Spring 1997 **A Short Reference Manual for LISP** CMSC 420

David Traum

This guide[1] details a few important functions of Common Lisp. Note that in Common Lisp, the expression reader will convert lower case to upper case, except for strings, so you may generally use either upper or lower case in your programs. These are the functions which come predefined, and are available at sign-on to Lisp. This is not a complete list.

1 Using Lisp on our system

1. Log into your account. Type `lisp` at the command line. Lisp will load and leave you at a prompt that looks like this:

   ```
   *
   ```

2. Now you are in a read-eval loop: when you type something, the system will read it in, evaluate it and print the result before giving you the next prompt.

   For example:

   ```
   *  (+ 2 2)

   4
   *
   ```

   In the examples below, we show the result following a →, to save space.

3. Some helpful commands.

`(quit)`	To quit lisp and return to your shell.
`:q`	To return to the top level
`:help`	For some limited help

4. Emacs
   Those of you familiar with the unix editor *Emacs* will greatly prefer running lisp within this editor. To do so, simply type `M-x run-lisp` to get lisp running in an emacs buffer. Others who want to use lisp (or other programming langauges) may decide that learning to use emacs is worth the trouble. To run emacs, just type `emacs` to the shell. To quit, type `C-x C-c`. (Note: in emacs, `C-c` means Control-c and `M-x` means Meta-x. If you don't have a Meta key, on your keyboard, use the Escape key. More information may be provided later about using lisp in emacs.

[1] Based on material from Hanan Samet and Carl Andersen

2 S-expression manipulation

`(car arg)`	Returns the first element in list `arg`, or the left pointer of an s-expression (cons node).
	`* (car '(1 2 3))` → `1`
	`* (car '(a . b))` → `A`

`(cdr arg)`	Return the rest of list `arg`, or the right pointer of an s-expression (cons node).
	`* (cdr '(1 2 3))` → `(2 3)`
	`* (cdr '(a . b))` → `B`

`(cons a1 a2)`	Allocate and return a cons node that points to `a1` and `a2`.
	`* (cons 'a '(1 2 3))` → `(A 1 2 3)`
	`* (cons 'a 'b)` → `(A . B)`

`(list a1 a2 ...an )`	Return a list of all the arguments. `* (list 1 2 3 4 5)` → `(1 2 3 4 5)`
`(append a1 a2)`	Return the merge of the two lists `a1` and `a2`. `* (append '(a b c) '(1 2 3))` → `(A B C 1 2 3)`
`(reverse arg)`	Reverses the list `arg`. `* (reverse '(1 2 3))` → `(3 2 1)`
`(subst new old sexp)`	Return a copy of `sexp` with all `old`'s converted to `new`'s. `* (subst 'a 'b '(a b c))` → `(A A C)`

3 S-expression predicates

`(null arg)`	Return `t` if `arg` is `nil`; `nil` otherwise.
	`* (null nil)` → `T`
	`* (null '())` → `T`
	`* (null 'a)` → `NIL`
	`* (null '(a b c))` → `NIL`

`(atom arg)`	Return `nil` if `arg` is a cons node; `t` otherwise.
	`* (atom nil)` → `T`
	`* (atom 'a)` → `T`
	`* (atom '(a b c))` → `NIL`

`(eq a1 a2)`	Return `t` if `a1` and `a2` point to the same object; `nil` otherwise.
	`* (eq 'a 'b)` → `NIL`
	`* (eq 'a 'a)` → `T`
	`* (eq '(a b) '(a b))` → `NIL`
	`* (setq x '(a b))`
	`* (eq (car (list x)) x)` → `T`

`(equal a1 a2)`	Return `t` if `a1` and `a2` have equivalent structure; `nil` otherwise.
	`* (equal 'a 'b)` → `NIL`
	`* (equal '(a b) '(a b))` → `T`

`(member a list)`	Return the sublist of `list` beginning with the first occurrence of `a`; `nil` if not in `list`.
	`* (member 'b '(a b c)` → `(B C)`
	`* (member 'a '(1 2 3))` → `NIL`

4 Logical functions

`(or a1 a2 ...an)`	Return `t` if any argument is non-`nil`; `nil` otherwise.
	`* (or 'a nil 'c)` → `A`
	`* (or nil nil nil)` → `NIL`

`(and a1 a2 ...an)`	Return `t` if all arguments are non-`nil`; `nil` otherwise.
	`* (and 'a nil 'c)` → `NIL`
	`* (or 'a 'b 'c)` → `C`

`(not arg)`	If `arg` is `nil`, returns `t`; else returns `nil`.
	`* (not nil)` → `T`
	`* (null 'a)` → `NIL`

5 Control and program functions (special forms)

`(quote a)` or `'a`	Return `a` unevaluated.
	`* 'a` → `A`
	`* (quote a)` → `A`
	`* '(cons a nil)` → `(CONS A NIL)`

`(cond`
`  (exp1 a1 a2 ...an)`
`  (exp2 b1 b2 ...bm)`
`  ...`
`  (expn z1 z2 ...zk))`
	Evaluates all `exp` until one is non-`nil`, then evaluates all expressions that follow and returns the result of the last.
	`* (cond ((null 'a) 1) ((eq 'a 'a) 2))` → `2`
	`* (cond ((null nil) 1) ((eq 'a 'a) 2))` → `1`

`(setq x y)`	Set `x` to the value of `y`.
	`* (setq x '(A B))`
	`* (car x)` → `A`

`(set x y)`	Like `setq`, except that `x` is evaluated to get an atom.
	`* (set (car x) '(1 2))`
	`* a` → `(1 2)`

`(lambda arglist body)`	Return a nameless function with argument list `arglist` and body `body`. The body should be a sequence of expressions, with the value of the last one being the value of the function. `* ((lambda (x) (car x)) '(1 2))` → `1`

`(defun name arglist`
`  body)`
	Define a function with name `name`.
	`* (defun foo (x) (car (cdr x))`
	`* (foo '(1 2 3))` → `2`

6 Arithmetic Functions

`(numberp n)`	Return `t` if `n` is a number; `nil` otherwise.
`(zerop n)`	Return `t` it `n` is zero; `nil` otherwise.
`(> n1 n2)`	Return `t` if `n1` is greater than `n2`.
`(< n1 n2)`	Return `t` if `n1` is less than `n2`.
`(+ n1 n2 ...nn)`	Return the sum of all arguments.
`(* n1 n2 ...nn)`	Return the product of all arguments.
`(- n1 n2)`	Return the quantity `n1` minus `n2`.
`(/ n1 n2)`	Return `n1` divided by `n2`.
`(1+ x)`	Return `x + 1`.
`(1- x)`	Return `x - 1`.

7 I/O Functions

`(read)`	Return as a value the next s-expression typed as input to the terminal.
`(read-char)`	Return next character as an atom.
`(print arg)`	Print `arg` to output.
`(terpri)`	Print a newline character.
`(symbol-function atom)`	Print the function definition of the function bound to `atom`.
`(load filename)`	Read in the file `filename` and evaluates all expressions in it, including function definitions. Values of expressions are not displayed.
`(dribble filename)`	Save the interaction between you and the lisp evaluator in file `filename`

8 Atom Manipulation

`(gensym arg)`	Create atom with name `argnnn`, where `arg` is a string and `nnn` is the number of times `gensym` has been called. `(gensym 'x)` might return `#:X123`.
`(get atom label)`	Return the value of property `label` on the property list of `atom`.
`(setf (get atom label) value)`	Set the value of the property `label` of `atom`'s property list to `value`.
`(remprop atom label)`	Remove the property `label` from the property list of `atom`.
`(symbol-plist atom)`	Return the property list of `atom` in the form `(p1 v1 p2 v2 ... pn vn)`, where `pi` is a property with value `vi`.

9 Other useful functions

`(nth number list)`	Return the element of `list` with index `number`, assuming zero-based indexing.
`(nthcdr number list)`	Return the result of applying `cdr` to the `list number` times.
`(length list)`	Return the number of elements in the top level of `list`.
`(trace fname ...)`	Turn on function tracing for function `fname`. Trace can take multiple arguments. A list of all functions being traced is returned.
`(untrace fname ...)`	Turn off tracing for the named functions.