Lenguaje de Programación LISP

Toma su nombre del inglés: **Lis**t **P**rocessing
Los tipos de datos básicos con los que se trabaja en LISP son: átomos y listas

Átomo: es un elemento indivisible que tiene significado propio.

Ej.: 54 , 3.14 , + , lunes , sol
Los átomos 54 y 3.14 son átomos numéricos y los otros son átomos simbólicos.

Lista: consta de un paréntesis izquierdo, seguida por cero o más átomos o listas, y un paréntesis derecho. Cada componente de la lista se denomina elemento.

Ej.: (a 23 (8 hola ( )) 3.1) lista con 4 elementos

La lista nula tiene la característica de ser lista y átomo a la vez , y se representa por:
( ) o bien NIL.
Los valores de verdad están representados por el átomo T y el átomo NIL (que representan
el verdadero y falso respectivamente.

Lisp también tiene otros tipos de datos como ser caracteres, arreglos, cadenas y estructuras.

La notación en LISP es prefija. Por ej.: (+ 2 3) para sumar 2 + 3

Funciones primitivas

-Selectores de elementos de una lista

1) **CAR** o bien **FIRST** aplicado a una lista devuelve el primer elemento de la lista

Ej.: (CAR '(1 2 3)) → 1

(FIRST '((a b) 8 (6))) → (a b)

2) **CDR** o bien **REST** aplicado a una lista devuelve la lista original sin el primer elem

Ej.: (CDR '(1 2 3)) → (2 3)

(REST '((a b) 8 (6))) → (8 (6))

3) **NTH** aplicado a una lista y un número devuelve el elemento ubicado en la posición indicada por el número

Ej.: (NTH 3 ‘(a b c d)) → c

- Función QUOTE

La función QUOTE evita la evaluación de su argumento devolviéndolo sin evaluar.

(quote (a b c)) → (a b c)

Se puede usar el apóstrofo como abreviatura para QUOTE . Por lo tanto la expresión que sigue es equivalente a la anterior.

‘(a b c) → (a b c)

El hecho de proporcionar el apóstrofo como una abreviatura para QUOTE, se considera una manera de *refinamiento sintáctico*.

- Constructores de listas

1) (**CONS** <obj> <lista>) devuelve una nueva lista donde el primer elemento es el objeto y los elementos restantes son los de la lista original

Ej.: (CONS ‘a ‘(1 2 3)) → (a 1 2 3)

2) (**APPEND** <lista> <lista>) devuelve una nueva lista uniendo los elementos de de las listas argumento

Ej.: (APPEND ‘(a b) ‘(1 2 3)) → (a b 1 2 3)

3) (**LIST** <obj1> <obj2> ... <objn>) devuelve una nueva lista cuyos elementos son los argumentos.

Ej.: (LIST ‘a ‘b ‘(1 2 3)) → (a b (1 2 3))

Ejemplo que muestra la diferencia entre las tres:

(CONS '(a b) '(1 2 3)) → ((a b) 1 2 3)

(APPEND '(a b) '(1 2 3)) → (a b 1 2 3)

(LIST '(a b) '(1 2 3)) → ((a b) (1 2 3))

- Reconocedores de objetos

1) ATOM predicado que verifica si su argumento es un átomo
2) SYMBOLP predicado que verifica si su argumento es un átomo no numérico
3) NUMBERP predicado que verifica si su argumento es un átomo numérico
4) LISTP predicado que verifica si su argumento es una lista
5) NULL predicado que verifica si su argumento es la lista nula
6) LENGTH cuenta el número de elementos del nivel superior que hay en la lista

- Funciones aritméticas : + , - , * , / , **expt** (potencia) , **rem** (resto) , **sqrt** (raiz cuad)

- Funciones booleanas : **AND** , **OR** , **NOT**

- Funciones relacionales: **eq** , < (lt) , > (gt)

Expresiones condicionales

1) (**IF** <expr1> <expr2> <expr3>)

El valor del condicional IF es el valor de la <expr2> si el valor de <expr1> es T o bien
el valor de la <expr3> si el valor de la <expr1> es NIL.

Ej.: (IF (> a b) (CAR X) (+ a b))

2) (**COND** **(** <expr11> <expr12> **)**
(<expr21> <expr22> **)**
...
(<exprn1> <exprn2> **)**
)

El valor del condicional COND será:
el valor de la <expr12> si el valor de la <expr11> es T
el valor de la <expr22> si el valor de la <expr21> es T y <expr11> es NIL
...
el valor de la <exprn2> si el valor de la <exprn1> es T y todas las <exprx1> anteriores son NIL

Ej.: (COND **(** (> a b) (CAR X) **)**
(T (+ a b) **)**
)

3) (**CASE <expr>**
(<cte1> <expr1> **)**
(<cte2> <expr2> **)**
...
[(otherwise <exprN> **)]**
)

El valor de la **<expr>** siempre debe ser un átomo y las ctes deben ser átomos o listas.
El valor del CASE será:

- el valor de la <expr1> si el valor de la <**expr**> es igual (eql) a la <cte1> o bien el valor de la <**expr**> pertenece a la lista <cte1>
- el valor de la <expr2> si el valor de la <**expr**> es igual (eq) a la <cte2> o bien el valor de la <**expr**> pertenece a la lista <cte2> y <**expr**> no es igual o pertenece a <cte1>
- ...
- el valor de la <exprN> si la última cte es otherwise o bien T

Ej.:

```
> (case 3
   (1  'a)
   (2  'b)
   (3  'c)
   (t  'z))
C
> (case 'hola
   ((uno hola dos) 8)
   ((tres cuatro) 9)
   ((cinco seis siete) 10)
```

```
    (otherwise 22)
)
8
> (case 'que
    ((uno hola dos) 8)
    ((tres cuatro) 9)
    ((cinco seis siete) 10)
)
NIL
```

Lectura y escritura

(**PRINT** <expresión>) evalúa su argumento y lo imprime en una nueva línea, seguida de un espacio en blanco y el valor devuelto por el print es el valor de su argumento

Ej.: (print ‘(a b c)) → (a b c) ; acción del print
(a b c) ; valor devuelto por la forma print

Ej.: (print “hoy es martes”) → “hoy es martes”
“hoy es martes”

Ej.: (+ (print 6) (print 8)) → 6
8
14

Ej.: (print (print 6)) → 6
6
6

(**FORMAT** t < cadena >) imprime la cadena en la terminal t y devuelve NIL
En lugar de t puede haber un símbolo que conecte FORMAT con un archivo de salida.

```
> (format t “hoy es martes”)
hoy es martes
nil
```

Para insertar un salto de línea se coloca un tilde~ seguido de una directiva %

```
> (format t "hoy ~%es martes")
hoy
```

es martes
NIL

> > (list (format t "hola ") (format t "que tal"))
hola que tal
(NIL NIL)

> (list (print "hola ") (print "que tal"))
"hola "
"que tal"
("hola " "que tal")

Existe también la directiva A que indica que se debe insertar en la cadena de salida el argumento adicional que aparece después de la cadena de caracteres del FORMAT

>(setq valor 5)
5
> (format t "el resultado de ~a por 2 es ~a" valor (* valor 2))
el resultado de 5 por 2 es 10
NIL

La forma de imprimir en columnas tabuladas es insertar un número de ancho de columna dentro de la directiva A. Por ej. 10A le indica a Format que va a alinear la información a izquierda dentro de un espacio de 10 posiciones.

> (format t "~5a*~5a=~10a*.*.*" 3 4 (* 3 4))
3 *4 =12 *.*.*
NIL

(**READ**) para ingreso de datos por teclado. El valor que devuelve read es lo ingresado por teclado

> (read)
10 ; valor ingresado por teclado
10 ; valor devuelto por el read

> (read)
cdr
CDR

> (+ (read) (read))
2
3

ARCHIVOS

Para grabar en un archivo primero hay que abrir el archivo con open en modo escritura y luego asociar el stream, que devuelve el open, con un nombre utilizando setq.
Luego se usa la función print para grabar los valores en dicho archivo. Al finalizar la grabación de los datos se debe cerrar el archivo con la función close.

Las siguientes funciones graban en un archivo los números 1, 2 y 3

```
(setq     arch    (open "archivo" :direction :output :if-exists :supersede))

(print 1 arch)

(print 2 arch)

(print 3 arch)

(close arch)
```

En el archivo que sigue se graban: la lista (A B (C D E) F G)
la cadena "hola que tal"
y el átomo A
y luego se leen los datos grabados en el archivo

```
(setq     arch    (open "archivo" :direction :output :if-exists :supersede))

(print '(a b (c d e) f g) arch)

(print "hola que tal" arch)

(print 'a arch)

(close arch)

(setq     arch    (open "archivo" :direction :input ))

> (read arch)
(A B (C D E) F G)
> (read arch)
"hola que tal"
> (read arch)
A
```

> (close arch)
T

Objetos funcionales

- Funciones definidas por el programador

(**DEFUN** <nombre> <lista parámetros> <cuerpo> **)**

Ej.: (DE FACT (N) (IF (EQ N 0) 1 (* N (FACT (- N 1)))))

Existe la posibilidad de usar parámetro opcionales con valores por defecto

```
( DEFUN  <nombre>  (  <parámetros>
                        &optional  ( <parám op> <valor>) ... ( <parám op> <valor>)
                    )
          <cuerpo>
)
```

```
Ej.:    (defun  lista_num   ( x   &optional   (z   nil )  )
                (cond  ( (null x)   z )
                       (  (numberp (car x))   (lista_num  (cdr x)  (cons (car x) z)  )  )
                       (   t     ( lista_num (cdr x)  z )  )
                )
        )
```

(lista_num '(a b 3 y 8 9 d)) → (3 8 9)

Aquí el segundo parámetro es opcional, o sea que la función lista_num se puede invocar con uno o con dos argumentos. Si se la invoca con un solo argumento, el valor del segundo parámetro es el valor establecido en la definición o sea que para este caso sería z=nil

- Funciones anónimas

(**LAMBDA** <lista parámetros> <cuerpo> **)**

Ej.: (LAMBDA (X) (+ X 1))

Aplicación al átomo 5: **(** **(**LAMBDA (X) (+ X 1) **)** 5 **)** → 6

- Formas funcionales

Las formas funcionales son funciones que tienen como parámetro otra/s funcion/es.

1) **(MAPCAR** <función> <lista> **)** Consiste en aplicarle la función a todos los elementos de la lista obteniendo una nueva lista.

Ej.: **(MAPCAR** atom ‘(a nil (1 2)) **)** → (t t nil)

Según el Lisp con que se trabaje esta función se la puede encontrar con el nombre de APPLY-TO-ALL

2) **(APPLY** <función> <lista> **)**

Si la función que se especifica es monádica, la lista debe contener un solo elemento, y el resultado es el valor que devuelve la función aplicada a ese elemento.
Si la función que se especifica es binaria, la lista debe contener solo dos elementos, y el resultado es el valor que devuelve la función aplicada a esos dos elementos.
Y así para funciones de más argumentos.

Ej.: **(APPLY** car ‘((a b)) **)** → a

Ej.: **(APPLY** cons ‘(a (1 2)) **)** → (a 1 2)

Ejemplo del uso de apply dentro de una función:

Vamos a escribir una función que aplicada a una lista dato y una lista que contiene nombres de funciones vaya modificando la lista dato con las funciones de la segunda lista.
(Sería como una especie de robot, al cual le damos la materia prima y una lista de ordenes para que aplique sobre la materia prima)

```
(defun robot ( objeto ordenes)
  (if  (null ordenes)  objeto
            (robot  (apply  (car ordenes) (list objeto))   (cdr ordenes) )
   )
)
```

Ej.: > (robot '(a (b c) d) '(cdr car cdr)) → (C)

Otra versión:

```
(defun robot ( objeto ordenes)
  (if  (null ordenes)  objeto
```

```
            (robot  (apply  ordenes (list objeto))   (read) )
   )
)
```

Ejemplos de invocaciones

```
> (robot '(a (b c) d)  (read)  )
cdr
car
cdr
nil
(C)

> (robot '(a (b c) d)  (read)  )
(lambda (x) (cons '1 x))
(lambda (x) (cons '2 x))
(lambda (x) (cons '3 x))
nil
(3 2 1 A (B C) D)
```

Uso del trace en Xlisp

(**trace** fn1 fn2 fn3 .....)

para desviar la salida por pantalla del trace a un archivo se pone:

(setq **trace-out** (open "C: test.txt" :direction :output :if-exists :supersede)

El trace es acumulativo. Cada vez que invoco trace con fns, éstas se agregan a las otras de los anteriores trace. Para sacar un función puesta en el trace se pone:

(untrace fn)

Si quiero vaciar la lista del trace pongo :

(trace NIL) o bien (untrace) en XLisp