x &rest y) y) ;; all but the first parameters are put ;; into a list

FOO
> (foo 3)
NIL
> (foo 1 2 3)
(2 3)
> (foo 1 2 3 4 5)
(2 3 4 5)

LISP tutorial: functions revisited

Functions with ‘keyword’ parameters

> (defun foo (&key x y) (cons x y))

FOO
> (foo :x 5 :y '(3))
(5 3)
> (foo :y '(3) :x 5)
(5 3)
> (foo :y 3)
(NIL 3)
> (foo)
(NIL)
LISP tutorial: arrays

List is a basic structure; but arrays and structures are supported

> (setf a (make-array '(3 2)) ;; make a 3 by 2 array
#2a((NIL NIL) (NIL NIL) (NIL NIL))
> (aref a 1 1)
NIL
> (setf (aref a 1 1) 2)
2
> (aref a 1 1)
2

LISP tutorial: structures

> (defstruct weather
  temperature
  rain
  pressure)
WEATHER
> (setf a (make-weather)) ;; make a structure
#s(WEATHER :TEMPERATURE NIL :RAIN NIL :PRESSURE NIL)
> (setf a (make-weather :temperature 35))
#s(WEATHER :TEMPERATURE 35 :RAIN NIL :PRESSURE NIL)
> (weather-temperature a) ;; access a field
35
> (weather-rain a) NIL
> (setf (weather-rain a) T) ;; set the value of a field
T
> (weather-rain a) T
LISP tutorial: iterations

Many ways to define iterations

Commands:
• loop
• dolist
• dotimes
• do, do*

Also we can write compactly the code for repeated application of function to elements of the list:
• mapc, mapcar

LISP tutorial: iterations

Iterations: loop

> (setq a 4)
4
> (loop (setq a (+ a 1))
   (when (> a 7) (return a))) ;; return exists the loop
8
> (loop (setq a (- a 1))
   (when (< a 3) (return)))
NIL
LISP tutorial: iterations

Iterations: dolist

> (dolist (x '(1 2 3 4)) (print x))
1
2
3
4
NIL ;; NIL is returned by dolist
>

LISP tutorial: iterations

Iterations: dotimes

> (dotimes (i 4) (print i)) ;; starts from 0 and continues till limit 4
0
1
2
3
4
NIL ;; returns NIL
LISP tutorial: iterations

Iterations: do

> (do ((x 1 (+ x 1)) ;; variable, initial value, next cycle update
    (y 1 (* y 2))) ;; the same
   ((> x 5) y) ;; end condition, value do returns
   (print (list x y)) ;; body of do – a sequence of operations
   (print ‘next))

(1 1)
NEXT
(2 2)
NEXT
(3 4)
NEXT
(4 8)
NEXT
(5 16)
NEXT
32

LISP tutorial: iterations

Iterations: do *

> (do* ((x 1 (+ x 1)) ;; variable, initial value, next cycle update
    (y 1 (* x 2))) ;; <<< --- update based on x
   ((> x 5) y) ;; end condition, value do returns
   (print (list x y)) ;; body of do – a sequence of operations
   (print ‘next))

(1 1)
NEXT
(2 4)
NEXT
(3 6)
NEXT
(4 8)
NEXT
(5 10)
NEXT
12
LISP tutorial: mapcar

Repeated application of a function to elements of the list

> (mapcar #'oddp '(1 2 3 4 5)) ;; named function
(T NIL T NIL T)
> (mapcar #'(lambda(x) (* x x)) '(1 2 3 4 5)) ;; temp function
(1 4 9 16 25)

LISP tutorial

Evals and function calls

• A piece of code can be built, manipulated as data
• What if we want to execute it?
> (setq b ‘(+ a 4))
(+ a 4)
> (eval b) ;; explicit evaluation call
16
> (funcall #’+ 2 4) ;; calls a function with args
6
> (apply #’+ 2 ‘(5 6)) ;; calls a function with args
(last args as a list)
13
LISP tutorial: input/output

You can input/output data to:
• standard input/output,
• string or
• file

A number of functions supported by the Lisp:
• (read) ;; reads the input from the standard input
• (print ‘a) ;; prints to the standard output
• (scanf …) (printf …) (format …) for formatted input and output
• (open ..) (close ..) for opening and closing the files

LISP tutorial: program calls

Assume you have your lisp code ready in the .lisp file
… and this is how you load it
(load "~/private/lisp/file-to-load.lisp")
… and you can call another load from it as well
LISP on CS machines

**Common Lisp for Linux** is available in

- `/usr/local/contrib/cmucl-19d/` directory

The bin directory with the lisp executable is:

- `/usr/local/contrib/cmucl-19d/bin/`

- Other subdirectories include manual, docs and libraries
- Please add them to your active path
- Type lisp to start the program

Other option to run lisp is to do it from emacs editor. There is a lisp interpreter window you get once you start the editor.