

# CSC 244/444 (Fall 2011)

## Lisp Tutorial

Daphne Liu

August 29, 2011

### 1 Getting Started / Quitting

- `acl` for Allegro Common LISP, or `cmucl`
- `(load "filename")`: Loads a LISP file
- `(exit)` for Allegro Common LISP, or `(quit)` for CMU CL

### 2 Basics

- S-expressions

Atoms

Numbers

Integer e.g., 5, -1

Floating point e.g., 2.5, -3.2

Literals

e.g., 'a, 'b

Non-atomic S-expressions

Dotted pairs e.g., '(2 . 5), '(1 2 . (3 2))

Lists

Data lists e.g., '(a 5 2), '((a b))

Forms e.g., (+ 5 2), (cons 'a '(b c))

- Evaluation of form **(op arg<sub>1</sub> arg<sub>2</sub> ... arg<sub>n</sub>)** in *prefix* notation
  1. Evaluate *arg<sub>1</sub>*, *arg<sub>2</sub>*, ... , and *arg<sub>n</sub>*.
  2. Apply function to the evaluated arguments.

e.g., `(list (* 2 4) (+ 5 6))` yields `(8 11)`.

- In order for LISP not to evaluate an argument, prefix it with an apostrophe (single quote);  
e.g., `(list '(* 2 4) (+ 5 6))` yields `((* 2 4) 11)`.
- Each line of comment in LISP code prefixed with a `;`
- Constant `t` or `T` for `true`, and `NIL` or `nil` for `false` or an empty list

### 3 Lists

- A data list is a (possibly empty) sequence of objects;  
e.g., `(a b)`, `((B))`, `NIL` or `()`.
- `first` or `car`: Returns the first element of a given list  
e.g., `(first '(a b))`  $\rightarrow$  `A`
- `rest` or `cdr`: Returns a list of all but the first element of a given list  
e.g., `(cdr '(a b))`  $\rightarrow$  `(B)`
- `cons`: Inserts a given element to the head of a given list  
e.g., `(cons 'a '(a b))`  $\rightarrow$  `(A A B)`
- `list`: Returns a list composed of any given arguments  
e.g., `(list 'a '(a b))`  $\rightarrow$  `(A A B)`
- `append`: Appends one given list to the end of another given list  
e.g., `(append '(a b) '(x))`  $\rightarrow$  `(A B X)`
- See also `length`, `reverse`, `nth`, `last`, `member`, `empty`.

### 4 Predicates

- `equal`: Determines equality between ordered contents of two given lists
- `eql`: Determines equality between two given symbols
- `=`: Determines equality between the values of two given numbers
- `null`: Determines whether a given argument is `NIL`
- `and`, `or`, `not`: logical predicates
- `listp`, `numberp`, `integerp`, `floatp`, `stringp`, `atom`: type-checking predicates
- `evenp`, `oddp`, `>`, `<`, `>=`, `<=`: numeric predicates

## 5 Branches

- `(if (expr) (then_form) [(else_form)])`: If `(expr)` evaluates to `t`, evaluate `(then_form)`, else evaluate `(else_form)`.
- `cond`: for multiple branches

```
(cond ((test1) (form1))
      ((test2) (form2))
      ...
      ((testn) (formn))
)
```

## 6 Assignment

- `setq` or `setf`: Assigns the value of a given argument to a given variable

```
(setq tem 'apple) --> APPLE
(cons tem '()) --> (APPLE)
```

- See `let` and `let*` for local variables.

## 7 User-Defined Functions

- `defun`: Defines and names a user-defined function

```
(defun my-square (x) (* x x))
(my-square 5) --> 25
```

- `lambda`: Defines an anonymous function

```
(setq foo (lambda (x) (* x x)))
(funcall foo 5) --> 25 ;(foo 5) does not evaluate to 25, though.
```

- `funcall`: Applies a given function to a sequence of given arguments

```
(funcall foo 5) --> 25
(funcall #'+ 3 3 3) --> 9
(funcall (lambda (x) (* 2 x)) 5) --> 10
```

- `apply` is similar to `funcall`, but takes all arguments in a list; e.g., `(apply #'+ '(3 3 3)) --> 9`

## 8 Iteration and Recursion

- See `dolist`, `loop`, `dotimes`, `do`, and `do*` for iteration constructs.

```
(loop for N in '(1 2 3) do (print N))
(loop for N from 1 to 10 do (print N))
```

- A recursive function calls and uses itself, and recursion can be used instead of iteration.

```
(defun incr(x)
  (cond ((null x) nil)
        (t (cons (+ 1 (car x)) (incr (cdr x))))))
(incr '(1 2 3)) --> (2 3 4)
```

## 9 Higher-Order Operations

- `mapcar`: Applies a given function to each top-level element in a given list

```
(mapcar (lambda (x) (* 2 x)) '(1 2)) --> (2 4)
(mapcar \#'oddp '(1 2 3)) --> (T NIL T)
```

- `reduce`: Applies a given binary function to given arguments from left to right

```
(reduce #'* '(1 2 3 4)) --> 24
(reduce #'- '(1 2 3 4)) == (- (- (- 1 2) 3) 4) --> -8
```

- See `find`, `find-if`, `find-if-not`, `remove`, `remove-if`, `remove-if-not` with `:test` and `:key` in class hand-outs.

## 10 Miscellaneous

- `eval`: Evaluates and returns the value of a given argument

```
(eval (* 5 2)) --> 10
(eval 5) --> 5
(eval 'apple) --> APPLE
(setq foo '(+ 2 3))
(eval foo) --> 5
```

- `print`: Prints the value of a given argument
- `format`: equivalent of C's `fprintf` for formatted printing e.g., `(format t "hello")`