CS W4701
Artificial Intelligence

Fall 2013

Lisp Crash Course

Jonathan Voris

(based on slides by Sal Stolfo)

Another Quick History Lesson

- 1956: John McCarthy organizes Dartmouth AI conference
  - Wants a list processing language for AI work
  - Experiments with “Advice Talker”
- 1958: MarCarthy invents LISP
  - LISt Processor
- 1960: McCarthy publishes Lisp Design
  - “Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I”
- Implemented by Steve Russel
  - eval in machine code
- 1962: First compilers by Tim Hart and Mike Levin

Another Quick History Lesson

- Afterwards, tons of variant Lisp projects
  - Stanford LISP
  - ZetaLisp
  - Franz Lisp
  - PSL
  - MACLISP
  - NIL
  - LML
  - InterLisp
  - SpiceLisp
  - AutoLisp
  - Scheme
  - Clojure
  - Emacs Lisp

Another Quick History Lesson

- 1981: DARPA sponsors meeting regarding splintering
- Several projects teamed up to define **Common Lisp**
- Common Lisp is a loose Language specification
- Many implementations
  - Such as LispWorks
- 1986: Technical working group formed to draft ANSI Common Lisp standard
- 1994: ANSI INCITS 226-1994 (R2004)

Why Lisp?

- Freedom
  - Very powerful, easily extensible language
- Development Speed
  - Well suited for prototyping
- Politics
  - McCarthy liked it, so should you
- Symbolic
  - **Homoiconic**: code structures are the same as data structures (lists!)

The Big Idea

- Everything is an expression
- Specifically, a **Symbolic** or **S-expression**
- Nested lists combining code and/or data
- Recursively defined as:
  - An atom, or
  - A list (a . b) where a and b are s-expressions

A Note on Syntax

- You'll usually see (a b c)
- Where are the dots?
- (a b c) is a shortcut for (a . (b . (c . NIL)))

Data

- Atoms (symbols) including numbers
  - All types of numbers including Roman! (well, in the early days)
  - Syntactically any identifier of alphanumerics
  - Think of as a pointer to a property list
  - Immutable, can only be compared, but also serve as names of variables when used as a variable
- Lists are the primary data object
- There are others
  - Arrays, Structures, Strings (ignore for now)
- S-expressions are interpreted list structures

Data

- Atoms (symbols) including numbers
  - All types of numbers including Roman! (well, in the early days)
  - Syntactically any identifier of alphanumerics
  - Think of as a pointer to a property list
  - Immutable, can only be compared, but also serve as names of variables when used as a variable
- Lists are the primary data object
- There are others
  - Arrays, Structures, Strings (ignore for now)
- S-expressions are interpreted list structures

Functions

- Defined using the **defun** macro

```
(defun name (parameter*)
  "Optional documentation string."
  body-form*)
```

Hello World

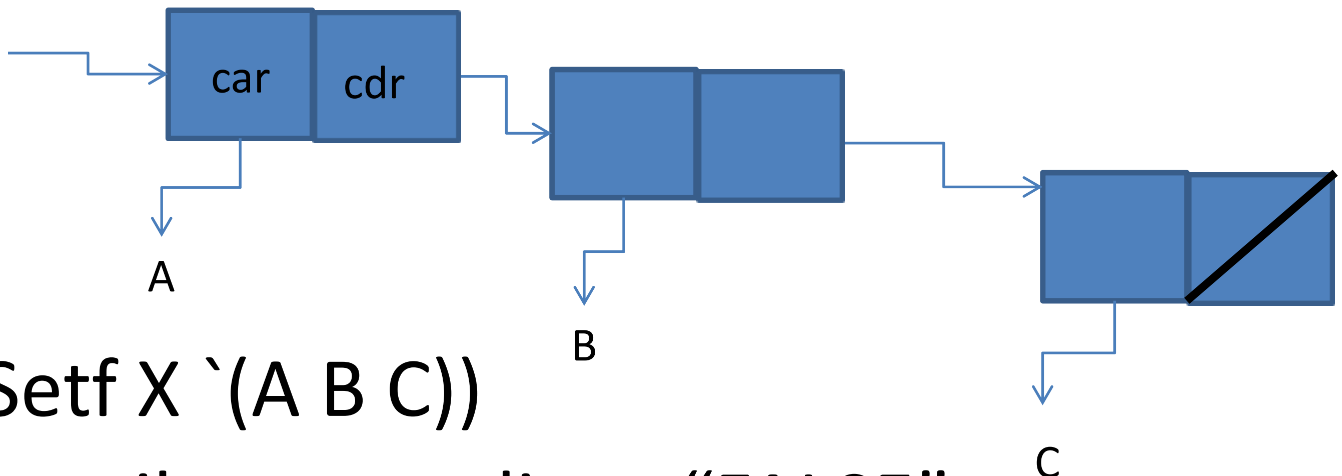
```
(defun hello ()
        (print "hello world“)
)
```

Programs

- Series of function definitions (there are many built-in functions)
- Series of function calls
- Read/Eval/Print
  - (Setf In (Read stdio))
  - (Setf Out (Eval In))
  - (Print Out)
- In other words (Loop (Print (Eval (Read))))

Singly linked Lists

- A “cons” cell has a First field (CAR) and a Rest field (CDR)
- X



- (Setf X `(A B C))
- () = nil = empty list = “FALSE”
  - Nil is a symbol, and a list and its value is false.

List Manipulation Funcs

- Car, First
  - (Car (Car (Car L)))
- Cdr, Rest
  - (Car (Cdr (Cdr L)))
- Cons
  - (Cons ‘1 nil) → (1)
  - (Cons ‘1 `(2)) → (1 2)

car and cdr: What’s in a Name

- Metasyntatic? Arbitrary? Foreign?
- Russel implemented Lisp on IBM 704
- Hardware support for special 36 bit memory treatment
  - **A**ddress
  - **D**ecrement
  - Prefix
  - Tag
- car: **C**ontents of the **A**ddress part of the **R**egister number
- cdr: **C**ontents of the **D**ecrement part of the **R**egister number
- cons: reassembled memory word

List Manipulation Functions

- List
  - (List 1 2 3) → (1 2 3)
- Quote, ‘
  - Don’t evaluate arguments, return them
  - (Quote (1 2)) = `(1 2) = (1 2) as a list with two elements
  - Otherwise “1” better be a function!
- List vs quote: List does not stop evaluation
- Listp
- Push, Pop
- Append
- Remove
- Member
- Length
- Eval

Arithmetic

- The usual suspects:
  - Plus +
  - Difference –
  - Times *
  - Divide /
- Incf
- Decf

Functional Composition

- Prefix notation
  - aka Cambridge prefix notation
  - aka Cambridge Polish notation
- (f (g (a (h t))) → f(g(a, h(t)))

Predicates

- Atom
  - (Atom `(A)) is false, i.e. **nil**, because (A) is a list, not an atom
  - (Atom `A) is true, i.e. **1 or T**
  - (Atom A) is either, depending upon its value! A here is regarded as a variable
- Numberp
- Null
  - (Null `(1)) is nil
  - (Null nil) is T
- Zerop
- And/Or/Not
  - (And A B C) = T if the value of all of the variables are non-nil
  - (Or A B C) = the value of the first one that is non-nil, otherwise nil

Property Lists – Association Lists

- Lisp symbols have associated **property list** structures
- Atom a has property p with value v
- A computing context consists of a set of variables and their current values
  - ((key1 val1) (key2 val2)…)
  - "key" is the name of a variable (a symbol)

Property List Manipulation

- Putprop/Get/Rempro all defunct in Common Lisp
- (Setf (Get Symbol Property) NewValue)
- (Get Symbol Property)

Assignment

- Atoms are variables if they are used as variables
  - Decided by syntactic context
- setq, set, rplaca, rplacd →
- setf
  - The general assignment function, does it all
  - (setf (car list) 5)
  - (setf A 1)

In case you hadn’t noticed

- PROGRAMS/FUNCTIONS have the same form as DATA
- Hmmm….

The Special Expression let

- let defines local variables
- (let ((var1 val) (var2 val) …)

  body)

body is a list of expressions

Conditional Expression

- (If expression expression) or (if expression expression expression)
- What about if-else?
  - Use cond!
- (Cond

  (Expression1 *list of expressions1*)

  (Expression2 *list of expressions2*)

  …

  (ExpressionN *list of expressionsN*) **)**

  First conditional expression that is true, the corresponding list of expressions is executed, and the value of the last one is returned as the value of the Cond.

Conditional Expression

- Use t for else in cond

```
(cond
        ((evenp x) (/ x 2))
        ((oddp x) (* x 2))
        (t x)))
```

Functions

- (Defun Name (variables) *body*)
  - *body* is a list of S-expressions

- Similar to:
  - (Setf Name (lambda(variables) *body*)

- Lambda is the primitive (unnamed) function
  - (Setf X (lambda(y) (Incr y)))
  - Now you can pass X to a function where you can evaluate it with
    - apply, funcall
- (mapcar f arglist)
  - Mapc
  - Map
  - (Mapreduce “borrowed” this off from LISP)

Equality

- Eq – exact same object in memory
- Eql – exact same object in memory or equivalent numbers
- Equal – List comparison too, each component should be "equal" to each other
  - (Equal L M) means every element of L is exactly equal to the corresponding element of M
    - L and M therefore must have the same length and structure, including all sub-components

Examples

```
(Defun mycount (n)
    (Cond ((Equal n 1) ‘one)
          ((Equal n 2) ‘two)
           (T `many)))
```

This function will return one of three Atoms as output, the atom ‘one, or ‘two or ‘many.

```
(Defun Sum (L)
    (Cond
          ((Null L) 0)
          (T (+ (Car L) (Sum (Cdr L)))))
```

This function returns the sum of numbers in the list L. Note: if an element of L is not a number, the “+” function will complain. The LISP debugger will announce it.

More examples

```
(Defun Reverse (L)
        (Cond
                ((Null L) nil)
                (t
                        (Append
                                (Reverse (Cdr L))
                                (List (Car L) ) ) ) ) )
```

This one is not a brain teaser…try it out by hand with a) nil b) a one element list c) a three element list. See how it works? Recursion and functional programming can create interesting results when combined.

More examples

- (Defun Member (x L)

```
(Cond
    ((Null L) nil)
    ((Equal x (car L)) L)
    (t (Member
        (x (Cdr L) ) ) ) )
```

Note: if the value of the variable x is actually a member of the list L, the value returned is the “sub-list” where it appears as the “car”. Hmmm… Try it out by hand.

Second note: What happens if a) x isn’t a member of L, and b) L isn’t a list?

Let’s Give EQUAL a Shot