Listas y Recursividad en LISP

LISP: Lists and recursion

Sebastián Molina Loaiza
Ingeniería de Sistemas y Computación
Universidad Tecnológica de Pereira, Pereira – Colombia
semolina@utp.edu.co

Resumen **- En el presente artículo se analizará la creación de funciones en LISP para resolver problemas que contengan listas y recursividad.**

Palabras clave **– LISP, recursividad, listas, funciones.**

Abstract- **This article has been written with the purpose of illustrating the implementation of different lists functions on LISP language.**

Key Word- **LISP, functions, lists, recursion.**

I. INTRODUCCIÓN

Se implementarán y analizarán la construcción de diferentes funciones en lenguaje LISP a fin de ilustrar el funcionamiento de las listas y la estructura del lenguaje.

II. TALLER 1

(* (/ (+ 3 3) (- 4 4)) (* 0 9))

```
CL-USER 1 > (* (/ (+ 3 3)(- 4 4))(* 0 9))

Error: Division-by-zero caused by / of (6 0).
  1 (continue) Return a value to use.
  2 Supply new arguments to use.
  3 (abort) Return to level 0.
  4 Return to top loop level 0.

Type :b for backtrace or :c <option number> to proceed»
.
Type :bug-form "<subject>" for a bug report template o»
r :? for other options.
```

La expresión anterior presenta un error debido a que la expresión(- 4 4) retorna 0 y la división por 0 no es permitida.

(+ (* 3 (- 5 3)) (/ 8 4))

```
CL-USER 2 : 1 > ( + ( * 3 ( - 5 3 ) ) ( / 8 4 ) )
8
```

La expresión anterior funciona correctamente, al final se evalúa la expresión (+ 6 2)

(* 3 2 2 (- 8 5))
Expresión correcta, se opera primero (- 8 5) y luego se realiza una cadena de multiplicaciones

((*) (* 2 (- 4 2)) (/ 8 2))

```
CL-USER 7 : 1 > ( ( * ) ( * 2 ( - 4 2 ) ) ( / 8 2 ) »
)

Error: Illegal argument in functor position: (*) in »
((*) (* 2 (- 4 2)) (/ 8 2)).
  1 (continue) Evaluate (*) and ignore the rest.
  2 (abort) Return to level 1.
  3 Return to debug level 1.
  4 Return to level 0.
  5 Return to top loop level 0.

Type :b for backtrace or :c <option number> to proce »
ed.
Type :bug-form "<subject>" for a bug report template »
 or :? for other options.
```

Expresión incorrecta debido a que la primera multiplicación no esta recibiendo argumentos

```
CL-USER 8 : 2 > ( sqrt ( abs (- 71 (expt ( + 2 4) 4 ))))
35.0

CL-USER 9 : 2 > '(' + '1 '(* 2 4) )
((QUOTE +) (QUOTE 1) (QUOTE (* 2 4)))

CL-USER 10 : 2 > (expt (mod 5 3) (abs (- 8 9)))
2

CL-USER 12 : 2 > (quote (nil `nil t `t))
(NIL (QUOTE NIL) T (QUOTE T))

CL-USER 13 : 2 > (log (expt 2 (- 15 5 4)))
4.158883

CL-USER 14 : 2 > (quote (quote (hello byebye)))
(QUOTE (HELLO BYEBYE))
```

III. TALLER 2

```
CL-USER 15 : 2 > (setq a 4 b 6 c 5 x (+ a b) y (- b c) z (max a c))
5

CL-USER 16 : 2 > eval x
10

CL-USER 17 : 2 > eval y
1

CL-USER 18 : 2 > eval a
4

CL-USER 19 : 2 > eval b
6
```

IV. TALLER 3

```
CL-USER 15 : 2 > (setq a 4 b 6 c 5 x (+ a b) y (- b c) z (max a c))
5

CL-USER 16 : 2 > (+ (* c z) b c)
36

CL-USER 21 : 2 > (abs (+ (* (- z x a) -100) B))
906

CL-USER 22 : 2 > (* (+ (* c x) (- a z c)) 2 y)
88

CL-USER 23 : 2 > (setq m (+ z a) n (- y c) p (* 2 z))
10

CL-USER 24 : 2 > (+ m z x y p b n)
37

CL-USER 25 : 2 > (- (- (* 3 z) (/ 100 c)) a c n p)
-20
```

V. TALLER 4

Expresiones:

```
CL-USER 26 : 2 > (cons (car `(axl wil rich))(cdr `(este anto alla)))
(AXL ANTO ALLA)

CL-USER 27 : 2 > (cons (cdar `((cons Go Up)))(third`(We Find(All Fine
))))
((GO UP) ALL FINE)

CL-USER 28 : 2 > (car (cdr (car (cdr `((a b) (c d) (e f))))))
D

CL-USER 29 : 2 > (car (car (cdr (cdr `((a b) (c d) (e f))))))
E

CL-USER 30 : 2 > (car (car (cdr `(cdr ((a b) (c d) (e f))))))
(A B)

CL-USER 31 : 2 > `(car (car (cdr (cdr ((a b) (c d) (e f))))))
(CAR (CAR (CDR (CDR #))))

CL-USER 32 : 2 > (cons (car nil)(cdr nil))
(NIL)
```

Formas:

```
CL-USER 33 : 2 > (cdr (car (cdr (car `((d (e f)) g (h i))))))
(F)

CL-USER 35 : 3 > (setq a `(+ 3 6))
(+ 3 6)

CL-USER 36 : 3 > (cdr a)
(3 6)

CL-USER 37 : 3 > (car (cdr a))
3

CL-USER 38 : 3 > (car (cdr (cdr a)))
6
```

La forma (setq a `(+ 3 6))) le sobraba un paréntesis que fue removido para poder ejecutarlo

VI. TALLER 5

```
CL-USER 41 : 6 > (setq a `(oso gato mapache ardilla))
(OSO GATO MAPACHE ARDILLA)

CL-USER 44 : 6 > (car (cdr (cdr a)))
MAPACHE

CL-USER 45 : 6 > (setq b `((oso gato) (mapache ardilla)))
((OSO GATO) (MAPACHE ARDILLA))

CL-USER 47 : 6 > (car (car (cdr b)))
MAPACHE

CL-USER 48 : 6 > (setq c `(((oso) (gato) (mapache) (ardilla))))
(((OSO) (GATO) (MAPACHE) (ARDILLA)))

CL-USER 49 : 6 > (car (cdr (cdr c)))
MAPACHE

CL-USER 50 : 6 > (setq d `(oso (gato) ((mapache)) (((ardilla)))))
(OSO (GATO) ((MAPACHE)) (((ARDILLA))))

CL-USER 51 : 6 > (car (cdr (cdr d)))
MAPACHE
```

VII. TALLER 6

```
CL-USER 19 > (setq a `(Managua Chinandega Rivas Leon Boaco))
(MANAGUA CHINANDEGA RIVAS LEON BOACO)

CL-USER 20 > (nth 3 a)
LEON

CL-USER 21 > (setq b `((Managua)(Chinandega Rivas Leon) Boaco))
((MANAGUA) (CHINANDEGA RIVAS LEON) BOACO)

CL-USER 22 > (nth 1 (nthcdr 1 (car (cdr b))))
LEON

CL-USER 23 > (setq c `(Managua (Chinandega (Rivas Leon Boaco))))
(MANAGUA (CHINANDEGA (RIVAS LEON BOACO)))

CL-USER 24 > (second (second (second c)))
LEON

CL-USER 25 > (setq d `(deportes (beisbol tenis) ((futbol) billar)))
(DEPORTES (BEISBOL TENIS) ((FUTBOL) BILLAR))

CL-USER 26 > (car (third d))
(FUTBOL)
```

VIII. TALLER 7

```
CL-USER 27 > (cons (second '(leo mana china))(cons(list(car '(1 2 3 4))(cdr »
 '(a b c d)))'2))
(MANA (1 (B C D)) . 2)

CL-USER 28 > (list (list '(ca ce) (last '(0 a 1 b ci)(third '(5 4 3 2 1)))( »
nth 3 '(pa pe pi po pu))))
(((CA CE) (1 B CI) PO))

CL-USER 29 > (list (append '(h o l a) '(m u n d o))(cdr '(sal pan arroz pol »
lo))(car '(buen mal))(caddr '(desayuno recreo almuerzo cena)))
((H O L A M U N D O) (PAN ARROZ POLLO) BUEN ALMUERZO)

CL-USER 30 > (list '(como estas?)(nth 0 '(bien mal rematado))(append (car ' »
((estas) estoy))(cdr '(entiendo entendiendo lisp))))
((COMO ESTAS?) BIEN (ESTAS ENTENDIENDO LISP))

CL-USER 31 > (length (cons (car '(verdad mentira falso))(list* 'es 'muy '(f »
acil))))
4

CL-USER 32 > (cdr(list(subseq '(z y x w v) 0 2)(cons '(a e i) '(o u))))
(((A E I) O U))
```

IX. TALLER 8

```
CL-USER 33 > (setq paises '((Nicaragua.Managua)(Italia.Roma)(Espana.Madri»
d)))
((NICARAGUA.MANAGUA) (ITALIA.ROMA) (ESPANA.MADRID))

CL-USER 34 > (list 'Nicaragua 'Italia 'Espana (SUBLIS paises '(Nicaragua »
Italia Espana))(nth 1 '(eran son seran))(car (nthcdr 1(cdr '(pueblos esta»
dos capitales barrios))))(cons 'de (cons 'estos (cons 'paises nil))))
(NICARAGUA ITALIA ESPANA (NICARAGUA ITALIA ESPANA) SON CAPITALES (DE ESTO»
S PAISES))
```

X. TALLER 9 10 y 11

```
CL-USER 1 > (setq Palabras '(grande bonito feliz humedo)
                  Sinonimos '(alto bello contento mojado)
                  Antonimos '(pequeno feo triste seco)
                  Ingles '(big beautiful happy humid))
(BIG BEAUTIFUL HAPPY HUMID)

CL-USER 2 > (setq Palabras-Sinonimos (pairlis Palabras Sinonimos))
((HUMEDO . MOJADO) (FELIZ . CONTENTO) (BONITO . BELLO) (GRANDE . ALTO))

CL-USER 3 > (setq Palabras-Antonimos (pairlis Palabras Antonimos))
((HUMEDO . SECO) (FELIZ . TRISTE) (BONITO . FEO) (GRANDE . PEQUENO))

CL-USER 4 > (setq Palabras-Ingles (pairlis Palabras Ingles))
((HUMEDO . HUMID) (FELIZ . HAPPY) (BONITO . BEAUTIFUL) (GRANDE . BIG))

CL-USER 5 > (last(cons(length(acons 'Lenguaje 'Ingles Palabras-Ingles))
                      (cons '(agregando)(cons '(Palabras)(List '(ala)
                          '(Lista-Asoc))))))
((LISTA-ASOC))

CL-USER 6 > (cdr(assoc 'Grande Palabras-Ingles))
BIG

CL-USER 7 > (cdr(assoc 'Feliz Palabras-Sinonimos))
CONTENTO

CL-USER 8 > (first(assoc 'Humedo(acons 'pal 'anto Palabras-Antonimos)))
HUMEDO
```

XI. TALLER FINAL PUNTO I

En los ejercicios a analizar se evidencia que todas las funciones Lambda realizan operaciones aritméticas básicas y se les definen sus parámetros para ser ejecutadas.

La conversión a Lambda se lleva a cabo reemplazando la palabra DEFUN por LAMBDA, luego se omite el nombre ya que no es necesario, dentro del primer campo se le pasaran los argumentos de la función, en el segundo se define el cuerpo de la función y por ultimo se le indican los valores que se le van a entregar como argumentos a la función.

XII. TALLER FINAL PUNTO II

La función FUNCALL no posibilita enviar una función como un parámetro a otra función, en el ejercicio lo que se hace es pasarle a la función PRINT otra función como argumento que en este caso es la función LAMBDA que realiza una operación aritmética simple.

XIII. TALLER FINAL PUNTO III

La función APPLY nos permite pasar una lista de argumentos a una función cualquiera, puede ser LAMBDA, propia del lenguaje como la suma, o definida por nosotros.

XIV. CONCLUSIONES

Algunos ejercicios contenían un error de sintaxis referente a la falta de paréntesis, en esos se corrigió de manera que le interprete pudiera ejecutar bien la sentencia. Se realizaron todos los talleres propuestos pudiendo obtener una visión mas de fondo sobre LISP, sus funcionalidades, funciones, palabras reservadas y demás.

VII. REFERENCIAS

[1] http://www.redesep.com/materias/blanda/index.php