Contenido

1 Introducción

1.1 Introducción

Para cargar el GOLDEN COMMON LISP se escribe:

```
C> GCLISP
```

y el sistema responde:

```
GOLDEN COMMON LISP, Version 1.01
Copyright (C) 1985 by Gold Hill Computers

; Reading file INIT.LSP
Type Alt-H for Help
Top-Level
*
```

GCLISP 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. GCLISP 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. Para salir se escribe `(EXIT)`. Veamos un ejemplo de sesión con GCLISP:

```
C> GCLISP

GOLDEN COMMON LISP, Version 1.01
Copyright (C) 1985 by Gold Hill Computers

; Reading file INIT.LSP
Type Alt-H for Help
Top-Level
* (+ 2 3)
5
* (EXIT)
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

1.2.1 Los átomos

Los símbolos
Los símbolos son cadenas continuas de caracteres (conteniendo al menos un carácter no numérico). Por ejemplo, `AGUA, A12, VAR-AUX, +` son símbolos.
Los números
GCLISP manipula números enteros sobre 16 bits (permitiendo calcular en el intervalo $[-2^{15}+1, 2^{15}-1]$, i.e. [-32767, 32767]) y números flotantes sobre 128 bits (permitiendo calcular en el intervalo [-1.0F+38, 1.0F+38]).
Las cadenas de caracteres
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.

1.2.2 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

1.3.1 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.

1.3.2 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

2.1 Funciones anónimas

((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 (M N) (+ M N)) 2 3)                ---> 5
(MAPCAR #'(LAMBDA (N) (* 2 N)) '(1 2 3))    ---> (2 4 6)
```

2.2 Funciones con nombres

(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
```

3 Predicados

3.1 Valores lógicos

NIL
su valor es NIL y representa "lo falso". Puede escribirse como `()`.

T
su valor es T y representa "lo verdadero".

3.2 Predicados de tipos

(NULL s)
devuelve T, si **s** es NIL y NIL, si no.

(ATOM s)
Devuelve T, si **s** es un átomo y NIL, si no.

```
(ATOM 12)      --->  T
(ATOM 'ABC)    --->  T
(ATOM "A B")   --->  T
(ATOM A)       --->  ERROR
(ATOM '(A B))  --->  NIL
(ATOM NIL)     --->  T
```

(SYMBOLP s)
devuelve T, si **s** es un símbolo y NIL, si no.

```
(SYMBOLP 12)      --->  NIL
(SYMBOLP 'ABC)    --->  T
(SYMBOLP "A B")   --->  NIL
(SYMBOLP  A)      --->  ERROR
(SYMBOLP '(A B))  --->  NIL
(SYMBOLP NIL)     --->  T
```

(NUMBERP s)
devuelve T, si **s** es un número y NIL, si no.

```
(NUMBERP 123)  --->  T
(NUMBERP 'A)   --->  NIL
```

```
(SETQ A 3)     --->  3
(NUMBERP A)    --->  T
```

(CONSP s)
devuelve T, si **s** es una lista no vacía y NIL, si no.

```
(CONSP  23)     --->  NIL
(CONSP '(A B))  --->  T
(CONSP ())      --->  NIL
```

(LISTP s)
devuelve T, si **s** es una lista y NIL, si no.

```
(LISTP  23)     --->  NIL
(LISTP '(A B))  --->  T
(LISTP ())      --->  T
```

3.3 Predicados de igualdad

(EQ s1 s2)
devuelve T si **s1** y **s2** son el mismo símbolo y `NIL` en caso contrario.

```
(EQ 'LISP 'LISP)                          ---> T
(EQ 'LISP 'LISA)                          ---> NIL
(EQ (CAR '(A B C)) 'A)                    ---> T
(EQ 3 3.0)                                ---> NIL
(EQ (CONS 'A '(B C)) (CONS 'A '(B C)))    ---> NIL
```

(EQUAL s1 s2)
devuelve T, si **s1** y **s2** tienen el mismo valor y NIL, si no.

```
(EQUAL 3 (+ 1 2))  --->  T
```

3.4 Operadores lógicos

(NOT s)

devuelve T, si **s** es NIL; NIL, si no.

(OR s1...sN)

evalúa sucesivamente **s1**,..., **sN** hasta que una de dichas expresiones tenga un valor distinto de NIL. OR devuelve este valor. Si el valor de todas las expresiones es NIL, entonces OR devuelve NIL.

```
(OR NIL 2 3)  --->  2
```

(AND s1...sN)

evalúa sucesivamente **s1**,..., **sN** hasta que el valor de una de dichas expresiones sea NIL; en cuyo caso, AND devuelve NIL. Si el valor de todas las expresiones es distinto de NIL, entonces AND devuelve el valor de **sN**.

```
(AND 1 2 3)    --->  3
(AND 1 NIL 3)  --->  NIL
```

4 Estructuras de control

4.1 Constantes y variables

4.1.1 Referencias

(QUOTE s)
devuelve **s** (sin evaluar).

```
(QUOTE (+ 2 3))  --->  (+ 2 3)
```

4.1.2 Asignaciones

(SET 'simb s)
asigna, destruyendo cualquier asignación previa, el valor de la expresión **s** al símbolo **simb**. Devuelve el valor de **s**.

```
(SET 'X (1+ 5))             --->  6
X                           --->  6
(SET (IF (= 1 2) 'A 'B) 3)  --->  3
A                           --->  ERROR
B                           --->  3
```

(SETQ simb1 s1 simb2 s2 ... simbN sN)
es como SET, pero se permiten asignaciones múltiples y no se evalúan los **simbi**. Devuelve el valor de **sN**.

SETQ viene a ser como SET QUOTE.

```
(SETQ X 3 Y (+ X 2))  --->  5
Y                     --->  5
```

(MAKUNBOUND simb)
borra el valor asignado a **simb**.

```
(SETQ A 3)       --->  3
A                --->  3
(MAKUNBOUND 'A)  --->  A
A                --->  ERROR: Unbound Variable: A
```

4.2 Invocación de funciones

(APPLY fn l)

devuelve el valor de la función **fn** actuando sobre los elementos de **l** como argumentos.

```
(APPLY #'+ '(1 2 3))  --->  6
```

(FUNCALL fn s1...sN)

es lo mismo que (APPLY **fn** (LIST **s1**...**sN**)).

```
(FUNCALL #'+ 1 2 3)  --->  6
```

4.3 Funciones de evaluación

(EVAL s)

devuelve el valor de **s**.

```
(EVAL (CONS '+ '(2 3)))  --->  5
```

(PROG1 s1... sN)

evalúa sucesivamente las expresiones **s1**,..., **sN** y devuelve el valor de **s1**.

```
(PROG1 (SETQ A 2)(SETQ B (+ A 3))(+ A B))  --->  2
```

(PROG2 s1...sN)

evalúa sucesivamente las expresiones **s1**,..., **sN** y devuelve el valor de **s2**.

```
(PROG2 (SETQ A 2)(SETQ B (+ A 3))(+ A B))  --->  5
```

(PROGN s1...sN)

evalúa sucesivamente las expresiones **s1**,..., **sN** y devuelve el valor de **sN**.

```
(PROGN (SETQ A 2)(SETQ B (+ A 3))(+ A B))  --->  7
```

4.4 Variables locales

(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
```

(LET* ((var1 val1)...(varN valN)) s1...sM)

es lo mismo que LET, pero asociando secuencialmente los valores a las variables.

```
(LET* ((A 2)(B (+ 1 A))) (+ A B))  --->  5
```

(PROGV lista-de-simbolos lista-de-valores s1...sN)

asocia a los símbolos de `lista-de-simbolos` los valores de `lista-de-valores`, evalúa las expresiones **s1**, ..., **sN** y devuelve el valor de la expresion **sN**.

```
(PROGV '(A B C) '(1 2 3) (+ A B C))  --->  6
```

4.5 Condicionales

(IF s s1 s2)

devuelve **s1**, si **s** no es NIL y **s2**, en caso contrario.

```
(IF T   1 2)  --->  1
(IF NIL 1 2)  --->  2
```

(IFN s s1 s2)

es lo mismo que (IF (NOT **s**) **s1 s2**).

```
(IFN T   1 2)    --->  2
(IFN NIL 1 2)    --->  1
```

(WHEN s s1...sN)

si el valor de **s** no es NIL, evalúa **s1**,..., **sN** y devuelve el valor de **sN**;
si el valor de s es NIL, devuelve NIL.

```
(WHEN T   1 2 3)  --->  3
(WHEN NIL 1 2 3)  --->  NIL
```

(UNLESS s s1...sN)

si el valor de **s** es NIL, evalúa **s1**,..., **sN** y devuelve el valor de **sN**; si el valor de s no es NIL, devuelve NIL.

```
(UNLESS T   1 2 3)  --->  NIL
(UNLESS NIL 1 2 3)  --->  3
```

(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 no NIL, evalúa las correspondientes expresiones **si** y devuelve el valor de la última.

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

Si ninguna condición se satisface, se devuelve NIL.

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
```

(CASE s l1...lN)

Cada **li** es de la forma

(si s'1...s'M)

s es una expresión que se evalúa, y su valor se toma como indicador. Se selecciona una **li**, y se ejecutan secuencialmente las expresiones **s'1**,..., **s'M** correspondientes, devolviéndose la última evaluación. El criterio de selección es el siguiente:

- Si **si** es un átomo, selecciona **li** si **s** = **si**.
- Si **si** es lista, selecciona **li** si (MEMBER **s si**).
- Si ninguna se selecciona, devuelve NIL

```
* (DEFUN PP (N)
    (CASE N
          (1 'PRIMERO)
          (2 'SEGUNDO)
          ((3 4 5) 'POSTERIOR) ) )
PP
* (PP 2)
SEGUNDO
* (PP 4)
POSTERIOR
* (PP 8)
NIL
```

4.6 Iteración

4.6.1 Iteración indefinida

(LOOP s1...sN)

evalúa sucesivamente **s1**,..., **sN** hasta encontrar un RETURN.

```
* (PROGN (SETQ S '(A B))
    (LOOP (IF S (PRINT (POP S))
          (RETURN 'FIN) )))
A
B
NIL
```

4.6.2 Iteración general

(DO lv lc exp1...expM)

La estructura de la función DO es la siguiente:

```
(DO ((var-1 val-in-1 val-inc-1)
     ..............................
     (var-N val-in-N val-inc-N))
    (condicion s1...sM)
    exp1
    ......
    expM')
```

Actúa de la siguiente forma:

1. Asigna a la variable **var-1** el valor **val-in-1**,..., a la variable **var-N** el valor **val-in-N**.
2. Evalúa la condicion
   (a) Si la condicion es verdadera, evalúa las expresiones **s1**,..., **sM** y devuelve el valor de **sM**.
   (b) Si la condicion es falsa, evalúa las expresiones **exp1**,..., **expM'**; asigna a la variable **var-1** el valor **val-inc-1**,..., a la variable **var-N** el valor **val-inc-N** y vuelve a (2).

```
* (DO ((A 3 (1- A)))
       ((= A 0) 'FIN)
    (PRINT A) )
3
2
1
FIN
```

(DO* lv lc exp1...expM)

es similar a DO, salvo que los valores de las variables se asocian secuencialmente.

```
* (DO* ((A 2 (1- A))
             (B A A) )
```

```
       ((= B 0) 'FIN)
    (PRINT B) )
2
1
FIN
```

4.6.3 Iteraciones particulares

(DOTIMES (var m resultado) s1...sN)
es lo mismo que

```
(DO ((VAR 0 (1+ VAR)))
    ((= VAR M) RESULTADO)
    S1 ... SN)
```

es decir; asigna a **var** el valor 0; evalúa **s1**,..., **sN**; aumenta el valor de **var** en 1, si dicho valor es igual a **M** devuelve **resultado**; en otro caso, itera el proceso.

```
* (DOTIMES (CONTADOR 3 'FIN)
    (PRINT CONTADOR) )
0
1
2
FIN
```

(DOLIST (var l resultado) s1...sN)
es lo mismo que

```
(DO* ((LISTA-AUX L (CDR L))
      (VAR (CAR LISTA-AUX) (CAR LISTA-AUX)) )
     ((NULL LISTA-AUX) RESULTADO)
     S1 ... SN)
```

es decir; asigna a **var** el primer elemento de la lista **l**; evalúa **s1**,..., **sN**; si **l** no tiene más elementos, devuelve **resultado**; en otro caso, le asigna a **var** el siguiente elemento de **l** e itera el proceso.

```
* (DOLIST (CONTADOR '(A B C) 'FIN)
    (PRINT CONTADOR) )
A
B
C
FIN
```

4.6.4 Funciones de aplicación

(MAPC fn l)

aplica la función **fn** tomando como argumento los primeros elementos de la lista **l**, después toma como argumento los segundos elementos de la lista y así hasta que la lista se termina.

```
* (MAPC #'PRINT '(A (B C) D) )
A
(B C)
D
NIL
```

(MAPLIST fn l1...lN)

devuelve la lista con los resultados de aplicar **fn** a las **li**, luego a los CDR de las **li**, luego a los CDDR de las **li**, hasta que una se termina.

```
* (MAPLIST #'APPEND '(A (B C) D) '(X Y))
((A (B C) D X Y)((B C) D Y))
* (MAPLIST #'REVERSE '(A (B C) D))
((D (B C) A)(D (B C))(D))
```

(MAPCAR fn l1...lN)

devuelve la lista con los resultados de aplicar **fn** a los CAR de las **li**, luego a los CADR de las **li**, hasta que una se termina.

```
(MAPCAR #'ATOM '(1 (1) A))      --->  (T NIL T)
(MAPCAR #'+ '(1 2 3) '(4 5 6))  --->  (5 7 9)
```

(EVERY fn l)

va aplicando **fn** a los elementos de **l** hasta que una aplicación dé NIL, o hasta que la lista se termine; en cualquier caso, devuelve la última aplicación.

```
(EVERY #'ATOM '(1 A))    --->  T
(EVERY #'ATOM '((1) A))  --->  NIL
```

(EVERY fn l1...lN)

actúa como MAPCAR con varias listas.

```
(EVERY #'= '(1 2) '(1 2))   --->  T
(EVERY #'= '(1 2) '(1 3)))  --->  NIL
```

(SOME fn l)

va aplicando **fn** a los elementos de **l** hasta que una aplicación dé distinto de NIL o hasta que la lista se termine; en cualquier caso, devuelve la última aplicación.

```
(SOME #'CONSP '(A (B) C))       --->  T
(DEFUN F (S) (IF (CONSP S) S))  --->  F
(SOME #'F '(A (B) C))           --->  (B)
```

(SOME fn l1...lN)

actúa como MAPCAR con varias listas.

```
(SOME '= '(1 2 3) '(3 2 1))  --->  T
```

4.6.5 Iteraciones del tipo PROG

(PROG l s1...sN)

l es una lista de variables locales que toman el valor inicial NIL.

PROG evalúa secuencialmente **s1**,..., **sN**; si no se detiene con un RETURN, devuelve el valor de **sN**.

Cuando, dentro de un PROG, se encuentra un símbolo, éste no es evaluado, sino que se toma como etiqueta.

(RETURN s)

da error si no está dentro de un PROG o un LOOP; cuando lo está, devuelve el valor de **s**, y detiene el ciclo.

(GO simb)

transfiere el control a la etiqueta **simb** del PROG. Da error si se usa fuera de un PROG.

```
* (PROG (L)
    ETIQUETA
    (COND ((= (LENGTH L) 3) (RETURN L))
          (T (SETQ L (CONS 7 L))
             (GO ETIQUETA) )))
(7 7 7)
```

5 Números

Los tipos de números admitidos por GCLISP son: enteros (por ejemplo, `123`), decimales (por ejemplo, `123.456`) y flotantes (por ejemplo, `1.23F+02`). Los números enteros y decimales pueden variar desde -32767 a 32767 y los flotantes, desde -8.43F-37 a 3.37F+38.

5.1 Operaciones numéricas

(+ n1 n2 ... nN)
devuelve el valor de la suma $n1 + n2 + ... + nN$. Si $N = 0$, da 0.

```
(+)                 --->  0
(+ 3)               --->  3
(+ 3 7 5)           --->  15
(+ 32000 32000)     --->  ERROR
(+ 32000.0 32000)   --->  64000.0
```

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

(- n1 n2 ... nN)
devuelve el valor de $n1 - n2 - ... - nN$. Si $N = 1$, da $-n1$.

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

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

(ABS n)
devuelve el valor absoluto de **n**.

```
(ABS 3)      --->  3
(ABS -3.6)   --->  3.6
```

(* n1 n2...nN)
devuelve el valor del producto $n1.n2...nN$. Si $N = 0$, da 1.

```
(*)                 --->  1
(* 3)               --->  3
(* 3 7 5)           --->  105
(* 32000 32000)     --->  ERROR
(* 32000.0 32000)   --->  1.024F+09
```

(/ 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
```

(MOD n1 n2)
devuelve el resto de la división entera de **n1** por **n2**.

```
(MOD 7 2)   --->  1
```

(MAX n1 ... nN)
devuelve el mayor valor de **n1**,..., **nN**.

```
(MAX 3)             --->  3
(MAX 1 2 3 4 5 2)   --->  5
(MAX -2.3 7 0)      --->  7.0
```

(MIN n1 ... nN)
devuelve el menor valor de **n1**,..., **nN**.

```
(MIN 3)             --->  3
(MIN 1 2 3 4 5 2)   --->  1
(MIN -2.3 7 0)      --->  -2.3
```

5.2 Comparaciones numéricas

(= n1 ... nN)

devuelve T si los valores de todos los argumentos son iguales; NIL, en caso contrario.

```
(= 10 (+ 3 7))      --->  T
(= 2 2.0 (+ 1 1))   --->  T
(= 1 2 3)           --->  NIL
(= 1 2 1)           --->  NIL
```

(/= n1 ... nN)

devuelve T si los valores de todos los argumentos son distintos; NIL, en caso contrario.

```
(/= 10 (+ 3 7))      --->  NIL
(/= 2 2.0 (+ 1 1))   --->  NIL
(/= 1 2 3)           --->  T
(/= 1 2 1)           --->  NIL
```

`(>= n1 ... nN)`

devuelve T si $n1 \geq \ldots \geq nN$; NIL, en otro caso.

```
(>= 4 3 3 2)  --->  T
(>= 4 3 3 5)  --->  NIL
```

`(> n1 ... nN)`

devuelve T si $n1 > \ldots > nN$; NIL, en otro caso.

```
(> 4 3 2 1)  --->  T
(> 4 3 3 2)  --->  NIL
```

`(<= n1 ... nN)`

devuelve T si $n1 \leq \ldots \leq nN$; NIL, en otro caso.

```
(<= 2 3 3 4)  --->  T
(<= 5 3 3 4)  --->  NIL
```

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

```
(< 1 2 3 4)  --->  T
(< 1 3 3 4)  --->  NIL
```

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

(PLUSP n)
es equivalente a `(> n 0)`.

(MINUSP n)
es equivalente a `(< n 0)`.

(EVENP n)
devuelve T si **n** es par; NIL, en caso contrario.

(ODDP n)
devuelve T si **n** es impar; NIL, en caso contrario.

5.3 Las funciones trigonométricas y matemáticas

En la versión 1.01, las siguientes funciones requieren un coprocesador matemático. En la versión 1.1, no es necesario.

(SIN n)
devuelve el seno de **n** radianes.

(COS n)
devuelve el coseno de **n** radianes.

(TAN n)
devuelve la tangente de **n** radianes.

(ASIN n)
devuelve el arco seno de **n** (expresado en radianes).

(ACOS n)
devuelve el arco coseno de **n** (expresado en radianes).

(ATAN n)
devuelve el arco tangente de **n** (expresado en radianes).

(EXP n)
devuelve e^n.

(EXPT n1 n2)
devuelve $n1^{n2}$.

(LOG n)
devuelve el logaritmo neperiano de **n**.

(LOG n1 n2)
devuelve el logaritmo de **n1** en la base **n2**.

(SQRT n)
devuelve la raíz cuadrada de **n**.

6 Listas

6.1 Las funciones de búsqueda en las listas

(CAR l)

devuelve el primer elemento de **l**, si **l** no es NIL; NIL, si **l** es NIL

```
(CAR '(A B C))  --->  A
(CAR ())        --->  NIL
(CAR 'A)        --->  ERROR
```

(FIRST l)

devuelve el primer elemento de **l**. Es equivalente a CAR.

```
(FIRST '(A B C))  --->  A
(FIRST ())        --->  NIL
```

(CDR l)

devuelve la lista formada por los elementos de **l**, excepto el primero, si **l** no es NIL; NIL si **l** es NIL.

```
(CDR '(A B C))  --->  (B C)
(CDR ())        --->  NIL
(CDR 'A)        --->  ERROR
```

(REST l)

devuelve la lista formada por los elementos de **l**, excepto el primero, si **l** no es NIL; NIL si **l** es NIL. Es equivalente a CDR.

```
(REST '(A B C))  --->  (B C)
(REST ())       --->  NIL
```

(C...R l)

es una combinación de las funciones CAR y CDR hasta 4 niveles.

```
(CADR '(A B C))  --->  B
```

(SECOND l)

devuelve el segundo elemento de **l**. Es equivalente a CADR.

```
(SECOND '(A B C D))  --->  B
```

(THIRD l)

devuelve el tercer elemento de **l**. Es equivalente a CADDR.

```
(THIRD '(A B C D))  --->  C
```

(MEMBER s l)

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

(MEMBER s l :test #'predicado)

devuelve la sublista de **l** que comienza por el primer elemento que, junto a **s**, satisface el predicado, si alguno lo satisface; NIL, en otro caso.

```
(MEMBER 'X '(A X B X C))        ---> (X B X C)
(MEMBER 'X '(A (X) B))          ---> NIL
(SETQ L' ((A B) (C D)))         ---> ((A B) (C D))
(MEMBER '(C D) L)               ---> NIL
(MEMBER '(C D) L :TEST #'EQUAL) ---> ((C D))
(MEMBER 2.0 '(1 2 3))           ---> NIL
(MEMBER 2.0 '(1 2 3) :TEST #'=) ---> (2 3)
(MEMBER 2.0 '(1 2 3) :TEST #'<) ---> (3)
```

(NTHCDR n l)

devuelve el n–ésimo CDR de **l**; NIL, si **l** tiene menos de **n** elementos o **l**, si **n** <= 0. Se empieza a contar por 0.

```
(NTHCDR 1 '(A B C))  --->  (B C)
```

(NTH n l)

devuelve el n-ésimo elemento de **l**. Se empieza a contar por 0.

```
(NTH 1 '(A B C))  --->  B
```

(LAST s)

devuelve la lista formada por el último elemento de **s**, si **s** es una lista y **s**, si **s** no es una lista.

```
(LAST '(A B C))  --->  (C)
(LAST 23)        --->  23
```

(LENGTH l)

devuelve el número de elementos de **l**.

```
(LENGTH '(A (B C) D))  --->  3
```

6.2 Las funciones de construcción de listas

(CONS s l)

devuelve la lista cuyo CAR es **s** y cuyo CDR es **l**.

```
(CONS 'A '(B C))  --->  (A B C)
```

(LIST s1 s2 ... sN)

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

```
(LIST 'A 'B 'C)  --->  (A B C)
```

(LIST* s1 s2...sN l)

es lo mismo que **(CONS s1 (CONS s2 (...(CONS sN l)...)))**.

```
(LIST* 'A 'B '(C))  --->  (A B C)
```

(MAKE-LIST n)

devuelve una lista formada por **n** elementos NIL.

```
(MAKE-LIST 3)  --->  (NIL NIL NIL)
```

(MAKE-LIST n :INITIAL-ELEMENT s)

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

```
(MAKE-LIST 3 :INITIAL-ELEMENT 'Z)      --->  (Z Z Z)
(MAKE-LIST 3 :INITIAL-ELEMENT (1+ 4))  --->  (5 5 5)
```

(APPEND l1 ... lN)

devuelve una copia de la concatenación de las listas **l1**,..., **lN**.

```
(APPEND '(A B) '(X Y))        --->  (A B C X Y)
(APPEND '(A) '(B) () '(X Y))  --->  (A B C X Y)
```

(REVERSE l)

devuelve la lista **l**, con sus elementos en orden inverso.

```
(REVERSE '(A (B C) D))  --->  (D (B C) A)
```

(COPY-LIST s)

fabrica una copia de la expresión **s** y devuelve el valor de **s**. Esta función, aparentemente inútil, sirve para preservar algún dato de las modificaciones permanentes.

```
* (SETQ X '(A B C) Y  X Z (COPY-LIST X))
(A B C)
(RPLACA X 'D)
(D B C)
* Y
(D B C)
Z
(A B C)
```

(BUTLAST l n)

devuelve la lista **l** sin los **n** últimos elementos.

```
(BUTLAST '(A B C) 1)  --->  (A B)
(BUTLAST '(A B C) 2)  --->  (A)
(BUTLAST '(A B C) 3)  --->  NIL
```

(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))
```

6.3 Las funciones de modificación física

(RPLACA l s)

devuelve la lista **l**, en la que se ha reemplazado el CAR por **s** y **l** queda alterada.

```
(SETQ L '(A B))  --->  (A B)
(RPLACA L '(C))  --->  ((C) B)
L                --->  ((C) B)
```

(RPLACD l s)

devuelve la lista **l**, en la que se ha reemplazado el CDR por **s** y **l** queda alterada.

```
(SETQ L '(A B))  --->  (A B)
(RPLACD L '(C))  --->  (A C)
L                --->  (A C)
```

(NCONC l1 ... lN)

devuelve **l1**, pero modificada concatenando físicamente todas las listas **li**. Además, modifica cada lista concatenando físicamente las siguientes.

```
(SETQ L1 '(A) L2 '(B) L3 '(C))  --->  (C)
(NCONC L1 L2 L3)                --->  (A B C)
L1                              --->  (A B C)
```

```
L2                                  --->  (B C)
L3                                  --->  (C)
```

(DELETE s l)

devuelve la lista **l**, en la que se han borrado las estancias de primer nivel de la expresión **s** y **l** queda alterada.

```
(SETQ L '(A B C B A))  --->  (A B C B A)
(DELETE 'B L)          --->  (A C A)
L                      --->  (A C A)
```

6.4 Listas de asociación (A–listas)

Una A–lista (lista de asociación) es una lista de pares punteados. Tiene la siguiente estructura:

```
((CLAVE-1 . VALOR-1) ... (CLAVE-N . VALOR-N))
```

cada elemento de una A–lista es un par formado por la CLAVE (el CAR) y el VALOR (el CDR).

(ACONS CLAVE VALOR AL)

añade el elemento compuesto por la CLAVE y el VALOR al comienzo de la lista AL y devuelve la lista así formada.

```
(ACONS 'A 1 '((B . 2)(C . 3))) ---> ((A . 1)(B . 2)(C . 3))
```

(PAIRLIS L1 L2 AL)

forma una A–lista mediante las claves de L1 y los valores de L2 y se la añade a la cabeza de AL.

```
* (PAIRLIS '(A B C) '(1 2 3) ())
((A . 1) (B . 2)(C . 3))
* (PAIRLIS '(A B) '(1 2) '((C . 3)))
((A . 1)(B . 2)(C . 3))
* (PAIRLIS '(A B) '((1) (2)) ())
((A 1) (B 2))
```

(ASSOC SIMB AL)

devuelve el elemento de la A–lista AL cuya clave es SIMB.

```
(ASSOC 'B '((A . 1) (B . 2) (C . 3)))) ;--> (B . 2)
(ASSOC '(B) '((A . 1) ((B) 1) (C D)))  ;--> ((B) 1)
```

(RASSOC VAL AL)

devuelve el primer elemento de la A–lista AL cuyo valor es VAL.

```
* (RASSOC 1 '((A) (B . 1) (C D E))))
(B . 1)
* (RASSOC '(D E) '((A) ((B) 1) (C D E)))
NIL
* (RASSOC '(D E) '((A) ((B) 1) (C D E)) :TEST #'EQUAL)
(C D E)
```

(SUBLIS AL S)

devuelve una copia de la expresión S en la que todas las ocurrencias de las claves de la A–lista AL se han sustituido por sus valores.

```
* (SETQ DICCIONARIO '((2 . DOS)
                      (4 . CUATRO)
                      (+ . MAS)(= . IGUAL-A)))
((2 . DOS)(4 . CUATRO)(+ . MAS)(= . IGUAL-A))
* (SUBLIS DICCIONARIO '(2 + 2 = 4))
(DOS MAS DOS IGUAL-A CUATRO)
```

6.5 Listas de propiedades (P–listas)

Una P–lista (lista de propiedades) es una lista de la forma

```
(IND1 VAL1 ... INDN VALN)
```

donde los indicadores IND y los valores VAL son S–expresiones.
Los argumentos de las funciones sobre P–listas son:

- PL: un símbolo que se utilizará como P–lista
- IND: un símbolo que se utilizará como indicador

- VAL: una expresión que se utilizará como valor

(SETF (SYMBOL-PLIST PL) L)
asocia a PL la P–lista L y devuelve PL. Es distinto de SETQ; hay que hacerlo así para que sobre L puedan actuar las funciones sobre P–listas.

```
* (SETF (SYMBOL-PLIST 'PEPE) '(EDAD 40 HIJOS (PEPITO PEPITA)))
PEPE
```

(SYMBOL-PLIST PL)
devuelve la P–lista asociada a PL, si la hay y `NIL`, si no.

```
(SYMBOL-PLIST 'PEPE) ---> (EDAD 40 HIJOS (PEPITO PEPITA))
(SYMBOL-PLIST 'JUAN) ---> NIL
```

(GET PL IND)
devuelve el valor asociado a IND en la P–lista PL.

```
(GET 'PEPE 'EDAD)   ---> 40
(GET 'PEPE 'HIJOS)  ---> (PEPITO PEPITA)
(GET 'PEPE 'PADRES) ---> NIL
```

(SETF (GET PL IND) VAL)
si IND es un indicador de la P–lista PL, modifica su valor a VAL; si no lo es, añade al comienzo de PL el indicador IND y el valor VAL. Devuelve VAL.

```
* (SETF (GET 'PEPE 'EDAD) 41)
41
* (SYMBOL-PLIST 'PEPE)
(EDAD 41 HIJOS (PEPITO PEPITA))
* (SETF (GET 'PEPE 'MADRE) 'ANA)
ANA
* (SYMBOL-PLIST 'PEPE)
(MADRE ANA EDAD 41 HIJOS (PEPITO PEPITA))
```

(REMPROP PL IND)
borra de la P–lista PL, el indicador IND y su valor asociado y devuelve T.

```
(REMPROP 'PEPE 'EDAD) ---> T
(SYMBOL-PLIST 'PEPE)  ---> (MADRE ANA HIJOS (PEPITO PEPITA))
```

7 Funciones de lectura y escritura

7.1 Funciones de lectura

(READ)

lee un objeto del canal de entrada y devuelve dicho objeto.

```
* (SETQ L (CDR (READ)))
(B C)
* L
(B C)
```

(READ-LINE)

lee una línea del canal de entrada y devuelve dicho objeto.

7.2 Variables de escritura

PRINT-BASE

indica la base en que se escriben los números. Por defecto es 10. Puede variar de 2 a 36.

```
*PRINT-BASE*            --->  10
8                       --->  8
(SETQ *PRINT-BASE* 2)   --->  10
8                       --->  1000
(SETQ *PRINT-BASE* 10)  --->  10
8                       --->  8
```

PRINT-LENGTH

indica el número de elementos que se escriben.

```
*PRINT-LENGTH*            --->  10
'(1 2 3 4 5 6)            --->  (1 2 3 4 5 6)
(SETQ *PRINT-LENGTH* 3)   --->  3
'(1 2 3 4 5 6)            --->  (1 2 3 ...)
```

PRINT-LEVEL

indica el número de niveles de paréntesis que se escriben.

```
(SETQ *PRINT-LENGTH* 30)  --->  30
*PRINT-LEVEL*             --->  4
'(A B (C D (E F) J) K)    --->  '(A B (C D (E F) J) K)
(SETQ *PRINT-LEVEL* 2)    --->  2
'(A B (C D (E F) J) K)    --->  (A B (C D # J) K)
```

7.3 Funciones de escritura

(PRINT s)

escribe en una nueva línea el valor de **s** y devuelve dicho valor.

```
* (PRINT (+ 2 3))
5
5
```

(PRIN1 s)

escribe el valor de **s**, usando caracteres de control.

```
* (PRIN1 "A")
"A"
"A"
```

(PRINC S)

escribe el valor de **s**, sin usar caracteres de control.

```
* (PRINC "A")
A
"A"
```

(TERPRI)

escribe una línea en blanco.

(FORMAT T CAD ARG1 ... ARGN)

escribe la cadena en CAD en la pantalla y devuelve NIL. CAD puede contener caracteres de control. Alguno de ellos son los siguientes:

- ~% nueva línea

- ~D si el argumento es un número decimal
- ~A si el argumento es un carácter ASCII
- ~B si el argumento es un número binario
- ~O si el argumento es un número octal
- ~X si el argumento es un número hexadecimal

```
* (FORMAT T "~%LINEA 1 ~%LINEA 2)
LINEA 1
LINEA 2
NIL
* (FORMAT T "~%EL CUADRADO DE ~D ES ~D" 3 (* 3 3))
EL CUADRADO DE 3 ES 9
NIL
* (SETQ L '(A B C))
(A B D)
* (FORMAT T
          "~%LA LONGITUD DE LA LISTA ~A ES ~D"
          L (LENGTH L))
LA LONGITUD DE LA LISTA '(A B C) ES 3
NIL
* (FORMAT T
          "~%10 EN BINARIO ES ~B Y EN OCTAL ES ~O"
          10 10)
10 EN BINARIO ES 1010 Y EN OCTAL ES 12
NIL
```

8 Ficheros

8.1 Funciones sobre ficheros

(DRIBBLE s)
escribe lo que sale en pantalla en el fichero de nombre **s**.

(DRIBBLE)
cierra el fichero abierto por DRIBBLE.

(SETQ SIMB (OPEN NOMBRE-FICHERO :DIRECTION :CLAVE))

abre un fichero.

- SIMB: es el nombre asignado al canal que va a estar unido al fichero que se está abriendo.
- NOMBRE-FICHERO: es el camino hacia el fichero, incluyendo su nombre.
- DIRECTION: es un modificador para indicar que el canal que se va a construir es de entrada, salida o ambas cosas. El valor de :CLAVE es :INPUT, :OUTPUT o :IO, respectivamente.

(CLOSE SIMB)
cierra el canal nombrado por SIMB.

```
* (SETQ FICH1 (OPEN "FICHERO.DAT" :DIRECTION :OUTPUT))
#<CLOSURE 4368:79BF>
* (FORMAT FICH1 "LINEA 1~%LINEA 2")
NIL
* (FORMAT FICH1 "~%LINEA 3")
NIL
* (CLOSE FICH1)
NIL
* (SETQ A (OPEN "FICHERO.DAT" :DIRECTION :INPUT))
#<CLOSURE 4368:5DF0>
* (READ-LINE A)
"LINEA 1"
NIL
* (READ-LINE A)
"LINEA 2"
NIL
* (READ-LINE A)
```

```
"LINEA 3"
T
```

(WITH-OPEN-FILE (SIMB NOMBRE-FICHERO :DIRECTION :CLAVE) S1...SN

abre el fichero NOMBRE-FICHERO, asocia el nombre del canal al símbolo SIMB, evalúa las expresiones S1,..., SN y devuelve el valor de SN.

Si la :CLAVE es :INPUT, las expresiones SI pueden ser de lectura como (READ SIMB)

Si la :CLAVE es :OUTPUT, las expresiones SI pueden ser de escritura como:
(FORMAT SIMB S), (PRINT S SIMB), (PRINC S SIMB) o (TERPRI SIMB).

```
* (WITH-OPEN-FILE (FICH1 "FICHERO.DAT" :DIRECTION :OUTPUT)
     (FORMAT FICH1 "UNO ~%DOS ~%TRES") )
NIL
* (WITH-OPEN-FILE (A "FICHERO.DAT" :DIRECTION :INPUT)
     (PRINT (READ A))
     (PRINT (READ A))
     (PRINT (READ A))
     'FIN )
UNO
DOS
TRES
FIN
```

8.2 La función LOAD

(LOAD nombre)

carga el fichero nombre.lsp

9 Ayudas

9.1 Funciones de ayuda

9.1.1 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.

```
* (DEFUN FACTORIAL (N)
    (COND ((<= N 1) 1)
          (T (* (FACTORIAL (1- N)) N))))
FACTORIAL
* (TRACE FACTORIAL)
; Autoloading definition of TRACE.
T
* (FACTORIAL 3)Entering: FACTORIAL, Argument list: (3)
 Entering: FACTORIAL, Argument list: (2)
  Entering: FACTORIAL, Argument list: (1)
  Exiting: FACTORIAL, Value: 1
 Exiting: FACTORIAL, Value: 2
Exiting: FACTORIAL, Value: 6

6
* (UNTRACE)
(FACTORIAL)
* (FACTORIAL 3)
6
```

9.1.2 La ejecución paso–a–paso

(STEP s)
evalúa la expresión **s** paso a paso, de forma que se puede controlar el proceso a voluntad.

Esta función se explica por sí misma, pues tecleando ? se obtiene el menú.

```
* (DEFUN FACTORIAL (N)
    (COND ((<= N 1) 1)
          (T (* (FACTORIAL (1- N)) N))))
FACTORIAL
* (STEP (FACTORIAL 2))
 (FACTORIAL 2)
  2
   2 = 2
  (COND ((<= N 1) 1) (T (* (FACTORIAL #) N)))
   (<= N 1)
    N
     N = 2
    1
     1 = 1
    (<= N 1) = NIL
    T
     T = T
    (* (FACTORIAL (1- N)) N)
     (FACTORIAL (1- N))
      (1- N)
       N
        N = 2
       (1- N) = 1
      (COND ((<= N 1) 1) (T (* (FACTORIAL #) N)))
       (<= N 1)
        N
         N = 1
        1
         1 = 1
       (<= N 1) = T
      1
       1 = 1
      (COND ((<= N 1) 1) (T (* (FACTORIAL #) N))) = 1
     (FACTORIAL (1- N)) = 1
    N
     N = 2
    (* (FACTORIAL (1- N)) N) = 2
   (COND ((<= N 1) 1) (T (* (FACTORIAL #) N))) = 2
```

```
   (FACTORIAL 2) = 2
2
```

9.1.3 Descripción de símbolos

(DESCRIBE s)

devuelve información sobre el objeto **s**.

```
* (DESCRIBE '-)
- is a symbol.
   Its local value is: (DESCRIBE (QUOTE -)).
   Its function definition is: #<COMPILED FUNCTION 150B:9FA9>
   Its property list is empty.

NIL
* (DESCRIBE "ABC")
"ABC" is a string of length 3.

NIL
* (SETQ A 23)
23
* (DESCRIBE 'A)
A is a symbol.
   Its global value is: 23.
   Its function definition is unbound.
   Its property list is empty.

NIL
* (DEFUN A (N) (1+ N))
A
* (DESCRIBE 'A)
A is a symbol.
   Its global value is: 23.
   Its function definition is: (LAMBDA (N) (1+ N)).
   Its property list is empty.

NIL
* (SETF (SYMBOL-PLIST 'A) '(X 1 Y 2))
A
* (DESCRIBE 'A)
```

```
A is a symbol.
   Its global value is: 23.
   Its function definition is: (LAMBDA (N) (1+ N)).
   Its property list contains:
    Property: X, Value: 1
    Property: Y, Value: 2

NIL
```

(SYMBOL-NAME simb)
devuelve el nombre del símbolo **simb**.

(SYMBOL-VALUE simb)
devuelve el valor asociado al símbolo **simb**.

(SYMBOL-FUNCTION simb)
devuelve la función asociada al símbolo **simb**.

(SYMBOL-PLIST simb)
devuelve la P–lista asociada al símbolo **simb**.

```
* (DESCRIBE 'A)
A is a symbol.
   Its global value is: 23.
   Its function definition is: (LAMBDA (N) (1+ N)).
   Its property list contains:
    Property: X, Value: 1
    Property: Y, Value: 2

NIL
* (SYMBOL-NAME 'A)
"A"
* (SYMBOL-VALUE 'A)
23
* (SYMBOL-FUNCTION 'A)
(LAMBDA (N) (1+ N))
* (SYMBOL-PLIST 'A)
(X 1 Y 2)
```

9.2 Macro–caracteres

(
: indica el comienzo de una lista.

)
: indica el final de una lista.

's
: es lo mismo que **(QUOTE s)**.

;
: indica el comienzo de un comentario hasta el fin de línea.

```
(DEFUN FACTORIAL (N)                    ;factorial de n
  (COND ((<= N 1) 1)                    ;base de recursion
        (T (* N (FACTORIAL (1- N)))))   ;paso de recursion
```

"+
: indica el comienzo o final de una cadena de caracteres.

#'s
: análogo a **'s** cuando s representa una función.

#|+
: indica el comienzo de un comentario.

|+#
: indica el final de un comentario.

9.3 Teclas definidas

Alt-A
: pide una cadena y devuelve una lista de los objetos que contienen dicha subcadena.

```
* <Alt-A>
* Apropos string: car
; Autoloading definition of APROPOS.
CAR - function
SETF-CAR - function
MAPCAR - function
```

Alt-D
pide una cadena y devuelve la documentación que posee sobre dicho objeto.

```
* <Alt-D>
* Documentation for which function: CAR
; Autoloading definition of DOC.
CAR is a FUNCTION.
(CAR list) -> FIRST-ELEMENT

This function returns the first element (i.e., the 'car') of
LIST.

LIST must be either a CONS or NIL (i.e., it must be of type
LIST). If it is a CONS, the 'car' (i.e., the first component)
is returned; otherwise, if it is NIL, NIL is returned.
```

Alt-H
llama a la ayuda.

Alt-K
devuelve la lista de macro caracteres definidos.

Control-C
vuelve al nivel superior.

Control-D
vuelve temporalmente al DOS. Para continuar la sesión, pulsar EXIT.

Control-E
llama al editor GMACS.

Control-L
limpia la pantalla.

Control-Pausa
detiene la ejecución.

9.4 Funciones sobre el sistema

(EXIT)
termina la sesión Lisp.

(TIME s)

devuelve el tiempo usado para evaluar la expresión **s**.

```
* (TIME (DOTIMES (CONT 10000) (+ 2 3)))
Evaluating: (DOTIMES (CONT 10000) (+ 2 3))
Elapsed time: 0:01.59

NIL
```

(DOS cadena)

ejecuta el comando **cadena** del sistema operativo DOS.

Bibliografía

1. ALLEN, J. *Anatomy of LISP*. MacGraw-Hill, New York, 1978.
2. BERK, A.A *LISP: The Language of Artificial Intelligence.* Collins, London, 1985.
3. CHAILLOUX, J. *LE_LISP de l'INRIA Version 15.21 (Le Manuel de référence)* INRIA, 1987.
4. CHAILLOUX, J.; DEVIN, M.; SERLET, B. *LE_LISP de l'INRIA (La Bibliotque initiale)* INRIA, 1984.
5. FARRENY, H. *Exercices Programmés d'Intelligence Artificielle* Masson, Paris, 1987.
6. FARRENY, H. *Introducción al LISP (El lenguaje básico para la Inteligencia Artificial)* Masson, Barcelona, 1986.
7. FARRENY, H. *Programmer en LISP* Masson, Paris, 1984.
8. FODERARO, J.K.; SKLOWER, K.L *FRANZ LISP Manual* Univ. of California, Berkeley, Ca., September 1981.
9. HASEMER, T. *LISP (une introduction à la programmation sur micro-ordinateur)* InterEditions, Paris, 1984.
10. HOLTZ, F. *LISP, the Language of Artificial Intelligence* TAB Books, 1985.
11. MAURER, W.D. *The Programmer's Introduction to LISP* Macdonald, London, 1972.
12. MILNER, W.L. *Common Lisp: a tutorial* Prentice Hall, 1988.
13. QUEINNEC, C. *LISP: Langage d'un autre type* Eyrolles, Paris, 1982.
14. QUEINNEC, C. *LISP: Mode d'emploi* Eyrolles, Paris, 1984.
15. QUEINNEC, C. *Programación en LISP* Paraninfo, Madrid, 1987.
16. RIBBENS, D. *Programmation non numérique: LISP 1.5* Dunod, Paris, 1969.
17. ROY, J.P.; KIREMITDJIAN, G. *Lire LISP* CEDIC, Paris, 1985.
18. SIKLOSSY, L. *Let's Talk LISP* Prentice-Hall, London, 1976.

19. STEELE, G.L. *Common LISP: The Language* Digital Press, 1984.

20. TOURETZKY, D.S. *LISP: Introducción al cálculo simbólico* Díaz de Santos, Madrid, 1986.

21. WERTZ, H. *LISP, Introducción a la programación* Masson, Barcelona, 1986.

22. WERTZ, H. *LISP, une introduction à la programmation* Masson, Paris, 1985.

23. WILENSKY, R. *LISPcraft* Norton, New York, 1984.

24. WINSTON, P.H.; HORN, B.K.P. *LISP (3 edition)* Addison Wesley, 1988.

Indice