Programming in Lisp

Lecture #2
Kenneth W. Flynn
RPI CS

Outline
- Items from last time
- Recursion, briefly
- How to run Lisp
- I/O, Variables and other miscellany
- Lists
- Arrays
- Other data structures

Items From Last Time I
- (cdr nil)
  ▶ Returns nil
- Unlimited arguments?
  ▶ &rest
  ▶ We'll talk about this next week

Items From Last Time II – (listp)
- (> (listp 'a) NIL)
- (> (listp 1) NIL)
- (> (listp '(1 2 3)) T)
- (> (listp ()) T)
- (> (listp nil) T)
- (> (listp '(HELLO)) T)
- (> (listp (5)) ; Error:

Items From Last Time III – '
- The following characters cannot appear in symbols:
  ▶ () 
  ▶ ;
  ▶ #
- So, ’*hi*’ is valid:
- > (listp ’*hi*) NIL
- > ['*hi*]
Recursion, Briefly

- Frequently we'll write recursive Lisp functions
- Recursive functions should have
  - Basis case (at least one)
  - Recursive case
- Don't forget terminating condition

How To Run Lisp

- Under UNIX
  - `kd, gd`
  - Specify in homework which used
  - `:q` if you make an error
  - `^D` to exit
- Under Win '95
  - Select Allegro CL Lite for Windows

How To Write Lisp

- Use a text editor with paren matching!
  - `vi`
    - `:set sm`
  - `emacs`
  - Others?
- Load code into Lisp and then try it...

Output With format

- Output is done with the format command
- `(format destination format-string args...)`
- Destination is "t" for the console
- Format string is similar to C's printf function
- Usually returns nil, but we don't care!

format Examples I

- `(format t "~%Hello World.~%")`
  - Hello World.
  - NIL
- `(format t "~A.~%" (+ 2 2))`
  - Two plus two is 4.
  - NIL
Words fail, buildings tumble.
The ground opens wide.

Incredibly powerful" says the text.
Reads input and parses into Lisp objects
Reads up to a newline, puts input into string
Preferred for reading from console

Used to create a "block"
Allows side-effects
Value of last expression evaluated is returned
Avoid if possible; frequently needed for debugging output, etc.

For numbers, you have (= args...)
For others you have: eq, eql, equal, equalp
eq: Implementationly identical (rarely used)
eql: Logically identical (what we were thinking)
equal: Object identical (lists)
Stick with equal (more info on Steele 103-110)
Equality Examples

> (equal '(1 2 3) '(1 2 3))
  T
> (eq '(1 2 3) '(1 2 3))
  NIL
> (equal "Hello" "Hello")
  T
> (= 1 1.0)
  T

setf

- Assigns value to variable
- Side-effect

> (setf x '(1 2 3))
  (1 2 3)
> x
  (1 2 3)

let

- Introduces new local variables
- Form ((let Variable-Bindings-List Expressions*)
  Variable-Bindings-List is a list of pairs of variables
  and expressions to set them equal to. These are
  your new local variables
  Implicit prog

let Example

> (let ((x "Hello")
  (y "World")
  (format t "~%~A~A~%" x y)
  (Hello World)
  NIL

Lists

- Lisp lists
- List construction functions
- Access (review)
- Mapping functions
- Sets, Sequences
- Dotted lists

Lisp Lists I

- cons
- car
- cdr
- nil

Kenneth W. Flynn (19-24)
09/09/98
Lisp Lists II

- A "cons" refers to a pair of pointers
- The first pointer may point to data or another cons
- The second may point to data, another cons, or nil
- cons is used to construct such a pair
- car refers to the first pointer
- cdr refers to the second pointer

List Construction Functions

- copy-list literally copies a list
  - (copy-list list)
- append copies the list arguments onto the beginning of the last list argument
  - (append list1 list2 list3)
  - list1 -> list2 -> list3
- Don't forget list and cons

Access (Review)

- car and cdr (first and rest), first, second, third...
- nth returns nth car in the list
  - >\(\text{nth} \ 2 \ '(1 \ 2 \ 3)\)
  - 3
- nthcdr returns the nth cdr in the list (confused?)
  - >\(\text{nthcdr} \ 2 \ '(1 \ 2 \ 3 \ 4)\)
  - (3 4)
- last returns the last cons in the list

Mapping Functions

- All about mapcar
- mapcar is used to apply a function to each element in one or more lists
- mapcar's first argument is a function
- One by one, the nth arguments of each list are passed to the function

Function Passing: #'

- #' Sharp Quote
- All functions can be passed as parameters
  - #'+
  - #'-
  - #'list
  - #'my-function
- Used in many standard functions
- Generics...

Ilambda Functions

(A rose without a name...)

- Sometimes you create a function just to pass it to something like mapcar
- Instead of naming the function, you can create a function with no name -- a lambda function
- Simply use the special symbol lambda instead of the function name
- #\((\text{lambda} \ (x \ y) \ (+ \ x \ y))\) is our old friend the adder

Kenneth W. Flynn (25-30)
09/09/98
**mapcar Examples**

> (mapcar #'+ '(1 2) '(1 2))  
(2 4)

> (mapcar #'(lambda (x y)  
          (+ x y))  
     '(1 2)  
     '(1 2))  
(2 4)

**member**

- (member object list) returns a cons beginning with object if present
- member takes several keyword arguments
- Keyword arguments are of the form
  - :keyword key-value
  - :test equivalence-function
  - :key function-to-be-applied-first
- Order is irrelevant

**member Examples I**

> (member 2 '(1 2 3))  
(2 3)

> (member 3 '(1 2 3)  
   :key #'(lambda (x)  
            (+ x 1))  
   )  
(2 3)

**member Examples II**

> (member '(1 2)  
         '((2 3) (1 2)))  
NIL

> (member '(1 2)  
         '((2 3) (1 2))  
         :test #'equal)  
((1 2))

**Sequences**

- length
  - (length '(1 2 3)) returns 3
- reverse
  - (reverse '(1 2 3)) returns (3 2 1)
- sort list sort-function
  - (sort '(3 1 2) #'>) returns (3 2 1)

**Dotted Lists**

- Proper list refers to a list in which every cdr points to another cons (or nil)
- Dotted list refers to the case when this is not true
To Dot Or Not To Dot?

- Lisp displays a list with the dot only for the cons which is not proper
- So a cons can be used as a two field data structure.
- Better ways of doing this, though... Structures (next week) come to mind.

Arrays

- Creation
  - make-array
  - 1 required argument – list of dimensions or integer
  - :initial-element initializes array
- Retrieval
  - aref
  - Returns reference to element

Array Example

```lisp
> (setf x (make-array 3 :initial-element 0))
#(0 0 0)
> (setf (aref x 1) 1)
1
> (setf (aref x 2) 2)
2
> x
#(0 1 2)
```

Vectors

- Just one dimensional arrays
- Create with vector
  - Similar to list
- Can access quickly with svref instead of aref
```lisp
> (setf x (vector 1 2 3))
#(1 2 3)
> (svref x 2)
3
```

Whew!

- We’ve covered a lot today!
- For next week
  - Read Chapters 3 and 4 in Graham
  - Start Homework #1 (Due 9/14. New due date!)
- On the next exciting episode
  - Structures
  - Control Flow
  - Gory function details

Kenneth W. Flynn (37-42)
09/09/98