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

Curso de programación en Lisp

José A. Alonso Jiménez
(jalonso@us.es)

Sevilla, 1991

Contenido

Capítulo 1

El cálculo aritmético

1.1 Los números y sus operaciones

1.1.1 Ejemplo: Primera sesión Lisp:

```
C> GCLISP
* 5

5
*
```

1.1.2 Nota: El valor de un número es dicho número.
1.1.3 Ejemplo: Segunda sesión Lisp:

```
* (* 3 5)      ; multiplica 3 por 5
15
* (* 2 3 4)    ; multiplica 2 por 3 y por 4
24
* (/ 20 5)     ; 20 dividido por 5
4.0
* (/ 5 20)
0.25
* (/ 24 3 2)   ; (24/3) /2
4.0
* (/ 0.5)      ; 1/0.5
2.0
```

1.1.4 Notas:

1. Los números pueden combinarse entre sí mediante operaciones aritméticas usando la notación prefija.

2. Para escribir un comentario se utiliza ; y el resto de la línea es ignorado por el intérprete.

3. La multiplicación se representa por `*` y la división por `/`. Pueden tener un número arbitrario de argumentos.

1.1.5 Ejemplo: Tercera sesión Lisp:

```
* (* (- 1 2) (+ 4 5))
-9
* (+ (+ 15 5) (- 100 45))
75
```

1.1.6 Notas:

1. La suma y la resta se representan por `+` y `-`, respectivamente.

2. Al efectuar una operación, LISP evalúa primero sus argumentos.

1.1.7 Ejercicio: Calcular

$$\frac{2(4-1)6}{18} + (8-6)7$$

Solución:

```
* (+ (* 2 (- 4 1) (/ 6 18)) (* (- 8 6) 7))
16.0
* (+ (* 2
        (- 4 1)
        (/ 6 18))
     (* (- 8 6)
        7))
16.0
```

1.1.8 Notas:

1. El número de niveles de operaciones puede ser elevado.

2. LISP no devuelve el resultado hasta que se cierran todos los paréntesis.

3. Conviene "formatear" las expresiones LISP para resaltar su estructura lógica.

1.1.9 Ejemplo: Fin de sesión Lisp:

```
* (EXIT)
C>
```

1.1.10 Nota: Para entrar al LISP pulsar GCLISP, y para salir (EXIT).

1.1.11 Definición: Funciones 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
```

1.1.12 Nota: También están definidas las funciones trigonométricas y exponenciales. No hay diferencia con los otros lenguajes.

1.2 Nombrar los objetos de cálculo

1.2.1 Ejemplo: Sesión con asignaciones:

```
* (SETQ PRECIO 80)   ; PRECIO <--- 80
80
* (SETQ IVA 10)      ; IVA <--- 10
10
* PRECIO
```

```
80
* (/ (* PRECIO IVA) 100)
8.0
* (+ 1 2 X 3)
ERROR:
Unbound variable: X
1> <Control C>
Top-Level
* (SETQ X (* 2 5))
10
* (+ 1 2 X 3 )
16
* (SETQ X 1 Y 2 Z (+ X Y))
3
* (+ X Y Z)
6
```

1.2.2 Definición: La función SETQ:

(SETQ SIMB-1 EXP-1 ... SIMB-N EXP-N) asigna al símbolo SIMB-1 el valor de la expresión EXP-1,..., a SIMB-N el valor de EXP-N y devuelve el valor de EXP-N.

1.2.3 Nota: La función SETQ modifica el entorno de cálculo.

1.2.4 Ejercicio: Calcular el valor de C después de la evaluación de la expresión

```
(SETQ A 3 C (+ (SETQ B 4) (+ A B)))
```

Solución: 11.

1.2.5 Definición: Las funciones INCF y DECF:

```
(INCF s n) = (SETQ s (+ s n)).
(INCF s)   = (INCF s 1).
(DECF s n) = (SETQ s (- s n)).
(DECF s)   = (DECF s 1).
```

1.2.6 Ejemplo:

```
(SETQ X 3) ---> 3
(INCF X)   ---> 4
X          ---> 4
(DECF X 3) ---> 1
X          ---> 1
```

1.3 Definición de nuevas funciones

1.3.1 Ejemplo: Definición de la función CUBO:

```
* (DEFUN CUBO (N)
    (* N N N))
CUBO
* (CUBO 2)
= 8
```

1.3.2 Nota: La función primitiva para definir funciones es DEFUN. Una función está determinada por su nombre [en el ej. `CUBO`], la lista de sus parámetros [en el ej. `N`] y su cuerpo [en el ej. `(* N N N)`]

1.3.3 Ejemplo: Definición de la función SUMA-BINARIA:

```
* (DEFUN SUMA-BINARIA (X Y)
    (+ X Y))
SUMA-BINARIA
* (SUMA-BINARIA 2 3)
5
* (SETQ X -1)
-1
* (SUMA-BINARIA 2 3)
5
* X
-1
* (+ (SUMA-BINARIA 2 3) X)
4
* (* (SUMA-BINARIA (+ 50 50) 1) 3)
303
```

1.3.4 Ejercicio: Escribir una función P2 que calcule el valor del polinomio $ax^2 + bx + c$. Por ejemplo,

```
(P2 4 3 2 2)   ---> 24
```

Solución:

```
(DEFUN P2 (A B C X)
  (+ (* A X X) (* B X) C))
```

1.3.5 Nota: La lista de parámetros de una función puede ser vacía. Por ejemplo,

```
* (DEFUN NUMERO ()
    (SETQ X (1+ X)))
NUMERO
* (SETQ X 0)                                        ; INICIALIZA X
0
* (+ (NUMERO) (NUMERO) (NUMERO))                    ; (+ 1 2 3)
6
* X
3
```

1.3.6 Nota: La función NUMERO tiene "efecto de borde" porque altera el entorno de cálculo. Al escribir funciones conviene evitar efectos de borde.

1.3.7 Nota: Un problema puede resolverse de forma "descendente" (descomponiéndolo en otros más simples) o de forma "ascendente" (desarrollar utilidades y combinarlas). Se aconseja la forma descendente.

1.3.8 Ejemplo: (de programación descendente) Definir la función

```
(MEDIA-CUADRADO X Y)
```

que devuelva la media de los cuadrados de X e Y.

```
* (DEFUN MEDIA-CUADRADO (X Y)
    (MEDIA (CUADRADO X) (CUADRADO Y)))
MEDIA-CUADRADO
* (DEFUN CUADRADO (X)
    (* X X))
CUADRADO
* (CUADRADO 3)                                      ; TEST DE CUADRADO
9
* (DEFUN MEDIA (X Y)
    (/ (+ X Y) 2))
MEDIA
* (MEDIA 4 8)                                       ; TEST DE MEDIA
6.0
* (MEDIA-CUADRADO 2 4)
10.0
```

1.3.9 Ejercicio: Definir CUADRADO-MEDIA tal que (CUADRADO-MEDIA X Y) sea el cuadrado de la media de X e Y. Por ejemplo,

```
(CUADRADO-MEDIA 2 4) ---> 9.0
```

Solución:

```
(DEFUN CUADRADO-MEDIA (X Y)
  (CUADRADO (MEDIA X Y)))
```

1.4 Variables globales y locales

1.4.1 Definición: La función LET:

(LET ((VAR1 VAL1) ... (VARN VALN)) S1 ...SM) asocia a la variable VAR1 el valor VAL1,..., a la variable VARN el valor VALN, calcula las expresiones S1,..., SM, reconstruye el entorno y devuelve el valor de SM.

1.4.2 Ejemplo:

```
(SETQ X 1)                    ---> 1
(LET ((X 2) (Y 3)) (+ X Y)) ---> 5
X                             ---> 1
```

1.4.3 Ejercicio: Definir una función F1 que calcule la media de los cuadrados de las raíces de la ecuación $ax^2 + bx + c = 0$. Por ejemplo,

```
(F1 1 -5 6) ---> 6.5
```

Solución:

```
(DEFUN F1 (A B C)
  (LET ((AUX1 (- B))
            (AUX2 (SQRT (- (* B B)
                           (* 4 A C))))
            (AUX3 (* 2 A)))
    (MEDIA-CUADRADO  (/ (+ AUX1 AUX2) AUX3)
                     (/ (- AUX1 AUX2) AUX3))))
```

1.4.4 Nota: Una función puede tener el mismo nombre que una variable.

1.4.5 Ejemplo:

```
* (SETQ DOS 2)
2
* (DEFUN DOS ()          ; UNA FUNCION QUE DEVUELVE 3.
    3)
DOS
* (+ DOS (DOS))
5
```

1.4.6 Ejercicio: Escribir una función P4 que calcule el valor del polinomio $3x^4 + 7x^2 + 4$ utilizando la función P2.
Solución:

```
(DEFUN P4 (X)
  (P2 3 7 4 (* X X)))
```

Capítulo 2

El cálculo simbólico

2.1 La función QUOTE sobre símbolos

2.1.1 Definición: Las funciones QUOTE y ':

(QUOTE SIMBOLO) devuelve SIMBOLO sin evaluar.

'SIMBOLO es una abreviatura de (QUOTE SIMBOLO).

2.1.2 Ejemplo: De usos de QUOTE:

```
* A

ERROR:
Unbound variable: A
1> <Control C>
Top-Level
* (SETQ A 3)
3
* (QUOTE A)
A
* 'A
A
```

2.1.3 Ejemplo: De uso de QUOTE y SETQ:

```
(SETQ HOMBRE 'SOCRATES) ---> SOCRATES
HOMBRE                  ---> SOCRATES
(SETQ SOCRATES 1)       ---> 1
HOMBRE                  ---> SOCRATES
(SETQ MORTAL HOMBRE)    ---> SOCRATES
MORTAL                  ---> SOCRATES
```

2.1.4 Definición: La función `SET`:

(SET EXP1 EXP2) asigna el valor de la expresión EXP2 al valor de la expresión EXP1 (que ha de ser una variable) y devuelve el valor de la expresión EXP2.

2.1.5 Ejemplo: (continuación del anterior)

```
(SET 'HOMBRE 'PLATON)  ---> PLATON
HOMBRE                 ---> PLATON
MORTAL                 ---> SOCRATES
(SET HOMBRE 'SOCRATES) ---> SOCRATES
HOMBRE                 ---> PLATON
PLATON                 ---> SOCRATES
```

2.1.6 Nota: (SETQ SIMB EXP) es lo mismo que (SET 'SIMB EXP)

2.2 Las expresiones en Lisp

2.2.1 Definición: Los símbolos son cadenas continuas de caracteres (conteniendo al menos un carácter no–numérico). Sirven para nombrar variables y funciones. Ejemplos: `X, 1987A, SOCRATES, X-1`.
2.2.2 Definición: Existen números enteros (ej: `5, -7`) y reales (ej: `1.743F+23`, `12.345`)
2.2.3 Definición: Un átomo es un símbolo o un número.
2.2.4 Definición: Una lista es una sucesión, posiblemente vacía, de átomos y listas. Sintácticamente se compone de un paréntesis abierto, listas o átomos separados por huecos y un paréntesis cerrado.
2.2.5 Ejercicio: Completar el siguiente cuadro:

LISTA	NUMERO DE ELEMENTOS	SEGUNDO ELEMENTO
(1 DOS 3 CUATRO)		
(SOCRATES (- 43 1))		
((X 1) (Y 2) (Z 3))		
(- (* 2 (+ A 3)))		
(((1)))		
(DEFUN SUMA (X Y) (+ X Y))		
()		

Solución:

```
LISTA                          NUMERO DE     SEGUNDO
                               ELEMENTOS     ELEMENTO
(1 DOS 3 CUATRO)                  4          DOS
(SOCRATES (- 43 1))               2          (- 43 1)
((X 1) (Y 2) (Z 3))               3          (Y 2)
(- (* 2 (+ A 3)))                 2          (* 2 (+ A 3))
(((1)))                           1          < NO TIENE >
(DEFUN SUMA (X Y) (+ X Y))        4          SUMA
()                                0          < NO TIENE >
```

2.2.6 Definición: Una expresión es un átomo o una lista.

2.2.7 Resumen:

$$\text{expresiones} \begin{cases} \text{átomos} \begin{cases} \text{números} \\ \text{símbolos} \end{cases} \\ \text{listas} \end{cases}$$

2.2.8 Nota: Hay átomos constantes como **T**, que representa el valor verdadero y **NIL**, que representa el valor falso.

2.3 La función QUOTE sobre listas

2.3.1 Definición: La función `QUOTE`:

(QUOTE LISTA) devuelve LISTA

'LISTA es lo mismo que (QUOTE LISTA).

2.3.2 Ejemplo:

```
(A B C)          ---> ERROR:
                      Undefined function: A
                      while evaluating: (A B C)
                      1> <Control C>
(QUOTE (A B C)) ---> (A B C)
'(A B C)         ---> (A B C)
(+ 2 3)          ---> 5
'(+ 2 3)         ---> (+ 2 3)
```

2.4 Funciones de búsqueda en listas

2.4.1.1 Definición: Las funciones CAR y CDR:

(CAR L) devuelve el primer elemento de L, si L es una lista no vacía y NIL, en otro caso.

(CDR L) devuelve la lista formada por los elementos de L, excepto el primero, si L es una lista no vacía y NIL, en otro caso.

2.4.1.2 Ejemplo:

```
(CAR '(A B C))                                  ---> A
(CAR '((D E) F (G H)))                          ---> (D E)
(SETQ L '(10 20))                               ---> (10 20)
(CAR L)                                         ---> 10
(CAR (SETQ M '(+ 1 2)))                         ---> +
(CDR ' (A B C))                                 ---> (B C)
(CDR '((D ((E F))) G (H I)))                    ---> (G (H I))
(CDR L)                                         ---> (20)
(CDR M)                                         ---> (1 2)
(CAR (CDR M))                                   ---> 1
(CAR (CDR (CDR M)))                             ---> 2
(CDR (CAR (CDR (CAR '((D (E F)) G (H I))))))  ---> (F)
```

2.4.1.3 Ejercicio: Definir las funciones `SEGUNDO` y `TERCERO` que devuelven el segundo o tercer elemento de una lista.

```
(SEGUNDO '(A B C D)) ---> B
(TERCERO '(A B C D)) ---> C
```

Solución:

```
(DEFUN SEGUNDO (L)
  (CAR (CDR L)))

(DEFUN TERCERO (L)
  (CAR (CDR (CDR L))))
```

2.4.1.4 Definición: La función (C...R L):

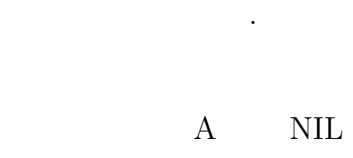
(C...R L) es una combinación de `CAR` y `CDR` hasta 4 niveles.

2.4.1.5 Ejemplo:

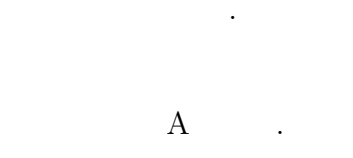
```
(SETQ M '(A (B C) D E)) ---> (A (B C) D E)
(CADR M)                ---> (B C)
(CADDR M)               ---> D
```

2.4.1.6 Nota: Una forma de representar las listas es mediante un árbol cuya raíz es el símbolo ".", su hijo izquierdo es el CAR, su hijo derecho es el CDR y sus hojas son átomos. Por ejemplo,

- la lista (A) se representa por



- la lista (A B) se representa por



- la lista ((A)) se representa por

2.4.1.7 Ejercicio: Escribir los árboles binarios de la siguientes listas

```
((A B) C (D E))
(DEFUN SUMA (X Y) (+ X Y))
```

y encontrar los `CADDR` y `CADADDR` de dichas listas.

2.4.1.8 Nota: `CAR` y `CDR` denotan "Content of Address Register" y "Content of Decrement Register", respectivamente.

2.4.1.9 Definición: Las funciones NTH, NTHCDR, LAST y LENGTH:

(NTH N L) devuelve el N-ésimo elemento de L, empezando a contar por 0.

(NTHCDR N L) devuelve el N-ésimo CDR de L, empezando a contar por 0.

(LAST L) devuelve la lista formada por el último elemento de L.

(LENGTH L) devuelve el número de elementos de L.

2.4.1.10 Ejemplo:

```
(SETQ L '(A (B C) D E))      ---> (A (B C) D E)
(NTH 0 L)                    ---> A
(NTH 2 L)                    ---> D
(NTH 8 L)                    ---> NIL
(NTHCDR 0 L)                 ---> (A (B C) D E)
(NTHCDR 2 L)                 ---> (D E)
(NTHCDR 8 L)                 ---> NIL
(LAST L)                     ---> (E)
(LENGTH L)                   ---> 4
```

2.4.1.11 Ejercicio: Sea L la lista

```
(DEFUN SUMA (X Y) (+ X Y))
```

Calcular el NTH 2, el NTHCDR 2, el LAST y el LENGTH de L.

Solución:

```
(SETQ L '(DEFUN SUMA (X Y) (+ X Y)))
(NTH 2 L)    ---> (X Y)
(NTHCDR 2 L) ---> ((X Y) (+ X Y))
(LAST L)     ---> ((+ X Y))
(LENGTH L)   ---> 4
```

2.4.1.12 Definición: Las funciones FIRST y REST son sinónimos (en Commom Lisp) de CAR y CDR.

2.4.1.13 Ejercicio: Diseñar expresiones para obtener dinero de:

```
(AGUA PAPEL DINERO TIERRA)
((AGUA PAPEL) DINERO TIERRA)
((AGUA PAPEL) (DINERO TIERRA))
(AGUA (PAPEL DINERO) TIERRA)
((AGUA PAPEL) ((DINERO) TIERRA))
```

Solución:

```
(CADDR '(AGUA PAPEL DINERO TIERRA))
(CADR '((AGUA PAPEL) DINERO TIERRA))
(CAR (CADDR '((AGUA PAPEL) (DINERO TIERRA))))
(CADDR (CADDR '(AGUA (PAPEL DINERO) TIERRA)))
(CAAR (CADDR '((AGUA PAPEL) ((DINERO) TIERRA))))
```

2.4.1.14 Ejercicio: Definir la función ULTIMO que de el último elemento de una lista. Por ejemplo,

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

Solución:

```
(DEFUN ULTIMO (L)
  (CAR (LAST L)))
```

2.4.1.15 Ejercicio: Definir la función PENULTIMO que devuelva el penúltimo elemento de una lista. Por ejemplo,

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

Solución:

```
(DEFUN PENULTIMO (L)
  (NTH (- (LENGTH L) 2) L))
```

2.4.1.16 Ejercicio: Dada una lista

```
(SETQ AGENDA '((PEPE SIERPES-12 "531420")
               (JUAN CUNA-5 "243568")
               (PABLO CERVANTES-45 "456345")))
```

Diseñar funciones tales que :

```
(NOMBRE N)    ---> el nombre N-esimo
(DIRECCION N) ---> la direccion N-esima
(TELEFONO N)  ---> el telefono N-esimo
```

Por ejemplo,

```
(NOMBRE 1)    ---> PEPE
(DIRECCION 2) ---> CUNA-5
(TELEFONO 3)  ---> "456345"
```

Solución:

```
(DEFUN NOMBRE (N)
  (CAR (NTH (1- N) AGENDA)))

(DEFUN DIRECCION (N)
  (CADR (NTH (1- N) AGENDA)))

(DEFUN TELEFONO (N)
  (CADDR (NTH (1- N) AGENDA)))
```

2.5 Funciones de construcción de listas

2.4.2.1 Definición: La función CONS:

(CONS EXPRESION LISTA) devuelve una lista cuyo CAR es EXPRESION y su CDR es LISTA.

2.4.2.2 Ejemplo:

```
(CONS 'A '(B C D))                          ---> (A B C D)
(CONS '(A B) '(C D))                        ---> ((A B) C D)
(CONS 'JB (CONS 0 (CONS 0 (CONS 7 NIL)))) ---> (JB 0 0 7)
(CONS NIL NIL)                              ---> (NIL)
(CONS (CAR '(A B)) (CDR '(B C)))            ---> (A C)
```

2.4.2.3 Nota: CONS no modifica el entorno de cálculo. Por ejemplo:

```
(SETQ L (CONS (- 5 3) '(ES UN NUMERO))) ---> (2 ES UN NUMERO)
(CONS '7+ L)                            ---> (7+ 2 ES UN NUMERO)
L                                       ---> (2 ES UN NUMERO)
```

2.4.2.4 Definición: La función LIST:

(LIST EXP1 ... EXPN) devuelve una lista cuyos elementos son los valores de las expresiones EXP1, ..., EXPN.

2.4.2.5 Ejemplo:

```
(LIST 'A 'B 'C)                  ---> (A B C)
(LIST 'A 1 'B 2)                 ---> (A 1 B 2)
(SETQ L (LIST 'EL 'NUMERO 3))    ---> (EL NUMERO 3)
(LIST (CAR L) 'DIGITO (CADDR L)) ---> (EL DIGITO 3)
(LIST '* (+ 2 3) '(/ X 2))       ---> (* 5 (/ X 2))
(LIST NIL)                        ---> (NIL)
```

2.4.2.6 Definición: La función APPEND:

(APPEND L1 ... LN) devuelve una copia de la concatenación de las listas L1, ... , LN.

2.4.2.7 Ejemplo:

```
(APPEND '(A B) '(C D))             ---> (A B C D)
(APPEND '(1 2) ' (3 4) '(5 6 7))  ---> (1 2 3 4 5 6 7)
(APPEND (LIST '* (+ 2 3)) '(4 5)) ---> (* 5 4 5)
(APPEND NIL '(A B))                ---> (A B)
```

2.4.2.8 Nota: Observar la diferencia de CONS, LIST, y APPEND:

```
(CONS '(A B) '(C D))   ---> ((A B) C D)
(LIST '(A B) '(C D))   ---> ((A B) (C D))
(APPEND '(A B) '(C D)) ---> (A B C D)
```

2.4.2.9 Ejercicio: Definir la función ROTAR-IZQ tal que (ROTAR-IZQ L) sea la lista L con su primer elemento colocado en último lugar. Por ejemplo,

```
(ROTAR-IZQ '(A B C D)) ---> (B C D A)
```

Solución:

```
(DEFUN ROTAR-IZQ (L)
  (APPEND (CDR L) (LIST (CAR L))))
```

2.4.2.10 Ejercicio: Definir la función PASA-1-A-3 que coloque el primer elemento de una lista en tercer lugar. Por ejemplo,

```
(PASA-1-A-3 '(A B C D)) ---> (B C A D)
```

Solución:

```
(DEFUN PASA-1-A-3 (L)
  (APPEND (LIST (CADR L) (CADDR L) (CAR L))
          (CDDDR L)))
```

2.6 Funciones de modificación física de listas

2.4.3.1 Nota: Ya hemos visto las siguientes funciones de modificación: SETQ, SET, INCR y DECR.

2.4.3.2 Definición: La función PUSH:

(PUSH EXP SIMB) es lo mismo que (SETQ SIMB (CONS EXP SIMB)).

2.4.3.3 Ejemplo:

```
(SETQ L '(Y Z)) ---> (Y Z)
(PUSH 'X L)     ---> (X Y X)
L               ---> (X Y Z)
```

2.4.3.4 Definición: La función POP:

(POP SIMB) (SIMB tiene que ser un símbolo cuyo valor sea una lista). La función devuelve el CAR de esta lista y asocia a SIMB el CDR de su antiguo valor.

2.4.3.5 Ejemplo:

```
(SETQ A '(X Y Z)) ---> (X Y Z)
(POP A)           ---> X
A                 ---> (Y Z)
```

Capítulo 3

El control

3.1 Los valores lógicos

3.1.1 Definición: Valores de verdad:

NIL y () se interpretan como falso.

T y todos los objetos distintos de NIL, se interpretan como verdad.

3.1.2 Definición: La función NULL:

(NULL EXP) devuelve T, si EXP es `()` y NIL, en caso contrario.

3.2 Funciones de comparación de números

3.2.1 Definición: Las funciones de comparación de números:

`(= 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 >= ... >= nN`; NIL, en otro caso.

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

`(> n1 ... nN)+` devuelve T si `n1 > ... > nN`; NIL, en otro caso.

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

`(<= n1 ... nN)+` devuelve T si `n1 <= ... <= nN`; NIL, en otro caso.

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

`(< n1 ... nN)+` devuelve T si `n1 < ... < 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.

3.3 Funciones de comparación de símbolos y listas

3.3.1 Definición: La función EQ:

(EQ EXP1 EXP2) devuelve T si EXP1 y EXP2 son el mismo símbolo y `NIL` en caso contrario.

3.3.2 Ejemplo:

```
(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
```

3.3.3 Definición: La función EQUAL:

(EQUAL S1 S2) devuelve T, si S1 y S2 tienen el mismo valor y `NIL`, en otro caso.

3.3.4 Ejemplo:

```
(EQUAL (CONS 'A '(B C)) '(A B C)) ---> T
(EQUAL (+ 2 3) 5)                 ---> T
```

3.3.5 Definición: La función MEMBER:

(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.

3.3.6 Ejemplo:

```
(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)
```

3.3.7 Definición: Los predicados básicos:

(ATOM S) devuelve T, si S es un átomo y `NIL`, en otro caso.

(SYMBOLP S) devuelve T, si S es un símbolo y `NIL`, en otro caso.

(NUMBERP S) devuelve T, si S es un número y `NIL`, en otro caso.

(CONSP S) devuelve T, si S es una lista no vacía y `NIL`, en otro caso.

(LISTP S) devuelve T, si S es una lista y `NIL`, en otro caso.

3.3.8 Ejemplo:

```
(SETQ N (1+ 3) S 'LISP L '(1 ES UN NUMERO)) ---> (1 ES UN NUMERO)
(ATOM N)                                    ---> T
(ATOM S)                                    ---> T
(ATOM T)                                    ---> T
(ATOM ())                                   ---> T
(ATOM L)                                    ---> NIL
(CONSP L)                                   ---> T
(CONSP ())                                  ---> NIL
(LISTP NIL)                                 ---> T
(SYMBOLP N)                                 ---> NIL
(SYMBOLP L)                                 ---> NIL
(SYMBOLP S)                                 ---> T
(SYMBOLP T)                                 ---> T
(SYMBOLP NIL)                               ---> T
(NUMBERP N)                                 ---> T
(NUMBERP (CAR L))                           ---> T
(NUMBERP S)                                 ---> NIL
```

3.4 Condicionales

3.4.1 La función IF

3.4.1.1 Definición: La función IF:

(IF S S1 S2) Si el valor de S es verdadero (i.e. una expresión distinta de NIL), devuelve el valor de S1. Si el valor de S es NIL, devuelve el valor de S2.

```
(IF T 1 2)                ---> 1
(IF NIL 1 2)              ---> 2
(IF (= (SETQ A 3) 4) 1 0) ---> 0
(IF (= A 3) 1 0)          ---> 1
(IF (= A 4) (+ A 2))      ---> NIL
(IF (/ A 4) (- A 2))      ---> 1
```

3.4.1.2 Ejercicio: Definir la función ABSOLUTO que devuelva el valor absoluto de un número (i.e. el número sin el signo). Por ejemplo,

```
(ABSOLUTO -5) ---> 5
```

Solución:

```
(DEFUN ABSOLUTO (N)
  (IF (< N 0) (- N) N))
```

3.4.2 La función COND

3.4.2.1 Definición: La función COND:

(COND L1 ...LN) Cada lista LI tiene una estructura del tipo

```
(CONDICION S1 ...  SM).
```

COND va evaluando las CONDICIONES, cuando encuentra la primera distinta de NIL, evalúa las correspondientes expresiones SI y devuelve el valor de la última. Si la condición seleccionada no va seguida de expresiones, devuelve el valor de la condición. Si todas las condiciones son NIL, devuelve NIL. Para forzar la selección, se escribe T como última condición.

3.4.2.2 Ejemplo: Redefinir ABSOLUTO usando COND.

Solución

```
(DEFUN ABSOLUTO (N)
  (COND ((> N 0) N)
        ((= N 0) 0)
        ((< N 0) (- N)) ))

(DEFUN ABSOLUTO (N)
  (COND ((> N 0) N)
        (T (- N)) ))
```

3.4.2.3 Ejemplo: Definir la función NOTA que lea un número y dé la nota correspondiente; es decir,

```
           | SUSPENSO     , si N < 5
           | APROBADO     , si 5 <= N < 7
(NOTA N) = | NOTABLE      , si 7 <= N < 9
           | SOBRESALIENTE, si 9 <= N <= 10
           | ERROR        , si N > 10

(DEFUN NOTA (N)
  (COND ((< N 5)   'SUSPENSO)
        ((< N 7)   'APROBADO)
        ((< N 9)   'NOTABLE)
        ((<= N 10) 'SOBRESALIENTE)
        (T 'ERROR)))
```

3.4.2.4 Ejercicio: Definir la función TIPO-DE que tome como argumento una expresión y devuelva como valor su tipo. (i.e. lista-no-vacía, átomo-nil, número o símbolo). Por ejemplo:

```
(TIPO-DE  '(A B))  ---> LISTA-NO-VACIA
(TIPO-DE  NIL)     ---> ATOMO-NIL
(TIPO-DE  (1+ 3))  ---> NUMERO
(TIPO-DE  T)       ---> SIMBOLO
(TIPO-DE  'A)      ---> SIMBOLO
```

Solución:

```
(DEFUN TIPO-DE (S)
  (COND ((CONSP S)     'LISTA-NO-VACIA)
        ((NULL S)      'ATOMO-NIL)
        ((NUMBERP S)   'NUMERO)
        (T             'SIMBOLO) ))
```

3.4.2.5 Nota: Una condición puede ir seguida de varias expresiones. Por ejemplo:

```
* (SETQ X T Y NIL Z '(3 4) U NIL)
NIL
* (COND (Y (SETQ U 'VAL1))
        ((CDR Z) (SETQ U 'VAL2))
        (X (SETQ Y (CAR Z))  (SETQ U 'VAL3))
        (T (SETQ U 'VAL 4)))
VAL2
* U
VAL2
* Y
NIL
```

3.4.2.6 Nota: No es necesario que a una condición le sigan expresiones. Por ejemplo:

```
* (COND ((1+ 2))
        (T 5))
3
```

3.4.2.7 Nota: La función ABSOLUTO puede definirse por

```
(DEFUN ABSOLUTO (N)
  (COND ((> N 0) N)
        (T (- N))))

(DEFUN ABSOLUTO (N)
  (COND ((> N 0) N)
        ((- N))))
```

3.4.2.8 Nota: No es necesario que la última condición sea T. Por ejemplo:

```
* (COND (( = (+ 2 3) 6) 'UNO)
       ('DOS))
DOS
* (COND ((ATOM  '(A)) 'UNO)
       ((LISTP  'A)  'DOS))
NIL
```

3.4.2.9 Ejercicio: Supongamos que hemos definido una variable `METALES` que contiene algunos metales y sus densidades mediante la expresión:

```
(SETQ METALES  '(HIERRO 7.8 COBALTO 8.9 NIQUEL 8.9))
```

y otra variable `GASES-NOBLES` mediante la expresión

```
(SETQ GASES-NOBLES '(HELIO NEON ARGON)).
```

Definir una función `AYUDA-QUIMICA` tal que al leer una expresión S devuelva:

- Una lista de la forma `(METAL DE DENSIDAD N)`, si S es un metal de la lista `METALES` y N es la densidad de S.
- `GAS-NOBLE`, si S está en la lista `GASES-NOBLES`.
- `DESCONOCIDO`, en otro caso.

Por ejemplo:

```
(AYUDA-QUIMICA  'COBALTO)  ---> (METAL DE DENSIDAD 8.9)
(AYUDA-QUIMICA  'HELIO)    ---> GAS-NOBLE
(AYUDA-QUIMICA  'ALUMINIO) ---> DESCONOCIDO
```

Solución:

```
(DEFUN AYUDA-QUIMICA (S)
  (COND ((MEMBER S METALES)
         (LIST  'METAL 'DE 'DENSIDAD (CADR (MEMBER S METALES))))
        ((MEMBER S GASES-NOBLES) 'GAS-NOBLE)
        (T 'DESCONOCIDO) ))
```

3.4.3 Las funciones AND y OR

3.4.3.1 Definición: La función AND:

(AND S1 ... SN) evalúa sucesivamente S1, ..., SN hasta que una vale NIL o no quedan más. En el primer caso, devuelve NIL; en el segundo, devuelve el valor de SN.

Ejemplo:

```
(AND (SETQ L '(A B)) (SETQ M (CDDR L)) (SETQ L '(X))) ---> NIL
M                                                      ---> NIL
L                                                      ---> (A B)
```

3.4.3.2 Ejercicio: Definir la función

```
(EMPIEZA-POR-NUMERO S)
```

de forma que devuelva T, si S es una lista que empieza por un número y NIL, en caso contrario. Por ejemplo:

```
(EMPIEZA-POR-NUMERO  '(1 A)) ---> T
(EMPIEZA-POR-NUMERO T)       ---> NIL
```

Solución:

```
(DEFUN EMPIEZA-POR-NUMERO (S)
  (AND (LISTP S)
       (NUMBERP  (CAR S)) ))
```

3.4.3.3 Definición: La función OR:

(OR S1 ... SN) evalúa sucesivamente S1, ..., SN hasta que una no vale NIL o no quedan más. En el primer caso devuelve el primer valor no NIL encontrado y en el segundo, devuelve NIL. Ejemplos:

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

3.4.4 Las funciones WHEN y UNLESS

3.4.4.1 Definición: La función WHEN:

(WHEN S S1 ... SN) Si el valor de S es verdadero, evalúa S1,..., SN y devuelve el valor de SN. Si el valor de S es NIL, devuelve NIL. Por ejemplo,

```
(WHEN (= (+ 2 3) 5) (SETQ A 1) (SETQ B 5) (+ A B)) ---> 6
(WHEN (= (+ 2 3) 6) (SETQ A 1) (SETQ B 5) (+ A B)) ---> NIL
```

3.4.4.2 Definición: La función UNLESS:

(UNLESS S S1 ... SN) es equivalente a (WHEN (NULL S) S1 ...SN). Por ejemplo,

```
(UNLESS ( = ( + 2 3) 5) (SETQ A 1) (SETQ B 5) (+ A B)) ---> NIL
(UNLESS ( = ( + 2 3) 6) (SETQ A 1) (SETQ B 5) (+ A B)) ---> 6
```

Capítulo 4

La programación recursiva

4.1 Funciones recursivas

4.1.1 Nota: La definición de una función es recursiva si en el cuerpo de la definición aparecen llamadas a la función.

4.1.2 Ejemplo: El factorial de un número n es el producto de todos los números enteros entre n y 1. Por ejemplo,

```
FACT (4) = 4.3.2.1
FACT (3) = 3.2.1
```

Por tanto,

```
FACT (4) = 4. FACT (3)
```

En general,

```
FACT (n) = n. FACT (n-1)
```

Vamos a definir la función FACT tal que (FACT N) sea el factorial de N. La función FACT está definida por

$$\texttt{FACT (N)} = \begin{cases} \texttt{1} & \text{si } \texttt{N = 1} \\ \texttt{N * FACT(N - 1)} & \text{si } \texttt{N} > 1 \end{cases}$$

```
(DEFUN FACT (N)
  (COND (( = N  1) 1)
        (T (* N (FACT (1- N))))  ))
```

El cálculo de (FACT 3) se realiza como sigue:

```
(FACT 3) = (* 3 (FACT 2)) =
         = (* 3 (* 2 (FACT 1))) =
         = (* 3 (* 2 1)) =
         = (* 3 2) =
         = 6
```

4.2 Las funciones TRACE y UNTRACE

4.2.1 Definición: Las funciones TRACE y UNTRACE:

(TRACE f1 ... fN) transforma las definiciones de las funciones f1, ..., fN para que impriman sus argumentos y valores. Devuelve T.

(UNTRACE f1 ... fN) elimina el efecto anterior y devuelve la lista (f1 ...fN).

(UNTRACE) devuelve la lista de todas las funciones con TRACE y elimina el efecto de TRACE de toda función que lo tenga.

4.2.2 Ejemplo: Supongamos que hemos cargado la anterior definición de FACT.Veamos como TRACE nos permite "rastrearla".

```
* (TRACE FACT)
(FACT)
* (FACT 3)
Entering: FACT, Argument list: (3)
 Entering: FACT, Argument list: (2)
  Entering: FACT, Argument list: (1)
  Exiting: FACT, Value: 1
 Exiting: FACT, Value: 2
Exiting: FACT, Value: 6

6
* (UNTRACE)
(FACT)
```

4.3 Aritmética entera positiva

4.3.1 Nota: El objeto de este parágrafo es "redefinir" las funciones

y predicados sobre los números enteros positivos usando solamente las funciones `1+`, `1-` y el predicado `=`.

4.3.2 Ejercicio: Definir las funciones SUC y PRED tales que

```
(SUC N)  = N + 1
(PRED N) = N - 1
```

Solución:

```
(DEFUN SUC (N) (1+ N))

(DEFUN PRED (N) (1- N))
```

4.3.3 Ejercicio: Definir los predicados IGUAL-P y DESIGUAL-P tales que

$$\texttt{(IGUAL-P N1 N2)} = \begin{cases} \texttt{T} & \text{si } \texttt{N1 = N2} \\ \texttt{NIL} & \text{en caso contrario} \end{cases}$$

$$\texttt{(DESIGUAL-P N1 N2)} = \begin{cases} \texttt{NIL} & \text{si } \texttt{N1 = N2} \\ \texttt{T} & \text{en caso contrario} \end{cases}$$

Solución:

```
(DEFUN IGUAL-P (N1  N2) (IF (= N1  N2) T NIL))

(DEFUN DESIGUAL-P (N1  N2) (IF (= N1  N2) NIL T) )
```

4.3.4 Ejercicio: Definir el predicado MENOR-P tal que

$$\texttt{(MENOR-P N1 N2)} = \begin{cases} \texttt{T} & \text{si } \texttt{N1} < \texttt{N2} \\ \texttt{NIL} & \text{en otro caso} \end{cases}$$

Por ejemplo,

```
(MENOR-P 0 3) ---> T
(MENOR-P 3 3) ---> NIL
```

Solución:

```
(DEFUN MENOR-P (N1 N2)
  (COND ((AND (IGUAL-P N1 0) (IGUAL-P N2 0)) NIL)
        ((IGUAL-P N1 0) T)
        ((IGUAL-P N2 0) NIL)
        (T (MENOR-P (PRED N1) (PRED N2)))  ))
```

El cálculo de (MENOR-P 3 5) se realiza como sigue:

```
(MENOR-P 3 5) = (MENOR-P 2 4) =
              = (MENOR-P 1 3) =
              = (MENOR-P 0 2) =
              = T
```

4.3.5 Ejercicio: Definir la función SUMA tal que

```
(SUMA N1 N2) = N1 + N2
```

(Indicación: usar los esquemas

$$\texttt{(N1 + N2)} = \begin{cases} \texttt{N2} & \text{si } \texttt{N1 = 0} \\ \texttt{(N1 - 1) + (N2 + 1)} & \text{en caso contrario} \end{cases}$$

$$\texttt{(N1 + N2)} = \begin{cases} \texttt{N2} & \text{si } \texttt{N1 = 0} \\ \texttt{((N1 - 1) + N2) + 1} & \text{en caso contrario} \end{cases}$$

Solución:

Usando el primer esquema:

```
(DEFUN SUMA (N1 N2)
  (COND ((= N1 0) N2)
        (T (SUMA (PRED N1) (SUC N2)))  ))
```

Para calcular (SUMA 2 3):

```
(SUMA 2 3) = (SUMA 1 4) =
           = (SUMA 0 5) =
           = 5
```

Usando el segundo esquema:

```
(DEFUN SUMA (N1 N2)
  (COND  ((= N1 0) N2)
         (T (SUC (SUMA (PRED N1) N2)))  ))
```

Para calcular (SUMA 2 3):

```
(SUMA 2 3) = (SUC (SUMA 1 3)) =
           = (SUC (SUC (SUMA 0 3))) =
           = (SUC (SUC 3)) =
           = (SUC 4) =
           = 5
```

4.3.6 Ejercicio: Determinar el número de llamadas a la función SUMA realizadas para calcular (SUMA 0 8), (SUMA 8 0), (SUMA X Y).

Solución: 0, 8 y X, respectivamente.

4.3.7 Ejercicio: Redefinir la función SUMA, SUMA 2, de forma que el número de llamadas a SUMA 2 al calcular (SUMA 2 X Y) sea el mínimo de X e Y.

Solución:

```
(DEFUN SUMA (N1 N2)
  (COND  ((MENOR-P N2 N1) (SUMA N2 N1))
         ((= N1 0) N2)
         (T (SUMA (PRED N1) (SUC N2)))  ))
```

4.3.8 Ejercicio: Definir la función RESTA tal que

$$\texttt{(RESTA N1 N2)} = \begin{cases} \texttt{NIL} & \text{si } \texttt{N1} < \texttt{N2} \\ \texttt{N1 - N2} & \text{en otro caso} \end{cases}$$

usando las funciones definidas en este parágrafo.

Solución: Por recursión sobre N2, se tiene:

$$\texttt{(RESTA N1 N2)} = \begin{cases} \texttt{NIL} & \text{si } \texttt{N1} < \texttt{N2} \\ \texttt{N1} & \text{si } \texttt{N2} = 0 \\ \texttt{((N1 - (N2 - 1)) - 1} & \text{si } \texttt{N2} > 0 \end{cases}$$

```
(DEFUN RESTA (N1 N2)
  (COND ((MENOR-P N1 N2) NIL)
        ((IGUAL-P N2 0) N1)
        (T (PRED (RESTA N1 (PRED N2))))  ))
```

4.3.9 Ejercicio: Definir la función PRODUCTO tal que

$$\texttt{(PRODUCTO N1 N2)} = \texttt{N1} * \texttt{N2}$$

usando las funciones definidas en este parágrafo.

Solución: Por recursión sobre N1:

$$\texttt{N1 * N2} = \begin{cases} \texttt{0} & \text{si } \texttt{N1 = 0} \\ \texttt{(N1 - 1) * N2 + N2} & \text{en otro caso} \end{cases}$$

```
(DEFUN PRODUCTO (N1 N2)
  (COND ((IGUAL-P N1 0) 0)
        (T (SUMA (PRODUCTO (PRED N1) N2) N2))  ))
```

Para disminuir el número de llamadas puede redefinirse

```
(DEFUN PRODUCTO (N1 N2)
  (COND ((MENOR-P N2 N1) (PRODUCTO N2 N1))
        ((IGUAL-P N1 0) 0)
        (T (SUMA (PRODUCTO (PRED N1) N2) N2))  ))
```

4.3.10 Ejemplo: Definir los predicados PAR-P e IMPAR-P tales que

$$\texttt{(PAR-P N)} = \begin{cases} \texttt{T} & \text{si N es par} \\ \texttt{NIL} & \text{en caso contrario} \end{cases}$$

$$\texttt{(IMPAR-P N)} = \begin{cases} \texttt{T} & \text{si N es impar} \\ \texttt{NIL} & \text{en caso contrario} \end{cases}$$

```
(DEFUN PAR-P (N)
  (COND ((IGUAL-P N 0) T)
        ((IGUAL-P N 1) NIL)
        (T (PAR-P (PRED (PRED N))))  ))

(DEFUN IMPAR-P (N)
  (COND ((IGUAL-P N 0) NIL)
        ((IGUAL-P N 1) T)
        (T (IMPAR-P (PRED (PRED N))))  ))
```

Ambos predicados pueden definirse mediante recursividad cruzada:

```
(DEFUN PAR-P (N)
  (COND ((IGUAL-P N 0) T)
        (T (IMPAR-P (PRED N)) )))

(DEFUN IMPAR-P (N)
  (COND ((IGUAL-P N 0) NIL)
        (T (PAR-P (PRED N)) )))
```

Por ejemplo, para calcular (PAR-P 3),

```
(PAR-P 1 3 ) = (IMPAR-P 1 2) = (PAR-P 1 1) = (IMPAR-P 1 0) = NIL
```

4.3.11 Ejercicio: Definir la función COCIENTE tal que

```
(COCIENTE N1 N2) = Parte entera de N1/N2
```

Por ejemplo:

```
(COCIENTE 7 2) = 3
```

Solución:

$$\texttt{COCIENTE (N1, N2)} = \begin{cases} \texttt{0} & \text{si } \texttt{N1} < \texttt{N2} \\ \texttt{COCIENTE (N1 - N2, N2) + 1} & \text{en otro caso} \end{cases}$$

```
(DEFUN COCIENTE (N1 N2)
  (COND ((MENOR-P N1 N2) 0)
        (T (SUC (COCIENTE (RESTA N1 N2) N2)))  ))
```

4.3.12 Ejercicio: Definir la función RESTO tal que

```
(RESTO N1 N2) = Resto de dividir N1 entre N2.
```

Por ejemplo,

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

Solución:

$$\texttt{RESTO (N1, N2)} = \begin{cases} \texttt{N1} & \text{si } \texttt{N1} < \texttt{N2} \\ \texttt{RESTO (N1 - N2, N2)} & \text{en otro caso} \end{cases}$$

```
(DEFUN RESTO (N1 N2)
  (COND ((MENOR-P N1 N2) N1)
        (T (RESTO (RESTA N1 N2) N2))  ))
```

4.3.13 Ejercicio: Definir la función DIVISION tal que (DIVISION N1 N2) sea una lista cuyo primer elemento sea la parte entera de N1/N2 y el segundo, el resto de dividir N1 entre N2. Por ejemplo,

```
(DIVISION 7 2) ---> (3 1)
```

Solución:

```
(DEFUN DIVISION (N1 N2)
  (COND ((MENOR-P N1 N2) (LIST 0 N1))
        (T (LET ((DIV (DIVISION (RESTA N1 N2) N2)))
             (LIST (SUC (CAR DIV)) (CADR DIV))  ))))
```

4.4 La aritmética ordinaria del lisp

4.4.1 Ejercicio: Definir la función POTENCIA tal que

$$\texttt{(POTENCIA N1 N2)} = \begin{cases} \texttt{INDET} & \text{si N1 = N2 = 0} \\ \texttt{0} & \text{si N1 = 0 y N2} \neq \texttt{0} \\ \texttt{1} & \text{si N1} \neq \texttt{0 y N2 = 0} \\ N1^{N2} & \text{en otro caso} \end{cases}$$

Por ejemplo,

```
(POTENCIA 2 3) ---> 8
```

Solución: Por inducción sobre N2 usando la relación

```
N1^N2 = N1 * N1^(N2 - 1)
```

```
(DEFUN POTENCIA (N1 N2)
  (COND ((AND (= N1 0) (= N2 0)) 'INDET)
        ((= N1 0) 0)
        ((= N2 0) 1)
        (T (* N1 (POTENCIA N1 (1- N2))))  ))
```

4.4.2 Ejercicio: Redefinir la función SQRT.

Solución:

```
(DEFUN N-SQRT (X)
  (N-SQRT-AUX (/ X 2) X) )

(DEFUN N-SQRT-AUX (RAIZ X)
  (IF (BUENAP RAIZ X) RAIZ
      (N-SQRT-AUX (MEJORA RAIZ X) X) ))

(DEFUN BUENAP (RAIZ X)
  (< (ABS (- X (* RAIZ RAIZ))) 0.001) )

(DEFUN MEJORA (RAIZ X)
  (/ (+ RAIZ (/ X RAIZ)) 2) )
```

4.5 Simulación de primitivas sobre listas

4.5.1 Ejercicio: Redefinir la función LENGTH (llamarle N-LENGTH).
Solución:

```
(DEFUN N-LENGTH (L)
  (COND ((NULL L) 0)
        (T (1+ (N-LENGTH (CDR L))))  ))
```

Por ejemplo,

```
(N-LENGTH '(A B C)) =
        = (1+ (N-LENGTH '(B C))) =
        = (1+ (1+ (N-LENGTH '(C)))) =
        = (1+ (1+ (1+ (N-LENGTH '())))) =
        = (1+ (1+ (1+ 0))) =
        = (1+ (1+ 1)) =
        = (1+ 2) =
        = 3
```

4.5.2 Ejercicio: Redefinir la función NTH (llamarle N-NTH).
Solución:

```
(DEFUN N-NTH (N L)
  (COND ((NULL L) ())
        ((= N 0) (CAR L))
        (T (N-NTH (1- N) (CDR L)))  ))
```

Por ejemplo,

```
(N-NTH 2 '(A B C D)) =
        = (N-NTH 1 '(B C D)) =
        = (N-NTH 0 '(C D)) =
        = (CAR '(C D)) =
        = C
```

4.5.3 Ejercicio: Redefinir la función APPEND (llamarle N-APPEND).
Solución:

```
(DEFUN N-APPEND (L1 L2)
  (COND ((NULL L1) L2)
        (T (CONS (CAR L1) (N-APPEND (CDR L1) L2)))  ))
```

Por ejemplo,

```
(N-APPEND '(A B) '(C D E)) =
        = (CONS 'A (N-APPEND '(B) '(C D E))) =
        = (CONS 'A (CONS 'B (N-APPEND '() '(C D E)))) =
        = (CONS 'A (CONS 'B '(C D E))) =
        = (CONS 'A '(B C D E)) =
        = (A B C D E) =
```

4.5.4 Ejercicio: Redefinir la función MEMBER de forma que

```
(N-MEMBER 'B '(A B C))          ---> (B C)
(N-MEMBER '(B) '((A) (B) (C))) ---> ((B) (C))
```

Solución:

```
(DEFUN N-MEMBER (E L)
  (COND ((NULL L) ())
        ((EQUAL E (CAR L)) L)
        (T (N-MEMBER E (CDR L)))  ))
```

Por ejemplo,

```
(N-MEMBER 'B '(A B C)) =
        = (N-MEMBER 'B '(B C)) =
        = (B C)
```

4.5.5 Ejercicio: La función (REVERSE L) devuelve la lista obtenida escribiendo los elementos de L en orden inverso. Por ejemplo,

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

Redefinir la función REVERSE y llamarle N-REVERSE.

Solución:

```
(DEFUN N-REVERSE (L)
  (COND ((NULL L) ())
        (T (APPEND (N-REVERSE (CDR L)) (LIST (CAR L)) ))))
```

Por ejemplo,

```
(N-REVERSE '(A B)) =
        = (APPEND (N-REVERSE '(B)) (LIST 'A)) =
        = (APPEND (APPEND (N-REVERSE '()) (LIST 'B)) '(A)) =
        = (APPEND (APPEND () '(B)) '(A)) =
        = (APPEND '(B) '(A)) =
        = (B A)
```

4.5.6 Ejercicio: La función (SUBST E1 E2 L) devuelve una lista de L en la que todas las ocurrencias de la expresión E2 se han sustituido por la expresión E1. Por ejemplo:

```
(SUBST 5 'A '(A B (A C))) ---> (5 B (5 C))
```

Redefinir la función SUBST y llamarle N-SUBST.

Solución:

```
(DEFUN N-SUBST (E1 E2 L)
  (COND ((ATOM L) (IF (EQ L E2) E1 L))
        ((EQUAL (CAR L) E2)
         (CONS E1 (N-SUBST E1 E2 (CDR L))) )
        (T (CONS (N-SUBST E1 E2 (CAR L))
                 (N-SUBST E1 E2 (CDR L))  ))))
```

4.5.7 Ejercicio: La función (REMOVE S L) devuelve la lista obtenida a partir de L borrando las estancias, de primer nivel, de la expresión S. Por ejemplo,

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

Redefinir la función REMOVE y llamarle N-REMOVE.

Solución:

```
(DEFUN N-REMOVE (S L)
  (COND ((NULL L) ())
        ((EQUAL S (CAR L)) (REMOVE S (CDR L)))
        (T (CONS (CAR L)
                 (N-REMOVE S (CDR L))  ))))
```

4.6 Definición de funciones sobre listas

4.6.1 Ejercicio: Definir la función SUBSTOP que trabaje como SUBST pero sólo en el primer nivel. Por ejemplo,

```
(SUBSTOP 5 'A '(A (A B) A)) ---> (5 (A B) 5)
```

Solución:

```
(DEFUN SUBSTOP (E1 E2 L)
  (COND ((NULL L) ())
        (T (CONS (COND ((EQUAL E2 (CAR L)) E1)
                       (T (CAR L)))
                 (SUBSTOP E1 E2 (CDR L))  ))))
```

4.6.2 Ejercicio: Definir la función REV que actúe como REVERSE pero a todos los niveles. Por ejemplo,

```
(REV '(A (B C) (D (E)))) ---> (((E) D) (C B) A)
```

Solución:

```
(DEFUN REV (L)
  (COND ((NULL L) L)
        ((ATOM (CAR L)) (APPEND (REV (CDR L)) (LIST (CAR L))))
        (T (APPEND (REV (CDR L)) (LIST (REV (CAR L))))) ))
```

4.6.3 Ejercicio: Definir la función (LINEALIZA L) que aplicada a una lista, suprime todos los paréntesis internos salvo los correspondientes a (). Por ejemplo:

```
(LINEALIZA '(A (B (C)) () E)) ---> (A B C NIL E)
```

Solución:

```
(DEFUN LINEALIZA (L)
  (COND ((NULL L) ())
        ((ATOM (CAR L)) (CONS (CAR L) (LINEALIZA (CDR L))))
        (T (APPEND (LINEALIZA (CAR L))
                   (LINEALIZA (CDR L))  ))))
```

4.6.4 Ejercicio: Definir la función CUENTA-ATOMOS que aplicada a una lista, devuelve el número de átomos que contiene. Por ejemplo,

```
(CUENTA-ATOMOS '(A (B (C)) () E)) ---> 5
```

Solución:

```
(DEFUN CUENTA-ATOMOS (L)
  (LENGTH (LINEALIZA L)))

(DEFUN CUENTA-ATOMOS (L)
  (COND ((NULL L) 0)
        ((ATOM L) 1)
        ((ATOM (CAR L)) (1+ (CUENTA-ATOMOS (CDR L))))
        (T (+ (CUENTA-ATOMOS (CAR L))
              (CUENTA-ATOMOS (CDR L))  ))))
```

4.6.5 Ejercicio: Definir (MISMA-FORMA E1 E2) que devuelva T o `NIL`, según que las expresiones E1 y E2 tengan o no la misma forma. Por ejemplo,

```
(MISMA-FORMA '(A (B C)) '(1 (2 3))) ---> T
```

Solución:

```
(DEFUN MISMA-FORMA (E1 E2)
  (COND ((AND (NULL E1) (NULL E2)) T)
        ((OR (NULL E1) (NULL E2)) ())
        ((AND (ATOM E1) (ATOM E2)) T)
        ((OR (ATOM E1) (ATOM E2)) ())
        (T (AND (MISMA-FORMA (CAR E1) (CAR E2))
                (MISMA-FORMA (CDR E1) (CDR E2)) ))))
```

4.6.6 Ejercicio: Definir MEM que devuelva T o `NIL` y actúe como MEMBER pero a todos los niveles. Por ejemplo,

```
(MEM 'A '((A B) C)) ---> T
```

Solución:

```
(DEFUN MEM (E L) ; USANDO LINEALIZA
   (IF (MEMBER E (LINEALIZA L)) T))

(DEFUN MEM (E L) ; RECURSIVA
  (COND ((NULL L) ())
        ((ATOM (CAR L)) (OR (EQUAL E (CAR L))
                            (MEM E (CDR L))  ))
        (T (MEM E (APPEND (CAR L) (CDR L))))  ))
```

4.6.7 Ejercicio: Definir la función SEPARA que tome una lista (de los niveles que sea) y separe las letras de los números. Por ejemplo,

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

Solución:

```
(DEFUN SEPARA (L)
  (COND ((NULL L) (LIST () ()))
        ((NUMBERP L) (LIST () (LIST L)))
        ((SYMBOLP L) (LIST (LIST L) ()))
        (T (LIST (APPEND (CAR (SEPARA (CAR L)))
                         (CAR (SEPARA (CDR L))))
                 (APPEND (CADR (SEPARA (CAR L)))
                         (CADR (SEPARA (CDR L)))  )))))
```

Nota: La siguiente redefinición es más eficaz:

```
(DEFUN SEPARA (L)
  (COND ((NULL L) (LIST () ()))
        ((NUMBERP L) (LIST () (LIST L)))
        ((SYMBOLP L) (LIST (LIST L) ()))
        (T (LET ((L1 (SEPARA (CAR L)))
                 (L2 (SEPARA (CDR L))))
             (LIST (APPEND (CAR L1) (CAR L2))
                   (APPEND (CADR L1) (CADR L2)))  )))))
```

4.6.8 Ejercicio: Escribir la función DOBLA-EL que duplique todos los elementos de la lista que toma por argumento. Por ejemplo,

```
(DOBLA-EL '(A B))   ---> (A A B B)
(DOBLA-EL '((A) B)) ---> ((A) (A) B B)
```

Solución:

```
(DEFUN DOBLA-EL (L)
  (COND ((NULL L) ())
        (T (CONS (CAR L)
                 (CONS (CAR L) (DOBLA-EL (CDR L)))  ))))
```

4.6.9 Ejercicio: Escribir la función DOBLA-ATOMOS que duplique todos los átomos de la lista que toma como argumento en el nivel que se encuentre. Por ejemplo,

```
(DOBLA-ATOMOS '((A) B)) ---> ((A A) B B)
```

Solución:

```
(DEFUN DOBLA-ATOMOS (L)
  (COND ((NULL L) L)
        ((ATOM (CAR L))
         (CONS (CAR L) (CONS (CAR L) (DOBLA-ATOMOS (CDR L)))))
        (T (CONS (DOBLA-ATOMOS (CAR L))
                 (DOBLA-ATOMOS (CDR L)) ))))
```

4.6.10 Ejercicio: Escribir una función AGRUPA que agrupa los elementos sucesivos de dos listas. Por ejemplo,

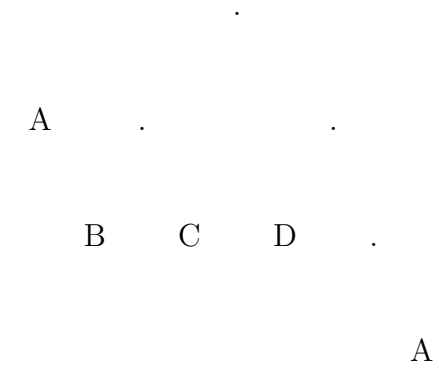
```
(AGRUPA '(A B C) '(1 2 3)) ---> ((A 1) (B 2) (C 3))
(AGRUPA '(A B) '(1 2 3))   ---> ((A 1) (B 2) 3)
(AGRUPA '(A B C) '(1 2))   ---> ((A 1) (B 2) C)
```

Solución:

```
(DEFUN AGRUPA (L1 L2)
  (COND ((NULL L1) L2)
        ((NULL L2) L1)
        (T (CONS (LIST (CAR L1) (CAR L2))
                 (AGRUPA (CDR L1) (CDR L2)) ))))
```

4.7 Funciones sobre árboles

4.7.1 Nota: Los árboles pueden representarse por listas. Por ejemplo, el árbol



se representa por la lista (A (B C) (D (A)) E).

4.7.2 Ejercicio: Escribir la función (MISMO-ARBOL L1 L2) que devuelva T si L1 y L2 representan el mismo árbol y `NIL`, si no lo representan. Por ejemplo

```
(MISMO-ARBOL '(1 (2 (3))) '(A (B (C)))) ---> T
```

Solución: Es igual que MISMA-FORMA. Otra definición es:

```
(DEFUN MISMO-ARBOL (L1 L2)
  (COND ((ATOM L1) (ATOM L2))
        ((ATOM L2) NIL)
        (T (AND (MISMO-ARBOL (CAR L1) (CAR L2))
                (MISMO-ARBOL (CDR L1) (CDR L2))  ))))
```

4.7.3 Ejercicio: Definir la función N-HOJAS que devuelva el número de hojas del árbol asociado a la lista que toma por argumento. Por ejemplo,

```
(N-HOJAS '(A (B C) (C (A)) E)) ---> 6
```

Solución:

```
(DEFUN N-HOJAS (L)
  (CUENTA-ATOMOS L))
```

4.7.4 Ejercicio: Definir (N-ARCOS L) que devuelva el número de arcos del árbol asociado a L. Por ejemplo,

```
(N-ARCOS '(A (B C) (C (A)) E)) ---> 9
```

Solución:

```
(DEFUN N-ARCOS (L)
  (COND ((ATOM L) 0)
        (T (+ 1 (N-ARCOS (CAR L)) (N-ARCOS (CDR L)))  )))
```

4.7.5 Ejercicio: Definir (PROFUNDIDAD L) que devuelva el número de niveles que existen en L; es decir, el número de arcos de la raíz a la hoja más lejana. Por ejemplo,

```
(PROFUNDIDAD '(A (B C) (D (A)) E)) ---> 3
```

Solución:

```
(DEFUN PROFUNDIDAD (L)
  (COND ((ATOM L) 0)
        (T (MAX (1+ (PROFUNDIDAD (CAR L)))
                (PROFUNDIDAD (CDR L))  ))))
```

4.8 Funciones sobre conjuntos

4.8.1 Nota: Representamos los conjuntos como listas sin elementos repetidos. Por ejemplo, (A B C) es un conjunto; pero, (A B A C) no lo es. Además, el orden de los elementos de un conjunto no es esencial. Así, (A B C) y (A C B) representan el mismo conjunto.

4.8.2 Ejercicio: Definir REDUCE-CONJUNTO que tome una lista y elimine los elementos repetidos; esto es, la convierta en "conjunto". Por ejemplo,

```
(REDUCE-CONJUNTO ' (A B A C)) ---> (B A C)
```

Solución:

```
(DEFUN REDUCE-CONJUNTO (L)
  (COND ((ATOM L) L)
        ((MEMBER (CAR L) (CDR L))
         (REDUCE-CONJUNTO (CDR L)))
        (T (CONS (REDUCE-CONJUNTO (CAR L))
                 (REDUCE-CONJUNTO (CDR L))))))
```

4.8.3 Ejercicio: Definir (UNION C1 C2) que devuelva la unión de los conjuntos C1 y C2 (i.e. una lista sin elementos repetidos que contenga los elementos que están en C1 o en C2). Por ejemplo,

```
(UNION '(0 3 6 9 12) '(0 4 8 12)) ---> (3 6 9 0 4 8 12)
```

Solución:

```
(DEFUN UNION (C1 C2)
  (COND ((NULL C1) C2)
        ((MEMBER (CAR C1) C2) (UNION (CDR C1) C2))
        (T (CONS (CAR C1) (UNION (CDR C1) C2)))))
```

Puede definirse usando REDUCE-CONJUNTO:

```
(DEFUN UNION (C1 C2)
  (REDUCE-CONJUNTO (APPEND C1 C2)))
```

4.8.4 Ejercicio: Definir (INTERSECCION C1 C2) que devuelva la intersección de los conjuntos C1 y C2 (i. e. la lista de sus elementos comunes). Por ejemplo,

```
(INTERSECCION '(0 3 6 9 12) '(0 4 8 12)) ---> (0 12)
```

Solución:

```
(DEFUN INTERSECCION (C1 C2)
  (COND ((OR (NULL C1) (NULL C2)) ())
        ((MEMBER (CAR C1) C2)
         (CONS (CAR C1) (INTERSECCION (CDR C1) C2)))
        (T (INTERSECCION (CDR C1) C2))))
```

4.8.5 Ejercicio: Definir (DIF C1 C2) que devuelva C1 - C2 (i.e. la lista de los elementos de C1 que no son elementos de C2). Por ejemplo,

```
(DIF '(0 3 6 9 12) '(0 4 8 12)) ---> (3 6 9)
```

Solución:

```
(DEFUN DIF (C1 C2)
  (COND ((NULL C1) ())
        ((MEMBER (CAR C1) C2) (DIF (CDR C1) C2))
        (T (CONS (CAR C1) (DIF (CDR C1) C2)))))
```

4.8.6 Ejercicio: Definir (DIF-SIM C1 C2) que devuelva la diferencia simétrica de C1 y C2 (i.e. la lista de los elementos que sólo están en uno de los conjuntos C1 ó C2).Por ejemplo,

```
(DIF-SIM '(0 3 6 9 12) '(0 4 8 12)) ---> (3 6 9 4 8)
```

Solución:

```
(DEFUN DIF-SIM (C1 C2)
  (DIF (UNION C1 C2) (INTERSECCION C1 C2)))
```

4.8.7 Ejercicio: Definir (IGUAL-CONJ C1 C2) que devuelva T si los conjuntos C1 y C2 son iguales y NIL, si no lo son. Por ejemplo,

```
(IGUAL-CONJ '(A B C) '(C A B)) ---> T
```

Solución:

```
(DEFUN IGUAL-CONJ (C1 C2)
  (COND ((NULL C1) (NULL C2))
        ((MEMBER (CAR C1) C2)
         (IGUAL-CONJ (CDR C1) (REMOVE (CAR C1) C2)))
        (T ())))
```

La siguiente definición es más eficaz:

```
(DEFUN IGUAL-CONJ (C1 C2)
  (COND ((NULL C1) (NULL C2))
        ((NOT (MEMBER (CAR C1) C2)) ())          ; NOT = NULL
        (T (IGUAL-CONJ (CDR C1) (REMOVE (CAR C1) C2)))))
```

La siguiente se basa en DIF-SIM

```
(DEFUN IGUAL-CONJ (C1 C2)
  (IF (DIF-SIM C1 C2) () T))
```

Capítulo 5

La iteración en Lisp

5.1 El grupo PROG, GO, RETURN

5.1.1 Definición: La función PROG:

(PROG ((VAR1 VAL1) ... (VARN VALN)) S1 ... SM) asigna a la variable VAR1 el valor VAL1,..., a la variable VARN el valor VALN, evalúa las expresiones S1,...,SM. Si no se detiene con un RETURN, devuelve el valor de SM.

Notas:

1. Si a una variable no se le ha especificado un valor, se le asocia NIL.
2. También puede escribirse (PROG () S1 ... SM)
3. La asignación de las variables es local.
4. Cuando dentro de un PROG aparece un símbolo, no se evalúa, sino que se toma como etiqueta.

5.1.2 Definición: La función RETURN:

(RETURN S) , cuando está dentro de PROG, devuelve S y detiene el ciclo.

5.1.3 Definición: La función GO

(GO SIMB) , cuando está dentro de PROG, transfiere el control a SIMB.

5.1.4 Ejemplo: Redefinir la función factorial usando PROG.

Solución:

```
(DEFUN FACT (N)
  (PROG ((CONTADOR N)
         (RESULTADO 1))
    ETIQUETA
    (IF (= 0 CONTADOR) (RETURN RESULTADO))
    (SETQ RESULTADO (* CONTADOR RESULTADO)
          CONTADOR (1- CONTADOR))
    (GO ETIQUETA)))
```

Por ejemplo, para calcular (FACT 3)

Iteracion	CONTADOR	RESULTADO
0	3	1
1	2	3
2	1	6
3	0	6
4		6

5.1.5 Ejercicio: Redefinir, usando PROG,

$$\texttt{(POTENCIA2 M N)} = M^N$$

Solución:

```
(DEFUN POTENCIA (M N)
  (PROG ((RESULTADO 1))
    ETIQUETA
    (SETQ RESULTADO (* M RESULTADO)
          N (1- N))
    (IF (= N 0) (RETURN RESULTADO))
    (GO ETIQUETA)))
```

5.1.6 Ejercicio: Escribir la definición iterativa de la función LENGTH usando PROG.

Solución:

```
(DEFUN N-LENGTH (L)
  (PROG ((AUX 0))
    ETIQUETA
    (IF (NULL L) (RETURN AUX))
    (SETQ AUX (1+ AUX)
          L (CDR L))
    (GO ETIQUETA)))
```

Por ejemplo, para calcular (N-LENGTH '(A B C))

Iteracion	AUX	L
0	0	(A B C)
1	1	(B C)
2	2	(C)
3	3	()
4	3	

5.1.7 Ejercicio: Escribir la definición iterativa de la función NTHCDR usando PROG.

Solución:

```
(DEFUN N-NTHCDR (N L)
  (PROG ()
    ETIQUETA
    (IF (= N 0) (RETURN L))
    (SETQ N (1- N)
          L (CDR L))
    (GO ETIQUETA)))
```

5.1.8 Ejercicio: Escribir la definición iterativa de la función MEMBER usando PROG.

Solución:

```
(DEFUN N-MEMBER (S L)
  (PROG ()
    ETIQUETA
    (COND ((NULL L) (RETURN NIL))
          ((EQUAL S (CAR L)) (RETURN L)))
    (SETQ L (CDR L))   ; PUEDE SUSTITUIRSE POR (POP L)
    (GO ETIQUETA)))
```

5.1.9 Ejercicio: Escribir la definición iterativa, usando PROG, de la función REVERSE.

Solución:

```
(DEFUN N-REVERSE (L)
  (PROG (AUX)
    ETIQUETA
    (IF (NULL L) (RETURN AUX))
    (SETQ AUX (CONS (CAR L) AUX)  ; PUEDE SUSTITUIRSE POR
          L (CDR L))              ; (PUSH (POP L) AUX)
    (GO ETIQUETA)))
```

Por ejemplo, para calcular (N-REVERSE '(A B C)):

Iteracion	L	AUX
0	(A B C)	()
1	(B C)	(A)
2	(C)	(B A)
3	()	(C B A)

5.1.10 Ejercicio: Definir iterativamente la función APPEND, usando PROG.

Solución:

```
(DEFUN N-APPEND (L1 L2)
  (SETQ L1 (REVERSE L1))   ; L1 ES LOCAL
  (PROG ()
    ETIQUETA
    (IF (NULL L1) (RETURN L2))
    (PUSH (POP L1) L2)
    (GO ETIQUETA)))
```

Por ejemplo, para calcular (N-APPEND '(A B) '(C D)):

Iteracion	L1	L2
0	(A B)	(C D)
0	(B A)	(C D)
1	(A)	(B C D)
2	()	(A B C D)

5.2 Las funciones DO y DO*

5.2.1 Definición: 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).

5.2.2 Ejemplo: Redefinir LENGTH usando DO.

Solución:

```
(DEFUN N-LENGTH (L)
  (DO ((X L (CDR X))
       (N 0 (1+ N)))
      ((NULL X) N)))
```

Por ejemplo, para calcular (N-LENGTH '(A B C D))

Iteracion	X	N
0	(A B C)	0
1	(B C)	1
2	(C)	2
3	()	3
4		3

5.2.3 Ejercicio: Redefinir FACTORIAL usando DO.

Solución:

```
(DEFUN FACT (N)
  (DO ((CONTADOR N (1- CONTADOR))
       (RESULTADO 1))
      ((= CONTADOR 1) RESULTADO)
    (SETQ RESULTADO (* RESULTADO CONTADOR)) ))
```

5.2.4 Nota: Las funciones LET, PROG y DO asocian sus valores en paralelo; es decir,

```
(LET ((A 1)(B 2))(+ A B)) ---> 3
(LET ((A 1)(B A))(+ A B)) ---> ERROR Unbound variable: A
```

Las correspondientes funciones que efectúan la asociación secuencial son `LET*` , `PROG*` y `DO*` . Por ejemplo,

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

5.2.5 Ejercicio: Redefinir FACTORIAL usando DO*.

Solución:

```
(DEFUN FACT (N)
  (DO* ((CONTADOR N (1- CONTADOR))
        (RESULTADO N (* RESULTADO CONTADOR)))
       ((= CONTADOR 1) RESULTADO)))
```

5.2.6 Ejercicio: Redefinir REVERSE usando DO*.

Solución:

```
(DEFUN N-REVERSE (L)
  (DO* ((L1 L (CDR L1))
        (RESULTADO (LIST (CAR L1)) (CONS (CAR L1) RESULTADO)))
       ((NULL (CDR L1)) RESULTADO)))
```

5.2.7 Ejercicio: Redefinir, usando DO, la función

$$\texttt{(POTENCIA M N)} = M^N$$

Solución:

```
(DEFUN POTENCIA (M N)
  (DO ((RESULTADO 1 (* M RESULTADO))
       (CONTADOR N (1- CONTADOR)))
      ((= CONTADOR 0) RESULTADO)))
```

5.2.8 Ejercicio: Definir la función (SIMBOLOS-DE L) que devuelva una lista de los símbolos de L. Por ejemplo,

```
(SIMBOLOS-DE '(1 + 1 = 2)) ---> (= +)
```

Solución:

```
(DEFUN SIMBOLOS-DE (L)
  (DO ((L1 L (CDR L1))
       (RESULTADO NIL))
      ((NULL L1) RESULTADO)
    (UNLESS (NUMBERP (CAR L1))
      (SETQ RESULTADO (CONS (CAR L1) RESULTADO)))))
```

5.2.9 Ejercicio: Definir la función (PONE-PARENTESIS L) que devuelva una lista en la que los elementos de L están entre paréntesis. Por ejemplo,

```
(PONE-PARENTESIS '(A B C)) ---> ((A)(B)(C))
```

Solución:

```
(DEFUN PONE-PARENTESIS (L)
  (DO ((L1 L (CDR L1))
       (RESULTADO NIL))
      ((NULL L1) (REVERSE RESULTADO))
    (SETQ RESULTADO (CONS (LIST (CAR L1)) RESULTADO))))
```

5.3 Las funciones DOTIMES y DOLIST

5.3.1 Definición: La función DOTIMES:

(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
```

5.3.2 Ejercicio: Redefinir FACT usando DOTIMES.

Solución:

```
(DEFUN FACT (N)
  (LET ((RESULTADO 1))
    (DOTIMES (CONTADOR N RESULTADO)
      (SETQ RESULTADO (* RESULTADO (1+ CONTADOR))) )))
```

5.3.3 Ejercicio: Redefinir POTENCIA usando DOTIMES.

Solución:

```
(DEFUN POTENCIA (M N)
  (LET ((RESULTADO 1))
    (DOTIMES (CONTADOR N RESULTADO)
      (SETQ RESULTADO (* RESULTADO M)) )))
```

5.3.4 Ejercicio: Redefinir REVERSE usando DOTIMES.

Solución:

```
(DEFUN N-REVERSE (L)
  (LET ((RESULTADO)
        (N (LENGTH L)) )
    (DOTIMES (CONTADOR N RESULTADO)
      (SETQ RESULTADO (CONS (CAR L) RESULTADO)
            L (CDR L) ))))
```

5.3.5 Definición: La función DOLIST:

(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
```

5.3.6 Ejercicio: Redefinir REVERSE usando DOLIST.

Solución:

```
(DEFUN N-REVERSE (L)
  (LET ((RESULTADO))
    (DOLIST (CONTADOR L RESULTADO)
      (SETQ RESULTADO (CONS CONTADOR RESULTADO)) )))
```

5.3.7 Ejercicio: Definir, usando DOLIST, la función

```
(CUENTA-APROBADOS L)
```

de forma que si L es una lista de notas, devuelva el número de aprobados. Por ejemplo,

```
(CUENTA-APROBADOS '(1 6 5 3 7))   --->  3
```

Solución:

```
(DEFUN CUENTA-APROBADOS (L)
  (LET ((RESULTADO 0))
    (DOLIST (CONTADOR L RESULTADO)
      (IF (>= CONTADOR 5) (SETQ RESULTADO (1+ RESULTADO))) )))
```

5.4 La función MAPCAR

5.4.1 Definición: La función MAPCAR:

(MAPCAR FN L1 ... LN) devuelve una lista con los resultados de aplicar la función FN a los CAR de las listas L1,...,LN; luego a los CADR; etc. Por ejemplo,

```
(MAPCAR 'ATOM '(A (B) 3))         ---> (T NIL T)
(MAPCAR '> '(5 8 3) '(4 9 2))     ---> (T NIL T)
(MAPCAR '+ '(1 2) '(3 4) '(5 6)) ---> (9 12)
(DEFUN CUADRADO (N) (* N N))      ---> CUADRADO
(MAPCAR 'CUADRADO '(1 2 3))       ---> (1 4 9)
```

5.4.2 Ejercicio: Redefinir SUBST, usando MAPCAR.

Solución:

```
(DEFUN N-SUBST (E1 E2 L)
  (N-SUBST-AUX L) )

(DEFUN N-SUBST-AUX (L)
  (COND ((EQUAL E2 L) E1)
        ((ATOM L) L)
        (T (MAPCAR #'N-SUBST-AUX L)) ))
```

5.4.3 Definición: La función APPLY:

(APPLY FN L) devuelve el valor de la función FN actuando sobre los elementos de L como argumentos. Por ejemplo,

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

5.4.4 Ejercicio: Redefinir CUENTA-ATOMOS usando APPLY.

Solución:

```
(DEFUN CUENTA-ATOMOS (L)
  (COND ((NULL L) 0)
        ((ATOM L) 1)
        (T (APPLY '+ (MAPCAR 'CUENTA-ATOMOS L)))))
```

5.4.5 Ejercicio: Definir (SUMA-VECTORES V1 V2) que devuelva la suma de los vectores V1 y V2. Por ejemplo,

```
(SUMA-VECTORES '(3 5) '(4 3)) ---> (7 8)
```

Solución:

```
(DEFUN SUMA-VECTORES (V1 V2)
  (MAPCAR '+ V1 V2))
```

5.4.6 Ejercicio: Definir (NORMA V) que devuelva la norma del vector V; es decir, la raiz cuadrada de la suma de los cuadrados de sus argumentos. Por ejemplo,

```
(NORMA '(3 4)) ---> 5.0
```

Solución:

```
(DEFUN NORMA (V)
    (SQRT (APPLY '+ (MAPCAR 'CUADRADO V))))
```

5.4.7 Ejercicio: Definir (PRODUCTO-ESCALAR V1 V2) que devuelve el producto escalar de V1 y V2; es decir,

```
(PRODUCTO-ESCALAR '(A1 ... AN) '(B1 ...BN)) ---> A1.B1 +...+ AN.BN
```

Por ejemplo,

```
(PRODUCTO-ESCALAR '(2 3) '(4 5)) ---> 23
```

Solución:

```
(DEFUN PRODUCTO-ESCALAR (V1 V2)
  (APPLY '+ (MAPCAR '* V1 V2)))
```

5.4.8 Ejercicio: Redefinir la función REV.

```
(REV '(A (B C) (D (E)))) ---> (((E) D) (C B) A)
```

Solución:

```
(DEFUN REV (L)
  (COND ((ATOM L) L)
        (T (REVERSE (MAPCAR 'REV L)))))
```

Capítulo 6

Funciones anónimas

6.1 La función LAMBDA

6.1.1 Definición: La función LAMBDA:

((LAMBDA (VAR1 ... VARN) S1 ... SM) VAL1 ... VALN) asocia a la variable VAR1 el valor VAL1,..., a la variable VARN el valor VALN, evalúa las expresiones S1,...,SM y devuelve el valor de SM.

6.1.2 Ejemplo:

```
((LAMBDA (M N) (+ M N)) 2 3)              ---> 5
((LAMBDA (X Y) (+ (* X X) (* Y Y))) 3 4) ---> 25
(SETQ N1 10)                              ---> 10
((LAMBDA (N) (* N 2)) N1)                 ---> 20
((LAMBDA (X) (LIST X X)) 2)               ---> (2 2)
((LAMBDA (X Y) (CONS Y X)) '(B) 'A)       ---> (A B)
((LAMBDA (L) (CAR L)) '(X Y))             ---> X

(MAPCAR #'(LAMBDA (N) (* N N)) '(1 2 3))  ---> (1 4 9)
((LAMBDA (N) (* N N)) '(3))               ---> ** * : error de argumento
(APPLY #'(LAMBDA (N) (* N N)) '(3))       ---> 9
```

6.1.3 Ejercicio: Definir la función (PARTES-DE L) que devuelva los subconjuntos de L. Por ejemplo,

```
(PARTES-DE '(A B C)) ---> (NIL (C) (B) (B C) (A) (A C) (A B) (A B C))
```

Solución:

```
(DEFUN PARTES-DE (L)
  (COND ((NULL L) (LIST NIL))
        (T (APPEND (PARTES-DE (CDR L))
                   (MAPCAR #'(LAMBDA (X) (CONS (CAR L) X))
                           (PARTES-DE (CDR L)) )))))
```

6.2 Las funciones EVERY y SOME

6.2.1 Definición: Las funciones EVERY y SOME:

(EVERY FN L1 ... LN) va aplicando FN a los elementos de la lista hasta que una aplicación dé `NIL` o se termine una lista. Devuelve la última aplicación.

(SOME FN L1 ... LN) actúa análogamente pero busca aplicaciones distintas de `NIL`.

6.2.2 Ejemplos:

```
(EVERY #'ATOM '(1 A))                  ---> T
(EVERY #'ATOM '((1) A))                ---> NIL
(EVERY #'= '(1 2) '(1 2))              ---> 2
(SOME #'ATOM '((1) A))                 ---> T
(SOME #'ATOM '((1) (A)))               ---> NIL
```

6.2.3 Ejercicio: Definir una función (SUBCONJUNTO A B) que devuelva T si A es un subconjunto de B y `NIL` en caso contrario. Por ejemplo,

```
(SUBCONJUNTO '(A C) '(A B C)) ---> T
(SUBCONJUNTO '(A D) '(A B C)) ---> NIL
```

Solución:

```
(DEFUN SUBCONJUNTO (A B)
  (IF (EVERY #'(LAMBDA (X) (MEMBER X B)) A) T NIL))
```

[En palabras: Si para todo X de A, X pertenece a B, devuelve T]. Otras definiciones son:

```
(DEFUN SUBCONJUNTO (A B)
  (IGUAL-CONJUNTO (UNION A B) B))

(DEFUN SUBCONJUNTO (A B)
  (IGUAL-CONJUNTO (INTERSECCION A B) A))
```

6.2.4 Ejercicio: Definir una función (DISJUNTOP A B) que devuelva T si A y B son disjuntos y `NIL`, en caso contrario. Por ejemplo,

```
(DISJUNTOP '(A C) '(A B)) ---> NIL
(DISJUNTOP '(A C) '(D B)) ---> T
```

Solución: Damos dos definiciones:

$$\texttt{(DISJUNTOP A B)} \leftrightarrow \neg(\exists x \in A)[x \in B]$$

```
(DEFUN DISJUNTOP (A B)
  (IF (NOT (SOME #'(LAMBDA (X) (MEMBER X B))
                 A))
      T
      NIL))

(DEFUN DISJUNTOP (A B)
  (NULL (INTERSECCION A B)))
```

Capítulo 7

Las A–listas (listas de asociación)

7.1 Pares punteados

7.1.1 Nota: Al aplicar la función CONS a dos átomos se obtiene un par punteado. Por ejemplo,

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

7.1.2 Nota: Al aplicarle la función CAR a un par punteado se obtiene el primer elemento y al aplicarle CDR, el segundo. Por ejemplo,

```
(CAR '(A . B))  ---> A
(CDR '(A . B))  ---> B
```

7.1.3 Nota: Las listas son casos particulares de pares punteados. Por ejemplo,

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

7.2 A–listas

7.2.1 Definición: 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).

7.2.2 Nota: Las A–listas pueden crearse mediante SETQ. Por ejemplo,

```
(SETQ L '((B . 2)(C . 3)))   ---> ((B . 2) (C . 3))
```

7.2.3 Definición: La función ACONS:

(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. Por ejemplo,

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

7.2.4 Ejercicio: Redefinir la función ACONS.

Solución:

```
(DEFUN N-ACONS (S1 S2 L)
  (CONS (CONS S1 S2 ) L))
```

7.2.5 Definición: La función PAIRLIS:

(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. Por ejemplo,

```
(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))
```

7.2.6 Ejercicio: Redefinir PAIRLIS.

Solución:

```
(DEFUN N-PAIRLIS (L1 L2 AL)
  (IF (AND (CONSP L1) (LIST L2))
      (ACONS (CAR L1)
             (CAR L2)
             (N-PAIRLIS (CDR L1) (CDR L2) AL))
      AL))
```

7.2.7 Definición: La función ASSOC:

(ASSOC SIMB AL) devuelve el elemento de la A–lista AL cuya clave es SIMB. Por ejemplo,

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

7.2.8 Ejercicio: Redefinir ASSOC.

Solución:

```
(DEFUN N-ASSOC (S AL)
  (COND ((ATOM AL) ())
        ((AND (CONSP (CAR AL))
              (EQUAL S (CAAR AL)))
         (CAR AL))
        (T (N-ASSOC S (CDR AL)))))
```

7.2.9 Definición: La función RASSOC:

(RASSOC VAL AL) devuelve el primer elemento de la A–lista AL cuyo valor es VAL. Por ejemplo,

```
(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)
```

7.2.10 Ejercicio: Redefinir RASSOC.

Solución:

```
(DEFUN N-RASSOC (VAL AL)
  (COND ((ATOM AL) NIL)
        ((AND (CONSP AL)
              (EQ VAL (CDAR AL)))
         (CAR AL))
        (T (N-RASSOC VAL (CDR AL)))))
```

7.2.11 Definición: La función SUBLIS:

(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. Por ejemplo,

```
* (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)
```

7.2.12 Ejercicio: Se consideran las siguientes listas:

```
NUMEROS = (1 2 3)
CASTELLANO = (UNO DOS TRES)
INGLES  = (ONE TWO THREE)
FRANCES = (UN DEUX TROIS)
```

Definir las A–listas

```
NUMEROS-CASTELLANO, CASTELLANO-INGLES, CASTELLANO-FRANCES
```

que asocien los primeros elementos con los segundos.

Solución:

```
(SETQ NUMEROS '(1 2 3))              ---> (1 2 3 4 5)
(SETQ CASTELLANO '(UNO DOS TRES))    ---> (UNO DOS TRES)
(SETQ INGLES  '(ONE TWO THREE))      ---> (ONE TWO THREE)
(SETQ FRANCES '(UN DEUX TROIS))      ---> (UN DEUX TROIS)

(SETQ NUMEROS-CASTELLANO (PAIRLIS NUMEROS CASTELLANO))
        ---> ((1 . UNO) (2 . DOS) (3 . TRES))
(SETQ CASTELLANO-INGLES (PAIRLIS CASTELLANO INGLES))
        ---> ((UNO . ONE) (DOS . TWO) (TRES . THREE))
(SETQ CASTELLANO-FRANCES (PAIRLIS CASTELLANO FRANCES))
        ---> ((UNO . UN) (DOS . DEUX) (TRES . TROIS))
```

7.2.13 Ejercicio: Escribir las funciones

```
(EN-CASTELLANO N),  (EN-INGLES N),  (EN-FRANCES N)
```

de forma que si N es un número, devuelva su nombre en el idioma apropiado. Por ejemplo,

```
(EN-CASTELLANO 2)   ---> DOS
(EN-INGLES 2)       ---> TWO
(EN-FRANCES 2)      ---> DEUX
```

Solución:

```
(DEFUN EN-CASTELLANO (N)
  (CDR (ASSOC N NUMEROS-CASTELLANO)) )

(DEFUN EN-INGLES (N)
  (CDR (ASSOC (EN-CASTELLANO N) CASTELLANO-INGLES)) )

(DEFUN EN-FRANCES (N)
  (CDR (ASSOC (EN-CASTELLANO N) CASTELLANO-FRANCES)) )
```

7.2.14 Ejercicio: Definir la función

```
(NOMBRE N I)
```

de forma que si N es un número e I es un idioma, devuelva el nombre de N en el idioma I. Por ejemplo,

```
(NOMBRE 2 'CASTELLANO)    ---> DOS
(NOMBRE 2 'INGLES)        ---> TWO
(NOMBRE 2 'FRANCES)       ---> DEUX
```

Solución:

```
(DEFUN NOMBRE (N I)
  (COND ((EQUAL I 'CASTELLANO) (EN-CASTELLANO N))
        ((EQUAL I 'INGLES) (EN-INGLES N))
        ((EQUAL I 'FRANCES) (EN-FRANCES N))
        (T 'ERROR) ))
```

7.2.15 Ejercicio: Escribir las funciones

```
(VALOR-CASTELLANO S), (VALOR-INGLES S), (VALOR-FRANCES S)
```

de forma que si S es el nombre de un número en el idioma indicado, devuelva su valor. Por ejemplo,

```
(VALOR-CASTELLANO 'DOS)   ---> 2
(VALOR-INGLES 'ONE)       ---> 1
(VALOR-FRANCES 'TROIS)    ---> 3
```

Solución:

```
(DEFUN VALOR-CASTELLANO (N)
  (CAR (RASSOC N NUMEROS-CASTELLANO)) )

(DEFUN VALOR-INGLES (N)
  (VALOR-CASTELLANO (CAR (RASSOC N CASTELLANO-INGLES))) )

(DEFUN VALOR-FRANCES (N)
  (VALOR-CASTELLANO (CAR (RASSOC N CASTELLANO-FRANCES))) )
```

7.2.16 Ejercicio: Definir la función

```
(VALOR N I)
```

de forma que si N es el nombre de un número en el idioma I es un idioma, devuelva su valor. Por ejemplo,

```
(VALOR 'DOS 'CASTELLANO)    ---> 2
(VALOR 'ONE 'INGLES)        ---> 1
(VALOR 'TROIS 'FRANCES)     ---> 3
```

Solución:

```
(DEFUN VALOR (N I)
  (COND ((EQUAL I 'CASTELLANO) (VALOR-CASTELLANO N))
        ((EQUAL I 'INGLES) (VALOR-INGLES N))
        ((EQUAL I 'FRANCES) (VALOR-FRANCES N))
        (T 'ERROR) ))
```

7.2.17 Ejercicio: Definir la función

```
(FRANCES-A-CASTELLANO S)
```

que traduzca números en francés a números en español. Por ejemplo,

```
(FRANCES-A-CASTELLANO 'TROIS) ---> TRES
```

Solución:

```
(DEFUN FRANCES-A-CASTELLANO (N)
  (EN-CASTELLANO (VALOR-FRANCES N)) )
```

Capítulo 8

Las P–listas (listas de propiedades)

8.1 P–listas

8.1.1 Definición: 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.

8.1.2 Nota: 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

8.1.3 Formación de P–listas:

(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. Por ejemplo,

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

8.1.4 Lectura de P–listas:

(SYMBOL-PLIST PL) devuelve la P–lista asociada a PL, si la hay y `NIL`, si no. Por ejemplo, continuando con el anterior,

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

8.1.5 Obtención de valores:

(GET PL IND) devuelve el valor asociado a IND en la P–lista PL. Por ejemplo, continuando con el anterior,

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

8.1.6 Asignación de valores:

(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. Por ejemplo, continuando con el anterior,

```
(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))
```

8.1.7 Eliminación de valores:

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

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

8.1.8 Ejercicio: Escribir tres P–listas JUAN, SOFIA y TOMAS que recojan los siguientes datos:

JUAN	SOFIA	TOMAS
EDAD 35	EDAD 30	EDAD 5
MUJER SOFIA	MARIDO JUAN	PADRE JUAN
HIJO TOMAS	HIJO TOMAS	MADRE SOFIA

Solución:

```
(SETF (SYMBOL-PLIST 'JUAN) '(EDAD 35 MUJER SOFIA HIJO TOMAS))
(SETF (SYMBOL-PLIST 'SOFIA) '(EDAD 30 MARIDO JUAN HIJO TOMAS))
(SETF (SYMBOL-PLIST 'TOMAS) '(EDAD 5 PADRE JUAN MADRE SOFIA))
```

8.1.9 Ejercicio: Consultar la base de datos anterior para obtener:

1. La edad de Tomás.
2. El hijo de Sofía.
3. La mujer del padre de Tomás.
4. La edad del marido de la madre de Tomás.
5. El hijo de la mujer del padre de Tomás.
6. El padre del hijo de Sofía.

Solución:

```
(GET 'TOMAS 'EDAD)                              ---> 5
(GET 'SOFIA 'HIJO)                              ---> TOMAS
(GET (GET 'TOMAS 'PADRE) 'MUJER)                ---> SOFIA
(GET (GET (GET 'TOMAS 'MADRE) 'MARIDO) 'EDAD) ---> 35
(GET (GET (GET 'TOMAS 'PADRE) 'MUJER) 'HIJO)  ---> TOMAS
(GET (GET 'SOFIA 'HIJO) 'PADRE)                 ---> JUAN
```

8.1.10 Ejercicio: Escribir una P–lista HIJOS-DE que recoja el siguiente árbol genealógico:

```
HIJOS DE ANICETO: ANTONIO, PEPE Y RODRIGO
HIJOS DE ANTONIO: ALFONSO Y ARTURO
HIJOS DE PEPE   : GONZALO
HIJOS DE ARTURO : JUAN
```

Solución:

```
(SETF (SYMBOL-PLIST 'HIJOS-DE)
      '(ARTURO (JUAN)
        PEPE (GONZALO)
        ANTONIO (ALFONSO ARTURO)
        ANICETO (ANTONIO PEPE RODRIGO)))
```

8.1.11 Ejercicio: Escribir una lista PADRES que contenga los padres de árbol anterior.

Solución:

```
(SETQ PADRES '(ARTURO PEPE ANTONIO ANICETO))
```

8.1.12 Ejercicio: Definir la función (PADRE X) que devuelva el padre de X. Por ejemplo,

```
                    (PADRE 'GONZALO) ---> PEPE
```

Solución:

```
(PADRE X) = Y <==>
  <==> Y pertenece a PADRES y X es un hijo de Y
  <==> Y pertenece a PADRES y X pertenece a (GET 'HIJOS-DE Y)
  <==> Y pertenece a PADRES y (MEMBER X (GET 'HIJOS-DE Y))
  <==> Y = (SOME '(LAMBDA (Y) (IF (MEMBER X (GET 'HIJOS-DE Y)) Y))
                 PADRES)
(DEFUN PADRE (X)
  (SOME #'(LAMBDA (Y) (IF (MEMBER X (GET 'HIJOS-DE Y)) Y))
        PADRES))
```

Por ejemplo, para calcular (PADRE 'GONZALO)

```
X                                   = GONZALO
PADRES                              = (ARTURO PEPE ANTONIO ANICETO)
Y =                                 = ARTURO PEPE
(GET HIJOS-DE Y)                    = (JUAN) (GONZALO)
(MEMBER X (GET 'HIJOS-DE Y))        = NIL (GONZALO)
```

Definición iterativa de PADRE:

```
(DEFUN PADRE (X)
  (PROG ((AUX PADRES))
    ETIQUETA
    (COND ((NULL AUX) (RETURN NIL))
          ((MEMBER X (GET 'HIJOS-DE (CAR AUX)))
           (RETURN (CAR AUX))))
    (SETQ AUX (CDR AUX))
    (GO ETIQUETA)))
```

Por ejemplo, para calcular (PADRE 'GONZALO)

```
Iteracion AUX                           (MEMBER 'GONZALO (GET...)
   1      (ARTURO PEPE ANTONIO ANICETO) NIL
   2      (PEPE ANTONIO ANICETO)        (GONZALO)
```

Definición recursiva de PADRE

```
(DEFUN PADRE (X)
  (PADRE-AUX X PADRES))

(DEFUN PADRE-AUX   (X L)
  (COND ((NULL L) NIL)
        ((MEMBER X (GET 'HIJOS-DE (CAR L))) (CAR L))
        (T (PADRE-AUX X (CDR L)))))
```

Por ejemplo, para calcular (PADRE 'GONZALO)

```
(PADRE 'GONZALO) =
   = (PADRE-AUX 'GONZALO '(ARTURO PEPE ANTONIO ANICETO))
   = (PADRE-AUX 'GONZALO '(PEPE ANTONIO ANICETO))
   = PEPE
```

8.1.13 Ejercicio: Definir la función (ABUELO X) que devuelva el abuelo de X. Por ejemplo,

```
(ABUELO 'JUAN) ---> ANTONIO
(ABUELO 'PEPE) ---> NIL
```

Solución:

```
(DEFUN ABUELO (X)
  (PADRE (PADRE X)))
```

8.1.14 Ejercicio: Definir la función (DECANO X) que devuelva el primer antepasado de X. Por ejemplo,

```
(DECANO 'JUAN) ---> ANICETO
(DECANO 'LUIS) ---> LUIS
```

Solución:

```
(DEFUN DECANO (X)
  (LET ((AUX (PADRE X)))
    (COND ((NULL AUX) X)
          (T (DECANO AUX)))))
```

8.1.15 Ejercicio: Definir la función (ANCESTROS X) que devuelva la lista de los antepasados de X. Por ejemplo,

```
(ANCESTROS 'JUAN) ---> (JUAN ARTURO ANTONIO ANICETO)
```

Solución:

```
(DEFUN ANCESTROS (X)
  (COND ((NULL (PADRE X)) (LIST X))
        (T (CONS X (ANCESTROS (PADRE X)))))))
```

Por ejemplo,

```
(ANCESTROS 'GONZALO) =
   = (CONS 'GONZALO (CONS 'PEPE (ANCESTROS 'ANICETO))) =
   = (CONS 'GONZALO (CONS 'PEPE '(ANICETO)) =
   = (GONZALO PEPE ANICETO)
```

Para evitar la duplicidad del cálculo (PADRE X), podemos redefinir ANCESTROS por:

```
(DEFUN ANCESTROS (X)
  (LET ((AUX (PADRE X)))
    (COND ((NULL AUX) (LIST X))
          (T (CONS X (ANCESTROS AUX))))))
```

8.1.16 Ejercicio: Utilizar la función ANCESTROS para redefinir la función DECANO.

Solución:

```
(DEFUN DECANO (X)
  (CAR (LAST (ANCESTROS X))))
```

8.1.17 Ejercicio: Definir la función (HERMANOS X) que devuelva la lista de los hermanos de X. Por ejemplo,

```
(HERMANOS 'PEPE) ---> (ANTONIO RODRIGO)
(HERMANOS 'JUAN) ---> NIL
```

Solución:

```
(DEFUN HERMANOS (X)
  (REMOVE X (GET 'HIJOS-DE (PADRE X))))
```

Por ejemplo,

```
(HERMANOS 'PEPE) =
   = (REMOVE 'PEPE (GET 'HIJOS-DE (PADR 'PEPE))) =
   = (REMOVE 'PEPE (GET 'HIJOS-DE 'ANICETO)) =
   = (REMOVE 'PEPE '(ANTONIO PEPE RODRIGO)) =
   = (ANTONIO RODRIGO)
```

8.1.18 Ejercicio: Definir la función (TIOS X) que devuelva la lista de los tíos de X. Por ejemplo,

```
(TIOS 'GONZALO) ---> (ANTONIO RODRIGO)
```

Solución:

```
(DEFUN TIOS (X)
  (HERMANOS (PADRE X)))
```

8.2 Bases de datos

8.2.1 Ejercicio: Deseamos construir una base de datos bibliográfica. Para ello, utilizamos las P–listas LIBRO-N que contiene el autor, título y editorial de libro de referencia N y la variable global LIBROS que contiene las P–listas almacenadas. Por ejemplo, si nuestra base comienza por el libro (FARRENY: "Programmer en LISP", ed. MASSON) escribimos

```
(SETF (SYMBOL-PLIST 'LIBRO-1)
      '(AUTOR "FARRENY"
      TITULO "Programmer en LISP"
      EDITORIAL "Masson"))

(SETQ LIBROS '(LIBRO-1))
```

Definir la función

```
(ADD-LIBRO REFERENCIA AUTOR TITULO EDITORIAL)
```

que añada el libro especificado a la base de datos y devuelva REFERENCIA. Por ejemplo,

```
(ADD-LIBRO 'LIBRO-2
           "WINSTON Y HORN"
           "LISP"
           "ADDISON WESLEY") ---> LIBRO-2
LIBROS                       ---> (LIBRO-2 LIBRO-1)
(GET 'LIBRO-2 'TITULO)       ---> "LISP"
```

Solución:

```
(DEFUN ADD-LIBRO (REFERENCIA AUTOR TITULO EDITORIAL)
  (SETF (GET REFERENCIA 'AUTOR) AUTOR
        (GET REFERENCIA 'TITULO) TITULO
        (GET REFERENCIA 'EDITORIAL) EDITORIAL)
  (SETQ LIBROS (CONS REFERENCIA LIBROS))
  REFERENCIA)
```

8.2.2 Ejercicio: Definir la función (BUSCAR-POR PROPIEDAD VALOR) que busque en la base de datos LIBROS y dé la lista de las referencias de los libros que verifican las condiciones. Por ejemplo,

```
(BUSCAR-POR 'AUTOR "FARRENY") ---> (LIBRO-1)
(BUSCAR-POR 'TITULO "LISP")   ---> (LIBRO-2)
```

Solución:

```
(DEFUN BUSCAR-POR (PROPIEDAD VALOR)
  (PROG ((AUX LIBROS)
         (RESULTADO NIL))
    ETIQUETA
    (COND ((NULL AUX) (RETURN RESULTADO))
          ((EQUAL (GET (CAR AUX) PROPIEDAD) VALOR)
           (SETQ RESULTADO (CONS (CAR AUX) RESULTADO))))
    (SETQ AUX (CDR AUX))
    (GO ETIQUETA)))
```

8.3 Programación dirigida por los datos

8.3.1 Definición: La función EVAL:

(EVAL S) devuelve el valor de S. Por ejemplo,

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

8.3.2 Ejercicio: Definir la función CALCULAR que permita sumar o restar números reales o complejos; es decir,

```
(CALCULAR 'A 'MAS B)            = A + B
(CALCULAR '(A B) 'MAS '(C D)    = (A + C  B + D)
(CALCULAR 'A 'MENOS 'B)         = A - B
(CALCULAR '(A B) 'MENOS '(C D) = (A - C  B - D)
```

Por ejemplo,

```
(CALCULAR 2 'MAS 3)              ---> 5
(CALCULAR '(5 4) 'MENOS '(2 0)) ---> (3 4)
```

Solución:

```
(DEFUN CALCULAR (OP1 OPR OP2)
   (EVAL (GET (IF (ATOM OP1) 'REAL 'COMPLEJO) OPR)))

(SETF (SYMBOL-PLIST 'REAL)
      '(MAS (+ OP1 OP2)
        MENOS (- OP1 OP2)))

(SETF (SYMBOL-PLIST 'COMPLEJO)
      '(MAS (LIST (+ (CAR OP1) (CAR OP2))
                  (+ (CADR OP1) (CADR OP2)))
        MENOS (LIST (- (CAR OP1) (CAR OP2))
                    (- (CADR OP1) (CADR OP2)) )))
```

Por ejemplo,

```
(CALCULAR 2 'MAS 3) =
   = (EVAL (GET (IF (ATOM 2) 'REAL 'COMPLEJO) 'MAS)) =
   = (EVAL (GET 'REAL 'MAS)) =
   = (EVAL '(+ 2 3)) =
   = 5
```

8.3.3 Ejercicio: Ampliar la anterior definición par que acepte la multiplicación real y compleja; es decir,

```
(CALCULAR 'A 'POR 'B)          = AB
(CALCULAR '(A B) 'POR '(C D)) = (AC - BD  AD + BC)
```

Por ejemplo,

```
(CALCULAR 2 'POR 3)            ---> 6
(CALCULAR '(1 2) 'POR '(3 4)) ---> (-5 10)
```

Solución:

```
(SETF (GET 'REAL 'POR)
      '(* OP1 OP2)
      (GET 'COMPLEJO 'POR)
      '(LIST (- (* (CAR OP1) (CAR OP2))
                (* (CADR OP1) (CADR OP2)))
             (+ (* (CAR OP1) (CADR OP2))
                (* (CADR OP1) (CAR OP2)) )))
```

8.4 Funciones con memoria

8.4.1 Ejercicio: Redefinir FACTORIAL como función con memoria.

Solución:

```
(DEFUN FACTORIAL (N)
  (COND ((= N 1) 1)
        ((GET 'FACTORIAL N))
        (T (SETF (GET 'FACTORIAL N) (* N (FACTORIAL (1- N))))) ))
```

Por ejemplo,

```
(FACTORIAL 4) =
   = (SETF (GET 'FACTORIAL 4)
           (* 4 (FACTORIAL 3))) =
   = (SETF (GET 'FACTORIAL 4)
           (* 4 (SETF (GET 'FACTORIAL 3)
                      (* 3 (FACTORIAL 2))))) =
   = (SETF (GET 'FACTORIAL 4)
           (* 4 (SETF (GET 'FACTORIAL 3)
                      (* 3 (* 2 (FACTORIAL 1)))))) =
```

```
= (SETF (GET 'FACTORIAL 4)
        (* 4 (SETF (GET 'FACTORIAL 3)
                   6))) =
= 24
```

Además, ha creado la P-lista FACTORIAL. Podemos verla mediante

```
(SYMBOL-PLIST 'FACTORIAL) ---> (4 24 3 6 2 2)
```

o, consultarla, mediante

```
(GET 'FACTORIAL 3) ---> 6
```

Cuando se repite la llamada a factorial no repite los cálculos. Por ejemplo,

```
(FACTORIAL 3) = 6

(FACTORIAL 5) =
   = (SETF (GET 'FACTORIAL 5) (* 5 (FACTORIAL 4))) =
   = (SETF (GET 'FACTORIAL 5) (* 5 24)) =
   = 120
```

8.4.2 Ejercicio: Mediante TRACE, comprobar el número de llamadas a la función FACTORIAL cuando se calcula (FACTORIAL 7) y (FACTORIAL 10)

Solución: Si la P-lista FACTORIAL estaba vacía, al calcular (FACTORIAL 7) llama 7 veces a la función FACTORIAL y al calcular a continuación (FACTORIAL 10) sólo llama 4 veces a la función FACTORIAL.

8.4.3 Definición: La función TIME:

(TIME S) devuelve el tiempo empleado en calcular la expresión S. Por ejemplo,

```
* (DEFUN FACT (N) (IF (= N 0) 1 (* N (FACT (1- N)))))
FACT
* (TIME (FACT 50.0))
Evaluating: (FACT 50.0)
Elapsed time: 0:00.11

6.80565F+38
```

8.4.4 Ejercicio: Comparar el tiempo empleado en calcular los factoriales de 100, 150 y 200 con las definiciones anteriores.

8.4.5 Ejercicio: La función de FIBONACCI está definida por

$$\texttt{FIB (N)} = \begin{cases} \texttt{1} & \text{si N = 0 ó N = 1} \\ \texttt{FIB(N-1) + FIB(N-2)} & \text{en otro caso} \end{cases}$$

Los primeros términos de la sucesion son 1, 1, 2, 3, 5, 8, 13, 21,... Escribir la función

```
(FIB N)
```

que devuelva el término N-ésimo de la función de FIBONACCI. Por ejemplo,

```
                              (FIB 6) ---> 13
```

Solución:

```
(DEFUN FIB (N)
  (COND ((= N 0) 1)
        ((= N 1) 1)
        (T (+ (FIB (- N 1)) (FIB (- N 2))))))
```

8.4.6 Ejercicio: Redefinir la función de FIBONACCI como función con memoria.

Solución:

```
(DEFUN FIB (N)
  (COND ((= N 0) 1)
        ((= N 1) 1)
        ((GET 'FIB N))
        (T (SETF (GET 'FIB N)
                 (+ (FIB (- N 1)) (FIB (- N 2))) ))))
```

8.4.7 Ejercicio: Comparar mediante TRACE y TIME las dos definiciones anteriores de la función de FIBONACCI.

8.4.8 Ejercicio: Escribir la función de 91 de McCarthy

$$\texttt{F91(N)} = \begin{cases} \texttt{N - 10} & \text{si } \texttt{N} > 100 \\ \texttt{F91(F91(N + 11))} & \text{si } 0 \leq \texttt{N} \leq 100 \end{cases}$$

Solución:

```
(DEFUN F91 (N)
  (COND ((> N 100) (- N 10))
        (T (F91 (F91 (+ N 11))) )))

; DEFINICION CON MEMORIA

(DEFUN F91 (N)
  (COND ((> N 100) (- N 10))
        ((GET 'F91 N))
        (T (SETF (GET 'F91 N)
                 (F91 (F91 (+ N 11))) ))))
```

```
; DEFINICION NO RECURSIVA

(DEFUN F91 (N)
  (COND ((> N 100) (- N 10))
        (T 91) ))
```

[Nota: No siempre la definición con memoria es la más inteligente]

8.4.9 Ejercicio: La función de ACKERMANN está definida por

$$\texttt{ACK(M, N)} = \begin{cases} \texttt{N + 1} & \text{si } \texttt{M = 0} \\ \texttt{ACK(M - 1, 1)} & \text{si } \texttt{M} > \texttt{0 y N = 0} \\ \texttt{ACK(M - 1, ACK(M, N-1))} & \text{en otro caso} \end{cases}$$

Escribir la definición de (ACK M N) y calcular

```
(ACK 3 4)
(TIME '(ACK 3 4))
```

Solución:

```
(DEFUN ACK (M N)
  (COND ((= M 0) (1+ N))
        ((= N 0) (ACK (1- M) 1))
        (T (ACK (1- M) (ACK M (1- N))))))

(ACK 3 4)          ---> 125
(TIME (ACK 3 4))   ---> 0:07.58 (en un AT)
```

Capítulo 9

Lectura y escritura

9.1 Funciones de lectura y escritura

9.1.1 Definición: La función PRINT:

(PRINT S) escribe en una nueva línea el valor de S y devuelve dicho valor.

9.1.2 Ejemplos:

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

9.1.3 Ejercicio: Escribir una función IMPRIME-TODO tal que al aplicarla a una lista L escriba los elementos de L uno en cada línea. Por ejemplo,

```
* (IMPRIME-TODO '(A (B C) D))
A
(B C)
D
NIL
```

Solución:

```
(DEFUN IMPRIME-TODO (L)
  (COND ((NULL L) NIL)
        (T (PRINT (CAR L))
           (IMPRIME-TODO (CDR L)) )))
```

9.1.4 Definición: Las funciones PRIN1 y PRINC:

(PRIN1 S) escribe el valor de S, usando caracteres de control.

(PRINC S) escribe el valor de S, sin usar caracteres de control.

9.1.5 Ejemplo:

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

9.1.6 Definición: La función TERPRI:

(TERPRI) deja una línea en blanco. Por ejemplo,

```
* (DEFUN F ()
    (TERPRI)
    (PRINC "LINEA ")
    (PRIN1 1)
    (TERPRI)
    (PRINC "LINEA ")
    (PRIN1 2)
    'FIN)
F
* (F)
LINEA 1
LINEA 2
FIN
```

9.1.7 Ejercicio: Definir una función TAB-CUAD tal que (TAB-CUAD N) imprima la lista de los cuadrados de 1 a N. Por ejemplo,

```
* (TAB-CUAD 3)
EL CUADRADO DE 1 ES 1
EL CUADRADO DE 2 ES 4
EL CUADRADO DE 3 ES 9
NIL
```

Solución:

```
(DEFUN TAB-CUAD (N)
  (PROG (AUX)
    (SETQ AUX 1)
    ETIQUETA
```

```
(COND ((> AUX N) (RETURN))
      (T (TERPRI)
         (PRINC "EL CUADRADO DE ")
         (PRINC AUX)
         (PRINC " ES ")
         (PRINC (* AUX AUX))
         (SETQ AUX (1+ AUX))
         (GO ETIQUETA) ))))
```

9.1.8 Definición: La función READ:

(READ) detiene la ejecución del programa hasta que se le introduce un dato. Por ejemplo,

```
* (READ)
'A

(QUOTE A)
* (SETQ A (READ))
2

2
* A

2
* (+ (READ) (READ))
3
4

7
```

9.1.9 Ejercicio: Definir una función CALCULA-CUADRADOS tal que

1. lea una expresión S;
2. si el valor de S es un número N, devuelva el cuadrado de N y vuelve a 1;
3. si el valor de S no es un número, escriba FIN y pare.

Por ejemplo,

```
* (CALCULA-CUADRADOS)
```

```
EL CUADRADO DE 3
ES 9

EL CUADRADO DE 5
ES 25

EL CUADRADO DE A
FIN

NIL
```

Solución:

```
(DEFUN CALCULA-CUADRADOS ()
  (PROG (AUX)
    ETIQUETA
    (TERPRI)
    (PRINC "EL CUADRADO DE ")
    (SETQ AUX (READ))
    (COND ((NOT (NUMBERP AUX)) (PRINC "FIN") (TERPRI))
          (T (PRINC "ES ")
             (PRINC (* AUX AUX))
             (TERPRI)
             (GO ETIQUETA) ))))
```

9.1.10 Definición: La función FORMAT:

(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

9.1.11 Ejemplo:

```
* (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, EN OCTAL ES ~O Y EN HEXADECIMAL ES ~X"
          10 10 10)
10 EN BINARIO ES 1010, EN OCTAL ES 12 Y EN HEXADECIMAL ES A
NIL
```

9.1.12 Ejercicio: Definir la función (CUADRADOS M N) que escriba la lista de los cuadrados desde M hasta N. Por ejemplo,

```
* (CUADRADOS 2 5)
EL CUADRADO DE 2 ES 4
EL CUADRADO DE 3 ES 9
EL CUADRADO DE 4 ES 16
EL CUADRADO DE 5 ES 25
FIN
```

Solución:

```
(DEFUN CUADRADOS (M N)
  (DO ((CONT M (1+ CONT)))
      ((> CONT N) 'FIN)
    (FORMAT T
            "~%EL CUADRADO DE ~D ES ~D"
            CONT (* CONT CONT)) ))
```

9.2 Funciones de lectura y escritura sobre ficheros

9.2.1 Apertura de un fichero:

(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.

9.2.2 Cierre de un fichero:

(CLOSE SIMB) cierra el canal nombrado por SIMB.

9.2.3 Ejemplo:

```
* (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
```

9.2.4 Definición: La función WITH-OPEN-FILE:

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

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:

```
(READ SIMB)
```

Si la :CLAVE es :OUTPUT, las expresiones SI pueden ser de escritura:

```
(FORMAT SIMB S)
(PRINT S SIMB)
(PRINC S SIMB)
(TERPRI SIMB)
```

9.2.5 Ejemplo:

```
* (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
```

9.2.6 Ejercicio: Supongamos que el contenido del fichero NOTAS.DAT es:

```
((AGUADO CASAS) (JUAN JOSE) 5)
((AGUILA PUENTE) (RAFAEL) 2)
((ALBA ADAME) (CARLOS) 9)
((ALCALDE PEREZ) (MARIA LUISA) 3)
```

Definir la función que LEE que lea el contenido del fichero. Por ejemplo:

```
* (LEE)
((AGUADO CASAS) (JUAN JOSE) 5)
((AGUILA PUENTE) (RAFAEL) 2)
((ALBA ADAME) (CARLOS) 9)
((ALCALDE PEREZ) (MARIA LUISA) 3)

NIL
```

Solución:

```
(DEFUN LEE ()
  (WITH-OPEN-FILE (NOTAS "NOTAS.DAT" :DIRECTION :INPUT)
     (DO ((ALUMNO (READ NOTAS NIL)
                  (READ NOTAS NIL) ))
         ((NOT ALUMNO))
       (TERPRI)
       (PRINC ALUMNO) )))
```

9.2.7 Ejercicio: Definir la función APROBADOS que lea el fichero NOTAS.DAT y escriba en el fichero APROBADOS.DAT la lista de aprobados.

Solución:

```
(DEFUN APROBADOS ()
  (WITH-OPEN-FILE (NOTAS "NOTAS.DAT" :DIRECTION :INPUT)
     (WITH-OPEN-FILE (APROBADOS "APROBADOS.DAT" :DIRECTION :OUTPUT)
        (DO ((ALUMNO (READ NOTAS NIL)
                     (READ NOTAS NIL) ))
            ((NOT ALUMNO))
            (WHEN (APROBADO-P ALUMNO)
              (PRINC ALUMNO APROBADOS)
              (TERPRI APROBADOS) )))))

(DEFUN APROBADO-P (ALUMNO)
  (>= (CADDR ALUMNO) 5) )
```

9.2.8 Ejercicio: Supongamos que tenemos una lista

```
(SETQ AGENDA '((PEPE SIERPES-12   "4531420")
               (JUAN CUNA-5       "4243568") ))
```

Escribir las siguientes funciones:

(BUSCAR NOMBRE) Devuelve (NOMBRE DIRECCION TELEFONO), si
NOMBRE está en AGENDA y NIL en caso contrario.

(CAMBIAR-DIR NOMBRE NUEVA-DIR) cambia la dirección de NOMBRE.

(CAMBIAR-TEL NOMBRE NUEVO-TEL) cambia el teléfono de NOMBRE.

(PONER NOMBRE DIRECCION TELEFONO) añade a la AGENDA (NOMBRE DIRECCION TELEFONO).

(QUITAR NOMBRE) quita (NOMBRE DIRECCION TELEFONO) a AGENDA.

(CERRAR) abre el fichero AGENDA.DAT, guarda el nuevo valor de AGENDA y lo cierra.

(ABRIR) abre el fichero AGENDA.DAT, lee el valor de AGENDA y lo cierra.

(AGENDA) presenta un menú para controlar el programa.

Por ejemplo,

```
* (BUSCAR 'PEPE)
(PEPE SIERPES-12 "4531420")
* (CAMBIAR-DIR 'PEPE 'TARFIA-SN)
(PEPE TARFIA-SN "4531420")
* (BUSCAR 'PEPE)
(PEPE TARFIA-SN "4531420")
* (CAMBIAR-TEL 'PEPE "4615600")
(PEPE TARFIA-SN "4615600")
* (BUSCAR 'PEPE)
(PEPE TARFIA-SN "4615600")
* (QUITAR 'PEPE)
((JUAN CUNA-5 "4243568"))
* (PONER 'PEPE 'TARFIA-SN "4615600")
((PEPE TARFIA-SN "4615600") (JUAN CUNA-5 "4243568"))
* (CERRAR)
NIL
* AGENDA


ERROR:
Unbound variable: AGENDA
1> <Control C>
* (ABRIR)
* AGENDA
((PEPE SIERPES-12 "4531420")(JUAN CUNA-5 "4243568")))
```

```
* ;;; EJEMPLO DE SESION:

* (AGENDA)
   1: LEER BASE DE DATOS        5: QUITAR NOMBRE

   2: BUSCAR UN NOMBRE          6: PONER NOMBRE

   3: CAMBIAR DIRECCION         7: GRABAR BASE DE DATOS

   4: CAMBIAR TELEFONO          8: SALIR


ESCRIBE OPCION: 1

ESCRIBE OPCION: 6

ESCRIBE NOMBRE: JUAN

ESCRIBE DIRECCION: TARFIA-SN

ESCRIBE TELEFONO: 4616500

ESCRIBE OPCION: 6

ESCRIBE NOMBRE: PEPE

ESCRIBE DIRECCION: CUNA-12

ESCRIBE TELEFONO: 4227700

ESCRIBE OPCION: 2

ESCRIBE EL NOMBRE: JUAN

NOMBRE:    JUAN
DIRECCION: TARFIA-SN
TELEFONO:  4616500

ESCRIBE OPCION: 7

ESCRIBE OPCION: 8
```

```
"FIN"
```

Solución:

```
(DEFUN BUSCAR (NOMBRE)
  (BUSCAR-AUX NOMBRE AGENDA) )

(DEFUN BUSCAR-AUX (NOMBRE LISTA)
  (COND ((NULL LISTA) NIL)
        ((EQUAL NOMBRE (CAAR LISTA)) (CAR LISTA))
        (T (BUSCAR-AUX NOMBRE (CDR LISTA))) ))

(DEFUN CAMBIAR-DIR (NOMBRE NUEVA-DIR)
  (LET ((AUX (BUSCAR NOMBRE)))
    (IF AUX
        (SETQ AGENDA (SUBST (LIST NOMBRE NUEVA-DIR (CADDR AUX))
                    AUX
                    AGENDA )))
    (BUSCAR NOMBRE) ))

(DEFUN CAMBIAR-TEL (NOMBRE NUEVO-TEL)
  (LET ((AUX (BUSCAR NOMBRE)))
    (IF AUX
        (SETQ AGENDA (SUBST (LIST NOMBRE (CADR AUX) NUEVO-TEL)
                    AUX
                    AGENDA )))
    (BUSCAR NOMBRE) ))

(DEFUN QUITAR (NOMBRE)
  (SETQ AGENDA (DELETE (BUSCAR NOMBRE) AGENDA)) )

(DEFUN PONER (NOMBRE DIRECCION TELEFONO)
  (IF (BOUNDP 'AGENDA)
      (SETQ AGENDA (CONS (LIST NOMBRE DIRECCION TELEFONO) AGENDA))
      (SETQ AGENDA (LIST (LIST NOMBRE DIRECCION TELEFONO))) ))

(DEFUN CERRAR ()
  (LET ((AUX (OPEN "AGENDA.DAT" :DIRECTION :OUTPUT)))
    (PRIN1 AGENDA AUX)
    (MAKUNBOUND 'AGENDA)
```

```
    (CLOSE AUX) ))

(DEFUN ABRIR ()
  (LET ((AUX (OPEN "AGENDA.DAT" :DIRECTION :INPUT)))
    (SETQ AGENDA (READ AUX))
    (CLOSE AUX) ))

(DEFUN MENU ()
  (FORMAT T
          "~%   1: LEER BASE DE DATOS        5: QUITAR NOMBRE
           ~%   2: BUSCAR UN NOMBRE          6: PONER NOMBRE
           ~%   3: CAMBIAR DIRECCION         7: GRABAR BASE DE DATOS
           ~%   4: CAMBIAR TELEFONO          8: SALIR
           ~%") )
```

```
(DEFUN AGENDA ()
  (SEND *TERMINAL-IO* :CLEAR-SCREEN)            ;LIMPIA LA PANTALLA
  (MENU)
  (SETQ AGENDA NIL)
  (LOOP
    (FORMAT T "~%ESCRIBE OPCION: ")
    (LET ((AUX (READ)))
      (SEND *TERMINAL-IO* :CLEAR-SCREEN)
      (MENU)
      (COND ((EQUAL AUX 1) (ABRIR) )
            ((EQUAL AUX 2)
             (FORMAT T "~%ESCRIBE EL NOMBRE: ")
             (SETQ PP (BUSCAR (READ)))
             (IF PP (FORMAT T
                      "~%NOMBRE:    ~A ~%DIRECCION: ~A ~%TELEFONO:  ~A"
                      (CAR PP) (CADR PP) (CADDR PP))
                    (FORMAT T "~%NO ESTA EN LA AGENDA") )
             (MAKUNBOUND 'PP)
             (TERPRI) )
            ((EQUAL AUX 3)
             (FORMAT T "~%ESCRIBE NOMBRE: ")
             (SETQ NOMBRE (READ))
             (FORMAT T "~%ESCRIBE NUEVA DIRECCION: ")
             (CAMBIAR-DIR NOMBRE (READ))
             (MAKUNBOUND NOMBRE) )
            ((EQUAL AUX 4)
             (FORMAT T "~%ESCRIBE NOMBRE: ")
             (SETQ NOMBRE (READ))
             (FORMAT T "~%ESCRIBE NUEVO NUMERO DE TELEFONO: ")
             (CAMBIAR-TEL NOMBRE (READ-LINE))
             (MAKUNBOUND NOMBRE) )
            ((EQUAL AUX 5)
             (FORMAT T "~%ESCRIBE NOMBRE: ")
             (QUITAR (READ)) )
            ((EQUAL AUX 6)
             (FORMAT T "~%ESCRIBE NOMBRE: ")
             (SETQ NOMBRE (READ))
             (FORMAT T "~%ESCRIBE DIRECCION: ")
             (SETQ DIRECCION (READ))
             (FORMAT T "~%ESCRIBE TELEFONO: ")
             (SETQ TELEFONO (READ-LINE))
```

```
 (PONER NOMBRE DIRECCION TELEFONO)
 (MAKUNBOUND 'NOMBRE)
 (MAKUNBOUND 'DIRECCION)
 (MAKUNBOUND 'TELEFONO) )
((EQUAL AUX 7) (CERRAR) )
((EQUAL AUX 8) (SEND *TERMINAL-IO* :CLEAR-SCREEN)
               (RETURN "FIN") )))))
```

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

Indice