Lisp Programming

CS161 Discussion Week 1

Basic Info

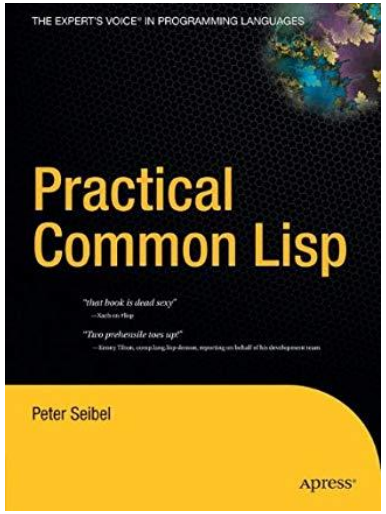
- Syllabus and Slides: CCLE
- Campuswire:
  - https://campuswire.com/p/G2EF3B1E3
  - access code 0628
- Office Hour:
  - Can Join Any TA's Office Hour
  - Check Course Website:
    - http://web.cs.ucla.edu/~guyvdb/teaching/cs161/2021f/

Grading

- Grading:
  - Homework (20%)
  - Midterm (35%) ; Closed book, in person.
  - Final (45%); Closed book, in person.
- Late Policy:
  - Regular programming homework (in LISP)
  - **-25%** of total score each day late

CLISP

- SEASnet:
  - ssh -X lnxsrv.seas.ucla.edu -l yourseasaccountname
  - (Make sure to be connected to UCLA vpn)
- Local machine:
  - https://clisp.sourceforge.io/
- Online:
  - https://jscl-project.github.io/
- ***Practical Common Lisp*** **(Book)**
  - http://www.gigamonkeys.com/book/



Background

- What is Lisp?
  - Originally specified in 1958 by John McCarthy , Lisp is the second-oldest high-level programming language
- Why do we use it in this class?
- Common Lisp
  - The modern, multi-paradigm, high-performance, compiled, ANSI-standardized, most prominent descendant of the long-running family of Lisp programming languages.
  - Object oriented programming and fast prototyping capabilities

Syntax

Two fundamental pieces

- ATOM
- S-EXPRESSION (fully parenthesis, prefix)

Comment

- ;;;; Description of program
- ;;; Basic comment
- ;; Indented with code
- ; End line of code
- #| .... |# Multiple line comments

Atom

```
30                  ; => 30
"Hello!"            ; string

t                   ; denoting true
nil                 ; false; the empty list: ()

:A                  ; symbol
A                   ; Error. Not defined
```

any non-NIL value is true!

Atom

```
99999999999999999999 ; integer
#b111                ; binary => 7
#x111                ; hexadecimal => 273
3.14159              ; float
```

s-expression

function arguments

```
(f x y z ...)

(+ 1 2 3 4)   ; 1+2+3+4 => 10

Use quote or ' to prevent it from being evaluated
'(+ 1 2)                ; => (+ 1 2)
(quote (+ 1 2))         ; => (+ 1 2)
‘(1 2 3)                ; list (1 2 3)
```

Basic arithmetic operations

```
(+ 1 1)          ; => 2
(- 8 1)          ; => 7
(* 10 2)         ; => 20
(expt 2 3)       ; => 8
(mod 5 2)        ; => 1
(/ 35 5)         ; => 7
(/ 1 3)          ; => 0.3333333..
```

Booleans and Equality

```
(not nil)          ; => T
(and 0 t)          ; => T
(or 0 nil)         ; => 0
(and 1 ())         ; => 0
```

empty list

Booleans and Equality

compare numbers

```
(= 3 3.0)            ; => T
(= 2 1)              ; => NIL
```

compare object identity

```
(eql 3 3)            ; => T
(eql 3 3.0)          ; => NIL
(eql (list 3) (list 3)) ; => NIL
```

compare lists, strings

```
(equal (list 'a 'b) (list 'a 'b)) ; => T
(equal (list 'a 'b) (list 'b 'a)) ; => NIL
```

Strings

type

```
(concatenate 'string "Hello," "world!") ;
=> "Hello,world!"


(format nil "Hello, ~a" "Alice") ; returns
"Hello, Alice"

(format t “Hello, ~a” “Alice”) ; returns
nil. formatted string goes to standard
output


(print "hello")   ; value is returned and
printed to std out

(+ 1 (print 2))   ; prints 2. returns 3.
```

Variables

- global (dynamically scoped) variable
- The variable name can use any character except: ()",'`;#|\

```
(defparameter age 35)
age                ; => 35
name               ; error

(defparameter *city* "LA")
*city*             ; => "LA"
```

Variables

```
(defparameter age 35)  ; age => 35
(defparameter age 60)  ; age => 60

(defvar newage 20)     ; newage => 20
(defvar newage 60)     ; newage => 20

(setq newage 30)       ; newage => 30
```

defvar does not change the value of the variable!

Set

- (set ls '(1 2 3 4)) ; Error - ls has no value
- (set 'ls '(1 2 3 4)) ; OK
- (setq ls '(1 2 3 4)) ; OK - make ls to (quote ls) and then have the usual set
- (setf ls '(1 2 3 4)) ; OK-same as setq so far BUT
- (setf (car ls) 10) ; Makes ls '(10 2 3 4) - not duplicated by setq/set

- **Note:** Not allowed to use this in homework, mostly helpful for your debugging purposes.

Local variable

```
(let ( (a 1) (b 2) )   ; binding
    (+ a b)         ; body
)
```

You will NOT be allowed to set global variables in your homework!

let only

Lists

- Linked-list data structures
- Made of CONS pairs

```
(cons 1 2)                          ; => '(1 2)
(cons 3 nil)                        ; => '(3)
(cons 1 (cons 2 (cons 3 nil)))      ; => '(1 2 3)
(list 1 2 3)                        ; => '(1 2 3)
(cons 4 '(1 2 3))                   ; => '(4 1 2 3)
(cons '(4 5) '(1 2 3))              ; =>  ?
```

Lists

```
(cons 1 (cons 2 (cons 3 nil)))      ; => '(1 2 3)
(list 1 2 3)                        ; => '(1 2 3)
(cons 4 '(1 2 3))                   ; => '(4 1 2 3)
(cons '(4 5) '(1 2 3))              ; => '((4 5) 1 2 3)

(append '(1 2) '(3 4))              ; => '(1 2 3 4)
(append 1 '(1 2))                   ; ERROR!
(append '(1 2) '(3 4))  ; => '(1 2 3 4)

(car '(1 2 3 4))                    ; => 1
(cdr '(1 2 3 4))                    ; => '(2 3 4)
car and cdr should be used for list
```

Functions

- Define a function

```
(defun hello (name) (format nil "Hello, ~A" name))
```

- Call the function

```
(hello "Bob")           ; => "Hello, Bob"
```

Control Flow

```
(if (equal *name* "bob")    ; test expression
    "ok"                    ; then expression
    "no")                   ; else expression
```

- Chains of tests: cond

```
(cond ((> *age* 20) "Older than 20")
      ((< *age* 20) "Younger than 20")
      (t "Exactly 20"))

(cond ((> *age* 20) "Older than 20")
      ((< *age* 20) "Younger than 20")); NIL when *age*=20
```

Programming Practice!

- Factorial
- compute list length
- find kth element
- check if list contains a number
- delete kth element

Factorial

```
(defun factorial (n)
  (if (< n 2)
    1                       ; returns 1 when n<2
    (* n (factorial (- n 1)))   ; when n>=2
   )
)

(factorial 5)                       ; => 120
```

Compute list length (top-level)

```
'((a b) (c (d 1)) e)  => 3

(defun listlength (x)
      (if (not x) ; base case: empty list
            0
            (+ (listlength (cdr x)) 1)
      )
)
```

'(1 2 3 4) -> '(2 3 4)

Compute list length (deep)

```
'((a b) (c (d 1)) e)  => 6

(defun deeplength (x)
      (cond ((not x) 0); empty list. returns 0
            ((atom x) 1)    ; atom. returns 1
            (t (+ (deeplength (car x)) ; else
                  (deeplength (cdr x))
               )
            )
      )
)
```

Check if list contains an element

```
(defun contains (e x)
       (cond ((not x) nil)
             ((atom x) (equal e x))
             (t (or (contains e (car x)) (contains
e (cdr x))))
       )
)

(contains 'a '((b a) (1 e c)))
```

Check if list contains a number

```
Consider this case: '((a b) (c (d 1)) e)


(defun contains_number (x)
       (if (atom x)              ; NIL if x is a list
           (numberp x)           ; numberp: check if x
is a number
           (or (contains_number (car x))
                 (contains_number (cdr x))    ;
recursively flatten
           )
       )
)
```

Find kth element (top-level)

```
(defun find_kth (k x)
       (if  (= k 1)
              (car x)
              (find_kth (- k 1) (cdr x))
       )
)
```

How do we find kth element in the flattened list?

3, '((a b) (c (d 1)) e)   => c

Delete kth element

```
(defun delete_kth (k x)
      (if  (= k 1)
            (cdr x)
            (cons (car x)
                  (delete_kth (- k 1) (cdr x))
            )
      )
)
```