Dpto. de Álgebra, Computación, Geometría y Topología
Universidad de Sevilla

Manual de Lisp para IACS (Curso 91–92)

Sevilla, 1992

Contenido

1 Introducción

1.1 Introducción

Para cargar el LeLisp se escribe:

```
C> lelisp
```

y el sistema responde indicando la versión y se entra en el bucle principal del intérprete que va, indefinidamente a leer una expresión sobre el terminal, evaluarla e imprimir el valor de esta evaluación. LeLisp indica que espera la lectura de una expresión imprimiendo el carácter `?` al comienzo de cada línea. El valor de la evaluación de una expresión se imprime en la línea siguiente precedido por `=`. Para salir se escribe `(end)`.
Veamos un ejemplo de sesión en LeLisp:

```
C> lelisp
? (+ 2 3)
= 5
* (end)
C>
```

1.2 Los objetos básicos

Los objetos que se usan en Lisp se llaman S–expresiones (por "Symbolic expressions").
Estos objetos se clasifican en los siguientes tipos:

$$\text{S–expresiones} \begin{cases} \text{átomos} \begin{cases} \text{números} \\ \text{símbolos} \\ \text{cadenas de caracteres} \end{cases} \\ \text{listas} \end{cases}$$

Para referirnos a dichos objetos, usaremos las siguientes abreviaturas:

s	S–expresión
a	átomo
simb	símbolo
n	número
l	lista

Los átomos
Los **símbolos** son cadenas continuas de caracteres (conteniendo al menos un carácter no numérico). Por ejemplo, `agua, a12, var-aux, +` son símbolos. LeLisp manipula **números** enteros (desde -32767 a 32767), decimales (por ejemplo, `23.45`) y notación científica (por ejemplo, `3.256e+102`).

Una **cadena** de caracteres es una sucesión de caracteres, con o sin huecos, que comienza y termina por dobles comillas. Por ejemplo, `"a 1 23"` es una cadena de caracteres.

Las listas

Una **lista** es una sucesión ordenada, posiblemente vacía, de objetos. Sintácticamente, se compone de un paréntesis abierto, objetos separados por huecos y un paréntesis cerrado. Por ejemplo, `(a 1 b), (), (a (b (c)))` son listas.

1.3 Funcionamiento básico del intérprete

Evaluación de los átomos

El valor de un número es el propio número.
El valor de un símbolo es:

- el número que tenga asignado, si actúa como variable numérica;
- la S–expresión que tenga asignada, en caso contrario.

El valor de una cadena de caracteres es la propia cadena.

Evaluación de las listas

Las listas se interpretan como llamadas a funciones. El primer elemento es el nombre de la función y el resto son los argumentos. Por ejemplo, la lista `(+ 2 3)`, se interpreta como la función `+` actuando sobre `2` y `3`.

2 Definición de funciones y variables

2.1 Definición de funciones

`(defun simb l s1...sN)`

permite definir nuevas funciones.

`simb` es el nombre de la función definida.

`l` es la lista de parámetros (argumentos); son variables locales que no afectan a posibles valores previos, en general. Si no hay argumentos, es obligatorio poner ().

`s1`,..., `sN` son las expresiones que definen el cuerpo de la función.

Devuelve `simb`.

```
? (defun cuadrado (n) (* n n))
= cuadrado
? (cuadrado 3)
```

```
= 9
```

`((lambda (var1...varN) s1...sM) val1...valN)`
asocia los valores `val1`,..., `valN` a las variables `var1`,..., `varN`; evalúa las expresiones `s1`,..., `sM` y devuelve el valor de `sM`.

```
((lambda (n) (* n n)) 3)  =>  9
```

2.2 Definición de variables

`(setf simb1 s1 simb2 s2 ...  simbN sN)`
asigna a `simb1` el valor de la expresión `s1`, ..., a `simbN` el valor de la expresión `sN` y devuelve el valor de la última expresión.

```
(setf x 3 y (+ x 2))  =>  5
y                     =>  5
```

`(setq simb1 s1 simb2 s2 ...  simbN sN)`
es equivalente a `(setf simb1 s1 simb2 s2 ...  simbN sN)`.

`(let ((var1 val1)...(varM valM)) s1...sN)`
asocia, en paralelo, a las variables `vari` los valores `vali`, evalúa las expresiones `si` y devuelve el valor de `sN`.

```
? (let ((a 2)
?       (b 3))
?      (+ a b))
= 5
? a
** error: a variable sin valor
```

3 Funciones sobre números

3.1 Operaciones aritméticas

`(+ n1 n2 ...  nN)`
devuelve el valor de la suma $n1 + n2 + ... + nN$.

```
(+ 3 7 5)            =>  15
```

`(1+ n)`
es equivalente a `(+ n 1)`.

`(- n1 n2 ... nN)`
devuelve el valor de $n1 - n2 - ... - nN$, si $N > 1$ y $-n1$, si $N = 1$.

```
(- 3)                =>  -3
(- 123 7 5)          =>  111
```

`(1- n)`
es equivalente a `(- n 1)`.

`(abs n)`
devuelve el valor absoluto de `n`.

```
(abs -3.6)    =>  3.6
```

`(* n1 n2...nN)`
devuelve el valor del producto $n1.n2...nN$.

```
(* 2 7 5)            =>  70
```

`(/ n1 n2)`
devuelve el valor de dividir `n1` por `n2`.

```
(/ 6 2)    =>  3.0
(/ 5 2)    =>  2.5
```

`(/ n)`
es lo mismo que `(/ 1 n)`; es decir, devuelve el inverso de `n`.

```
(/ 2)      =>  0.5
(/ 0.5)    =>  2.0
```

`(modulo n1 n2)`
devuelve el resto de la división entera de `n1` por `n2`.

```
(modulo 7 2)   =>  1
```

`(quo n1 n2)`
devuelve cociente entero de `n1` por `n2`.

```
(quo 7 2)   =>  3
```

3.2 Funciones numéricas

`(sqrt n)`
devuelve la raíz cuadrada de `n`.

`(power n m)`
devuelve el valor de n^m.

`(random n m)`
devuelve un número aleatorio del intervalo $[n, m)$.

4 Expresiones condicionales y predicados

4.1 Valores lógicos

`()`
su valor es `()` y representa "lo falso".

`t`
su valor es `t` y representa "lo verdadero".

4.2 Predicados numéricos

`(= n1 ... nN)`
devuelve `t` si los valores de todos los argumentos son iguales; `()`, en caso contrario.

```
(= 10 (+ 3 7))     =>  t
(= 2 2.0 (+ 1 1))  =>  t
(= 1 2 3)          =>  ()
```

```
(= 1 2 1)          =>  ()
```

`(/= n1 ... nN)`
devuelve `t` si los valores de todos los argumentos son distintos; `()`, en caso contrario.

```
(/= 10 (+ 3 7))     =>  ()
(/= 2 2.0 (+ 1 1))  =>  ()
(/= 1 2 3)          =>  t
(/= 1 2 1)          =>  ()
```

`(>= n1 ... nN)`
devuelve `t` si $n1 \geq \ldots \geq nN$; `()`, en otro caso.

```
(>= 4 3 3 2)  =>  t
(>= 4 3 3 5)  =>  ()
```

`(> n1 ... nN)`
devuelve `t` si $n1 > \ldots > nN$; `()`, en otro caso.

```
(> 4 3 2 1)  =>  t
(> 4 3 3 2)  =>  ()
```

`(<= n1 ... nN)`
devuelve `t` si $n1 \leq \ldots \leq nN$; `()`, en otro caso.

```
(<= 2 3 3 4)  =>  t
(<= 5 3 3 4)  =>  ()
```

`(> n1 ... nN)`
devuelve `t` si $n1 < \ldots < nN$; `()`, en otro caso.

```
(< 1 2 3 4)  =>  t
(< 1 3 3 4)  =>  ()
```

`(zerop n)`
es equivalente a `(= n 0)`.

`(evenp n)`
devuelve `t` si `n` es par; `()`, en caso contrario.

`(oddp n)`
devuelve `t` si `n` es impar; `()`, en caso contrario.

4.3 Condicionales

`(if s s1 s2)`
devuelve `s1`, si `s` no es `()` y `s2`, en caso contrario.

```
(if t   1 2)  =>  1
(if ()  1 2)  =>  2
```

`(cond l1...lN)`
es la función condicional más general de Lisp. Cada `li` tiene una estructura del tipo:

`(condicion s1...sM)`

`cond` va evaluando las `condiciones`; cuando encuentra la primera distinta de `()`, evalúa las correspondientes expresiones `si` y devuelve el valor de la última.

Si la condición seleccionada no va acompañada de expresiones, devuelve justamente el valor de la condición.

Si ninguna condición se satisface, devuelve `()`.

Para forzar la selección, suele escribirse `t` para la última condición; su sentido viene a ser "en otro caso..."

```
? (defun notas (n)
?        (cond ((n 5) 'suspenso)
?              ((n 7) 'aprobado)
?              ((n 9) 'notable)
?              (t 'sobresaliente) ))
= notas
? (notas 8)
= notable
```

4.4 Operadores lógicos

`(not s)`
devuelve `t`, si `s` es `()`; `()`, si no.

`(or s1...sn)`
evalúa sucesivamente `s1`,..., `sN` hasta que una de dichas expresiones tenga un valor distinto de `()` y devuelve este valor. Si el valor de todas las expresiones es `()`, entonces devuelve `()`.

```
(or () 2 3)  =>  2
```

`(and s1...sn)`
evalúa sucesivamente `s1`,..., `sN` hasta que el valor de una de dichas expresiones sea `()`; en cuyo caso, devuelve `()`. Si el valor de todas las expresiones es distinto de `()`, entonces devuelve el valor de `sn`.

```
(and 1 2 3)    =>  3
(and 1 () 3)   =>  ()
```

5 Funciones de control de la evaluación

5.1 Inhibición de la evaluación

`(quote s)`
devuelve `s` (sin evaluar).

```
(quote (+ 2 3))  =>  (+ 2 3)
(quote a)        =>  a
```

`'s`
es lo mismo que `(quote s)`.

```
'(+ 2 3)  =>  (+ 2 3)
'a        =>  a
```

5.2 Forzamiento de la evaluación

`(eval s)`
devuelve el valor de `s`.

```
(eval (cons '+ '(2 3)))  =>  5
```

6 Funciones sobre átomos y listas

6.1 Funciones básicas sobre listas

`(car l)`
devuelve el primer elemento de `l`.

```
(car '(a b c))  =>  a
```

`(cdr l)`
devuelve la lista formada por los elementos de `l`, excepto el primero.

```
(cdr '(a b c))  =>  (b c)
(cdr ())        =>  ()
```

`(cons s l)`
devuelve la lista cuyo `car` es `s` y cuyo `cdr` es `l`.

```
(cons 'a '(b c))      =>  (a b c)
(cons '(a b) '(c d))  =>  ((a b) c d)
```

6.2 Predicados de tipos

`(null s)`
devuelve `t`, si `s` es la lista vacía y `()`, si no.

`(atom s)`
devuelve `t`, si `s` es un átomo y `()`, si no.

```
(atom 12)      =>  t
(atom 'abc)    =>  t
(atom "a b")   =>  t
(atom a)       =>  error
(atom '(a b))  =>  ()
(atom ())      =>  t
```

`(numberp s)`
devuelve `t`, si `s` es un número y `()`, si no.

```
(numberp 123)  =>  t
(numberp 'a)   =>  ()
(setq a 3)     =>  3
(numberp a)    =>  t
```

6.3 Predicados de igualdad

`(eq a1 a2)`
devuelve `t` si `a1` y `a2` son el mismo átomo y `()` en caso contrario.

```
(eq 'lisp 'lisp)        => t
(eq 'lisp 'lisa)        => ()
(eq (car '(a b c)) 'a)  => t
(eq 3 3.0)              => ()
(eq '(b c) '(b c))      => ()
```

`(equal s1 s2)`
devuelve `t`, si `s1` y `s2` tienen el mismo valor y `()`, si no.

```
(equal 'lisp 'lisp)     =>  t
(equal 3 3.0)           =>  t
(equal '(b c) '(b c))   =>  t
```

6.4 Funciones auxiliares sobre listas

`(c...r l)`
es una combinación de las funciones `car` y `cdr` hasta 4 niveles.

```
(cadr '(a b c))  =>  b
```

`(member s l)`

devuelve la sublista de `l` que comienza por el primer elemento de `l` que es igual a `s` si hay alguno que lo sea; `()`, en otro caso.

```
(member 'x '(a x b x c))        =>  (x b x c)
(member 'x '(a (x) b))          =>  ()
(setq l' ((a b) (c d)))         =>  ((a b) (c d))
(member '(c d) l)               =>  ((c d))
(member 2.0 '(1 2 3))           =>  (2 3)
```

`(nth n l)`

devuelve el n–ésimo elemento de `l`. Se empieza a contar por 0.

```
(nth 1 '(a b c))  =>  b
```

`(last l)`

devuelve la lista formada por el último elemento de la lista `l`.

```
(last '(a b c))  =>  (c)
```

`(length l)`

devuelve el número de elementos de `l`.

```
(length '(a (b c) d))  =>  3
```

`(list s1 s2 ...  sn)`

devuelve la lista cuyos elementos son las expresiones `si`.

```
(list 'a 'b 'c)       =>  (a b c)
(list '(a b) '(c d))  =>  ((a b)(c d))
```

`(makelist n s)`

devuelve una lista formada por `n` elementos `s`.

```
(makelist 3 'a)  =>  (a a a)
```

`(append l1 ... ln)`
devuelve una copia de la concatenación de las listas `l1`,..., `ln`.

```
(append '(a b) '(c d))        =>  (a b c d)
(append '(a) '(b) () '(d a))  =>  (a b c d a)
```

`(reverse l)`
devuelve la lista `l`, con sus elementos en orden inverso.

```
(reverse '(a (b c) d))  =>  (d (b c) a)
```

`(subst sn sa s)`
devuelve la expresión `s`, en la que se ha sustituído `sa` (expresión antigua) por `sn` (expresión nueva) a todos los niveles.

```
(subst '(x y) 'a '(a b (a c)))  =>  ((x y) b ((x y) c))
```

`(remove s l)`
devuelve la lista `l`, en la que se han borrado las estancias, de primer nivel, de la expresión `s`.

```
(remove 'a '(a b (a c) a))  =>  (b (a c))
```

6.5 Funciones de aplicación

`(apply fn l)`
devuelve el valor de la función `fn` actuando sobre los elementos de `l` como argumentos. Es equivalente a `(eval (cons 'fn l))`.

```
(apply '+ '(1 2 3))  =>  6
```

`(funcall fn s1...sN)`
es equivalente a `(apply fn (list s1 ... sN))`.

```
(funcall '+ 1 2 (- 6 3))  =>  6
```

`(mapcar fn l1...lN)`
devuelve la lista con los resultados de aplicar `fn` a los primeros de las `li`, luego a los segundos, hasta que una se termina.

```
(mapcar 'atom '(1 (1) a))      =>  (t () t)
(mapcar '+ '(1 2 3) '(4 5 6))  =>  (5 7 9)
```

7 Misceláneas

7.1 El rastreador

`(trace fn1 ... fnN)`
Permite rastrear las funciones `fn1`,..., `fnN` mostrando la acción de las funciones `fn1`,..., `fnN` cada vez que actúan.

`(untrace fn1 ... fnN)`
elimina el efecto de `trace` de las funciones `fn1`,..., `fnN`.

`(untrace)`
elimina el efecto de `trace` de toda función que lo tenga. Devuelve la lista de las funciones con `trace`.

7.2 Funciones de escritura

`(print s1 ... sN)`
imprime los valores de las expresiones `s1`,...,`sN` y devuelve el valor de la última.

```
? (print "(fib(" (+ 1 3) ")= " 3)
(fib 4) = 3
= 3
```

7.3 Funciones sobre cadenas

`(concat c1 ... cN)`
devuelve la concatenación de las cadenas `c1`,...,`cN`.

```
? (concat (+ 1 4) "/" 6)
= 5/6
```

7.4 Funciones sobre el sistema

`(end)`
: termina la sesión Lisp.

`(runtime s)`
: devuelve el tiempo usado desde el comienzo de la sesión.

8 El editor y los ficheros de programas

8.1 Llamada al editor

`(edit)`
: invoca al editor

8.2 Funciones del editor

Control-X Control-R
: lee un fichero. Si no existe, lo crea.

Control-X Control-E
: evalúa el fichero que se está editando.

Control-X Control-S
: salva el fichero.

Control-X Control-C
: cambia del editor a la sesión Lisp.

8.3 Funciones para cargar ficheros

`(load f)`
: carga el fichero `f`.

```
? (load "a:fib.ll")
= a:fib.ll
```

Indice