

# Desarrollo básico en Lisp

## Basic development in Lisp

Javier Mauricio Gil Cardona

IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia

mauriciogi@utp.edu.co

**Resumen—** En este documento se mostrará el correcto funcionamiento de la herramienta integrada de desarrollo multiplataforma para el lenguaje lisp más concretamente el software lispworks para esto lo podremos a prueba con algunas funciones recursivas ya establecidas y previamente revisadas en clase.

**Palabras clave—** software, lispworks, lisp, herramienta integrada, desarrollo, multiplataforma, lenguaje de programación, inteligencia artificial, funciones, recursividad.

**Abstract—** In this document you will find the correct functioning of the integrated multi platform development tool for the lisp language, more specifically the lisp software works for this we will be able to test it with some functions already established and previously reviewed in class.

**Keyword—** software, lisp works, lisp, integrated tool, development, multiplatform, programming language, artificial intelligence, functions, recursion.

## I. INTRODUCCIÓN

En este paper vamos a explicar detalladamente algunas funciones realizadas en el lenguaje Lisp, las cuales fueron previamente revisadas y explicadas por parte del docente en la clase.

Una definición de Lisp es que es un lenguaje de computadora de tipo multiparadigma con larga historia y una inconfundible y útil sintaxis basada en la notación polaca. Fue desarrollado originalmente por John McCarthy y algunos colaboradores en el instituto tecnológico de Massachusetts, cabe resaltar que Lisp es de los más antiguos y de alto nivel que aún tienen un uso extendido, solo por debajo de COBOL y FORTRAN estos tres han ido evolucionando y cambiando muchas cosas desde sus inicios.

Lisp fue creado originalmente como una notación matemática práctica para los programas de computadora, basada en el cálculo lambda de Alonzo Church. Se convirtió rápidamente en el lenguaje de programación favorito en la investigación de

la inteligencia artificial. Como lenguaje de programación precursor Lisp fue pionero en muchas ideas en las ciencias de la computación, incluyendo las estructuras de datos de árbol, el manejo de almacenamiento automático, tipos dinámicos, y el compilador autocontenido.

El acrónimo Lisp significa List Processor “Procesamiento de listas” y ya veremos que como su nombre lo indica todo el lenguaje está basado en listas dentro de listas.

## II. CONTENIDO

Para implementar estas funciones y muchos otros programas con este lenguaje usaremos un software informático llamado LispWorks

Este es una implementación patentada y un entorno de desarrollo integrado para el lenguaje de programación Common Lisp. LispWorks fue desarrollado por la compañía de software del Reino Unido Harlequin Ltd, y publicado por primera vez en 1989. Esta compañía finalmente se separó de su división Lisp como Xanalys Ltd, que se hizo cargo de la gestión y los derechos de LispWork. En enero de 2005, el equipo de Xandalys Lisp formó Lisp Works Ltd. para comercializar, desarrollar y respaldar el software.

El software tiene una interfaz minimalista y sencilla aunque no debemos confundir estas palabras y vincularlas a su funcionalidad, ya que contamos con una herramienta robusta en la que podremos trabajar cómodamente nuestros códigos.

Cabe resaltar que este software es Open Source y se encuentra disponible para múltiples sistemas operativos

Cuando abrimos el software nos sale una pequeña ventana la cual se muestra en la **FIGURA 1.0** donde se describen las edición que estamos usando y las que podemos usar si hacemos un Upgrade a las versiones pro del software, claro que estas ya tiene un costo.

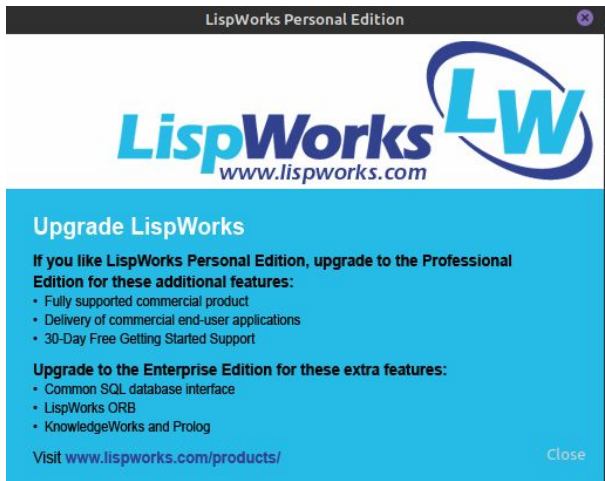


FIGURA 1.0 Ventana Descriptiva de LispWorks

Simultáneo a esta ventana **FIGURA 1.1**, también nos saldrá las ventanas para ejecutar y escribir nuestros programas y hacer las configuraciones pertinentes para una mejor experiencia.

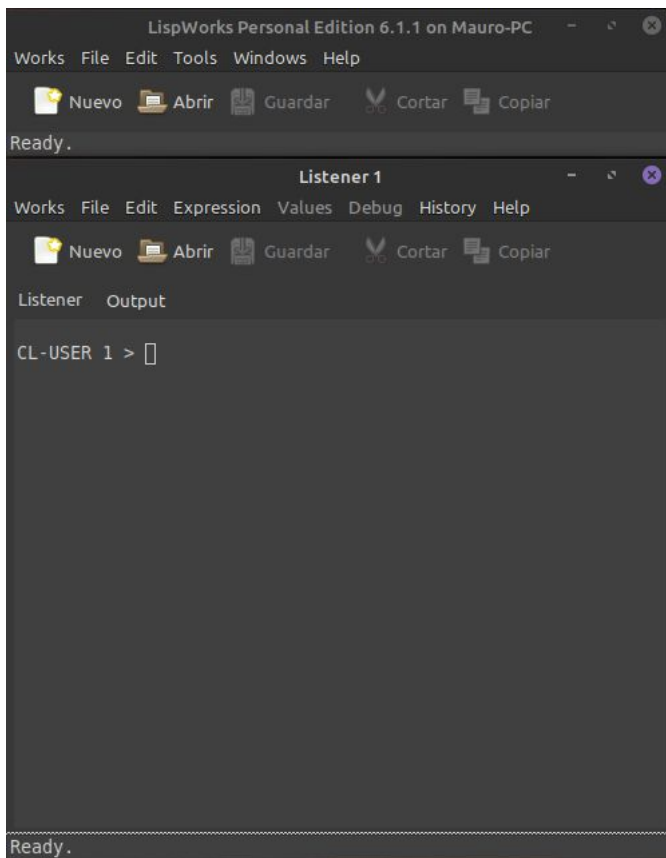


FIGURA 1.1 Entorno de LispWorks 6.1.1

### **Función Miembro-de:**

Esta función que se muestra en la **FIGURA 2.0** lo que realiza es buscar si un elemento se encuentra dentro de una lista similar a lo que realiza member la cual es una sentencia propia

de Lisp, a diferencia de member esta función muestra el elemento en la lista y el resto de elemento de la lista que siendo el elemento buscado la nueva cabeza de dicha lista creando así una segunda lista.

```
(defun miembro-de-lista (elemento lista)
  (cond ((null lista) nil)
        ((equal elemento (car lista)) lista)
        (t (miembro-de-lista elemento (cdr lista)))))
```

FIGURA 2.0 Función miembro-de

Si analizamos detalladamente la función línea por línea podemos ver que las funciones se crean con la palabra reservada defun seguido del nombre que le queramos poner a la variable en este caso miembro-de-lista después proseguimos a colocar los argumentos que recibirá la función los cuales para esta serán uno llamado elemento que es el que se va a buscar dentro de la lista, y como segundo argumento el de la lista a revisar.

En la segunda línea se puede observar que tenemos una condición que nos indica que si la lista es nula; es decir que no tenga elementos entonces retorna nil que equivale a un falso.

En la tercera línea podemos ver otra sentencia que nos sirve también para finalizar la recursividad, esta línea lo que hace es que si el elemento es igual a la cabeza de la lista entonces retorna la lista completa, debido a que la lista a generarse sería la misma.

La cuarta línea nos indica que si la sentencia es verdadera ingrese a hacer una recursividad de que el miembro de la lista encontrado va a ser la nueva cabeza de la lista y así sucesivamente hasta encontrar el elemento seleccionado a buscar.

Para usar esta función lo haremos como se muestra en la **FIGURA 2.1**.

```
(print (miembro-de-lista 4 '1 2 3 4 5 6))
```

FIGURA 2.1 llamado de miembro-de.

Con el print vamos a mostrar en pantalla el resultado de este llamado de función el cual lo que realizará es buscar si el elemento 4 se encuentra en dentro de la lista posteriormente escrita y a su vez ponerlo de cabeza para el resto de la lista.

El resultado de esta función se muestra en la **FIGURA 2.2**.

```
;;; Cross referencing is on
; MIEMBRO-DE-LISTA
; (TOP-LEVEL-FORM 2)

(4 5 6)

---- Done ----
```

**FIGURA 2.2** Resultado de función miembro-de.

#### **Función Longitud:**

Esta función **FIGURA 3.0** a simple vista lo que hace esta función es contar cuántos elementos tenemos dentro de una lista.

```
(defun longitud (lista)
  (cond ((null lista) 0)
        (t (+ (longitud (cdr lista)) 1))))
```

**FIGURA 3.0** Función longitud.

Más detalladamente lo que hace en la primera línea es definir la función en este caso se llama longitud y recibe un argumento lista.

En la segunda línea tenemos un condicional el cual nos dice que si la lista es nula entonces es 0.

En la tercera nos muestra que si la sentencia es verdadera entonces que nos sume la longitud de la cola más uno y esto recursivamente hasta que la función se finalice con la y esto pasará cuando la lista sea nula.

Para usar esta función lo haremos como se muestra en la **FIGURA 3.1**.

```
(print (longitud 'a b c d e f g h))
```

**FIGURA 3.1** llamado de la función longitud.

Lo que se realiza en esta línea de código es mostrar en pantalla el resultado del llamado a función el cual nos dirá cuántos elementos tiene la lista también escrita dentro de argumento solicitante.

El resultado de la función se muestra en la **FIGURA 3.2**.

```
;;; Compilation speed = 1, Debug = 2, Fixnum safety = 3
;;; Source level debugging is on
;;; Source file recording is on
;;; Cross referencing is on
; LONGITUD
; (TOP-LEVEL-FORM 2)

8

---- Press Space to continue ----
```

**FIGURA 3.2** Resultado de la función longitud.

#### **Función termino-n**

Esta función la usaremos para extraer un elemento cualquiera dentro de una lista sin importar la posición en la que se encuentre, la funcionalidad de esta es similar a lo que hace la palabra reservada nth que nos muestra el elemento de una lista según su posición dentro de ella. **FIGURA 4.0**.

```
(defun termino-n (n lista)
  (cond ((zerop n) (car lista))
        (t (termino-n (- n 1) (cdr lista)))))
```

**FIGURA 4.0** Función termino-n.

Observando más a fondo esta función podemos ver que como las anteriores se define una función llamada termino-n la cual recibe dos argumentos uno llamado n el cual corresponde a un número que será la posición del elemento que quieres mostrar de la lista y el otro argumento es la lista a analizar.

En la segunda línea tenemos una condición que nos dice que si n = 0 entonces el resultado es la cabeza de la lista.

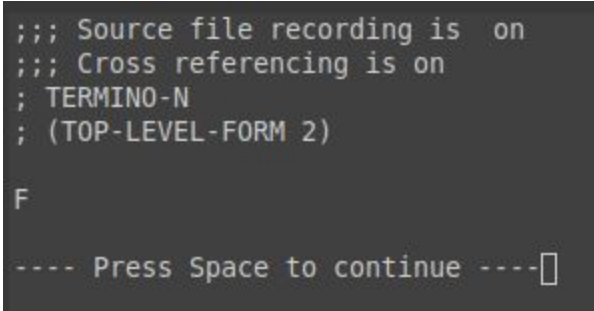
Por último tenemos que si la sentencia es verdadera; es decir que si n != 0 entonces recursivamente llamamos a 'n' y restamos n - 1 posterior a esto mostraremos la cola de la lista con esto lograremos una recursividad hasta que n sea igual a 0.

Para llamar esta función y obtener resultados la llamamos de la forma que se muestra en la **FIGURA 4.1**.

```
(print (termino-n 5 'a b c d e f g h i j k l))
```

**FIGURA 4.1** llamado de la función termino-n.

Cómo resultado a todo esto tenemos que elemento 5 de la lista es la F, cabe aclarar que Lisp usa index-0, es decir que sus lista se cuentan desde la posición 0 siendo esta la ubicación del elemento número uno de lista. Como se muestra en la **FIGURA 4.2**.



```
;;; Source file recording is on
;;; Cross referencing is on
; TERMINO-N
; (TOP-LEVEL-FORM 2)

F

---- Press Space to continue ----
```

**FIGURA 4.2** Resultado de la función termino-n.

### III. CONCLUSIÓN

Se aprecia que el software lispWorks es bastante antiguo pero aun así sigue siendo bastante robusto y funcional. Por otra parte el software puede llegar a ser un poco tedioso en su manejo debido a que no es tan amigable a la vista pero igualmente cumple con lo que tiene que cumplir en cuanto a su interfaz y funcionalidad.

Se comprobó que las funciones que se muestran dentro del contenido del capítulo están totalmente funcionales, correctas y listas para su aplicación en proyectos más ambiciosos.

### IV. REFERENCIAS

1. [http://www.redesep.com/materias/inteligencia/5\\_2\\_\\_estructuras\\_rekursivas.php](http://www.redesep.com/materias/inteligencia/5_2__estructuras_rekursivas.php)
2. <http://www.lispworks.com/>
3. <https://en.wikipedia.org/wiki/LispWorks>
4. <https://es.wikipedia.org/wiki/Lisp>