Lisp

Daniel Delgado, *Estudiante, ITCR,* Wilbert Gonzales, *Estudiante, ITCR,*
Anthony Leandro, *Estudiante, ITCR,* and Bryan Mena, *Estudiante, ITCR*

1. Datos Historicos

En 1960, John McCarthy publicó un notable artículo: *Funciones recursivas de expresiones simbólicas y su cómputo a máquina, Parte I* (la Parte II nunca fue publicada), en el que hizo para programar algo parecido a lo que Euclides hizo para la geometría. Él demostró cómo, dado un grupo de operadores simples y una notación para las funciones, se puede construir un lenguaje de programación entero. Llamó a este lenguaje Lisp, para "List Processing", porque una de sus ideas clave era usar una estructura de datos simple llamada *lista* para código y datos.

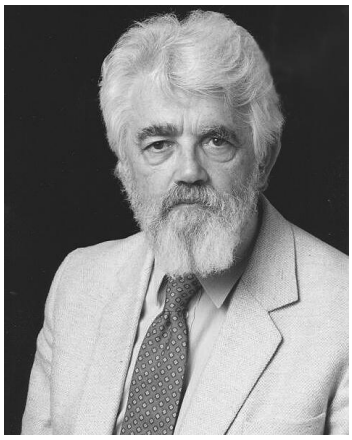

Figura 1. John McCarthy (1927-2011).

Desde su inicio, Lisp estaba estrechamente relacionado con la comunidad de investigación de la inteligencia artificial, especialmente en sistemas PDP-10 (ver Figura 2). Lisp es un lenguaje multiparadigma que utiliza *notación polaca*.

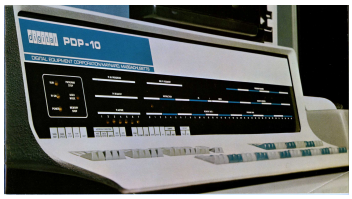

Figura 2. Computador PDP-10.

La notación original de McCarthy usaba *Expresiones-M* en corchetes que serían traducidas a *Expresiones-S*. Como un ejemplo, la Expresión-M *car[cons[A,B]]* es equivalente a la Expresión-S *(car (cons A B))*. Una vez que Lisp fue implementado, los programadores rápidamente eligieron usar Expresiones-S, y las Expresiones-M fueron abandonadas. Una Expresión-S es una notación en forma de texto, para representar una estructura de datos de árbol, basada en listas anidadas, en donde cada sublista es un subárbol (ver Figura 3).

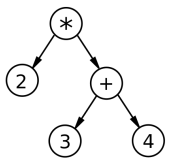

Figura 3. Árbol de Expresión-S

2. Tipos de Datos

2.1. Integer

Números sin parte fraccionaria. El rango de valores de un entero depende de la máquina. El intervalo mínimo es de $-536{,}870{,}912$ a $536{,}870{,}911$ (30 bits, es decir, -2^{29} a $2^{29}-1$), pero muchas máquinas proporcionan un rango más amplio. La sintaxis de lectura para números enteros es una secuencia de dígitos (base diez) con un signo opcional al principio y un punto *(.)* opcional al final. La representación impresa producida por el intérprete de Lisp nunca tiene un '+' o un final '.'.

Cuadro 1. Código de tipo de dato Integer en Lisp

```
−1              ; El entero −1.
1               ; El entero 1.
1.              ; Igual, el entero 1.
+1              ; Igual, el entero 1.
```

Como una excepción especial, si una secuencia de dígitos especifica un entero demasiado grande o demasiado pequeño para ser un objeto entero válido, el intérprete de Lisp lo lee como un número de punto flotante.

2.2. Floating-Point

Números con partes fraccionarias y con una gran rango. Los números de punto flotante son el equivalente computacional de la notación científica; se puede pensar en un número de punto flotante como una fracción junto con una potencia de diez. El número preciso de figuras significativas y el rango de posibles exponentes es específico de la máquina. La representación impresa para números de punto flotante requiere un punto decimal (con al menos un dígito siguiente), un exponente o ambos. Por ejemplo:

Cuadro 2. Maneras de representar el valor 1500 en punto flotante

```
1500.0          ;1500.0
+15e2           ;15 positivo por 10 a la 2
15.0e+2         ;15 por 10 a la 2 positivo
+1500000e−3     ;1500000 por 10 a la −3
.15e4           ;0.15 por 10 a la 4
```

2.3. Character

La representación de letras, números y caracteres de control. Un caracter de Lisp no es más que un entero. En otras palabras, los caracteres están representados por sus códigos de caracteres. Por ejemplo, el caracter A se representa como el entero 65. Los caracteres especiales iniciales en la tabla ASCII se pueden invocar utilizando un backslash y su letra asociada. Para utilizar un backslash, se debe anteponer otro backslash. Por ejemplo:

Cuadro 3. Representación de caracteres

```
 ?Q = 81
 ?q = 113
 ?\a = 7          ; control-g
 ?\b = 8          ; backspace, <BS>
 ?\t = 9           ; tab, <TAB>
 ?\n = 10         ; newline
 ?\v = 11     ; vertical tab
 ?\f = 12          ; formfeed character
 ?\r = 13          ; carriage return, <RET>
 ?\e = 27         ; escape character, <ESC>
 ?\s = 32         ; space character, <SPC>
 ?\\ = 92          ; backslash character, \
 ?\d = 127   ; delete character, <DEL>
?\(       ; parentesis
```

2.4. Symbol

Objeto multiuso que hace referencia a una función, variable o lista de propiedades, y tiene una identidad única.

Cuadro 4. Ejemplos de símbolos

```
foo        ; Un simbolo llamado foo.
FOO ;Un simbolo llamado FOO, distinto de foo.
1+         ; Un simbolo llamado  1+
           ; (no +1, el cual es un entero).
\+1              ; Un simbolo llamado  +1
           ; (no es un nombre muy practico).
\(*\ 1\ 2\)    ; Un simbolo llamado (* 1 2)
                 ; (poco practico).
```

2.5. Sequence

Las listas y arreglos se clasifican como secuencias. Las listas son las secuencias más comúnmente usadas. Una lista puede contener elementos de cualquier tipo, y su longitud se puede cambiar fácilmente agregando o quitando elementos. La lista vacía () siempre significa el mismo objeto, *nil*.

2.6. Cons cell

Es un objeto que consta de dos espacios, llamados: el espacio del *car* y el espacio del *cdr*. Cada espacio puede contener cualquier objeto de Lisp. También, se dice que el *car* de este objeto es cualquier objeto que su espacio mantiene actualmente, y también para el *cdr*. Un objeto que no es un Cons cell es denominado como *atom*.

2.7. Array

Un arreglo está compuesto por un número arbitrario de espacios para referirse a otros objetos de Lisp, ordenados en un bloque de memoria contínua. El acceso de cualquier elemento de un arreglo toma aproximadamente la misma cantidad de tiempo. En cambio en una lista el acceso a un elemento requiere un tiempo proporcional a su posición en la lista. Entre más al final de la lista esté, más tiempo toma acceder a ese elemento.

2.8. String

Es un arreglo eficiente de caracteres. Se escribe con doble comilla ("), seguidamente de un número arbitrario de caracteres y de nuevo, doble comilla ("). Para incluir una comilla entre un string se debe poner un backslash antes de la doble comilla.

2.9. Vector

Son arreglos unidimensionales de elementos de cualquier tipo. Su representación consiste en una llave cuadrada para abrir ([ ), los elementos y una llave cuadrada para cerrar ( ]).

Cuadro 5. Ejemplo de vector

```
[1 "two" (three)] ; Vector de 3 elementos
```

2.10. Char-Table

Arreglos disperso unidimensional de elementos de cualquier tipo, indexado por caracteres. Los Char-Table tienen ciertas características adicionales para hacerlas más útiles para muchos trabajos que implican asignar información a los códigos de caracteres; por ejemplo, una tabla de caracteres puede tener un padre de donde heredar, un valor predeterminado y un número pequeño de espacios adicionales para uso con fines especiales. Un Char-Table también puede especificar un solo valor para un juego de caracteres completo. La representación impresa de un Char-Table es como un vector excepto que comienza con #^ seguido de la longitud.

2.11. Bool-Vector

Arreglo unidimensional cuyos elementos deben ser *t* o *nil*. La representación impresa de un Bool-Vector es como un string, excepto que comienza con #&. La constante de string que sigue especifica realmente el contenido del Bool-Vector como un mapa de bits. Cada caracter del string contiene 8 bits, que especifican los siguientes 8 elementos del Bool-Vector (1 representa *t*, y 0 para *nil*).

2.12. Hash Table

Un Hash Table es un tipo muy rápido de tabla de búsqueda, algo así como una lista que asigna claves a los valores correspondientes, pero mucho más rápido.

2.13. Function

Es una pieza de código ejecutable e invocable desde cualquier lugar, al igual que las funciones en otros lenguajes de programación. En Lisp, a diferencia de la mayoría de los lenguajes, las funciones también son objetos de Lisp. Una función no compilada en Lisp es una expresión lambda y tampoco tiene un nombre intrínseco. Una expresión lambda se puede llamar como una función aunque no tenga nombre. Una función con nombre en Lisp es sólo un símbolo con una función válida en su celda de función.

2.14. Macro

Un método para expandir una expresión en otra expresión, más fundamental pero menos bonita.

2.15. Primitive Function

Una función primitiva es una función llamada desde Lisp pero escrita en el lenguaje de programación C.

2.16. Byte-Code

Una función escrita en Lisp, luego compilada.

2.17. Autoload

Tipo utilizado para cargar automáticamente las funciones raramente utilizadas. Un objeto Autoload es una lista cuyo primer elemento es el símbolo *autoload*.

2.18. Finalizer

Un objeto Finalizer ayuda a limpiar el código de Lisp después de que los objetos ya no son necesarios. Un Finalizer tiene un objeto de función Lisp.

3. Estructuras de Control y Expresiones

3.1. Operadores aritméticos

Si A tiene el valor 10, y B el valor 20:

Operador	Descripción	Ejemplo
+	Suma 2 operandos	(+ A B) devuelve 30
-	Resta el segundo operando al primero	(- A B) devuelve -10
*	Multiplica ambos operandos	(* A B) devuelve 200
/	Divide el numerador con el denominador	(/ B A) devuelve 2
mod, rem	Módulo y residuo	(mod B A) devuelve 0
incf	Incrementa el valor entero del primer argumento con el segundo argumento	(incf A 3) devuelve 13
decf	Decrementa el valor entero del primer argumento con el segundo argumento	(decf A 4) devuelve 6

3.2. Operadores de comparación

Si A tiene el valor 10, y B el valor 20:

Operador	Descripción	Ejemplo
=	Revisa si 2 operandos son iguales o no	(= A B) is not true
/=	Revisa si 2 operandos son distintos o no	(/= A B) is true
>	Revisa si los valores de 2 operandos son decrecientes	(> A B) is not true
<	Revisa si los valores de 2 operandos son crecientes	(< B A) is true
>=	Revisa si el valor del operando izquierdo es mayor o igual que el operando derecho	(>= A B) is not true
<=	Revisa si el valor del operando izquierdo es menor o igual que el operando derecho	(<= A B) is true
max	Devuelve el mayor valor entre 2 operandos	(max A B) devuelve 20
min	Devuelve el menor valor entre 2 operandos	(min A B) devuelve 10

3.3. Operadores lógicos en valores booleandos

Si A tiene el valor *nil*, y B el valor 5:

Operador	Descripción	Ejemplo
and	Toma cualquier cantidad de argumentos y revisa de izquierda a derecha, si todos los argumentos son no **nil**, devuelve el valor del último argumento, sino **nil**	(and A B) devuelve **nil**
or	Toma cualquier cantidad de argumentos y revisa de izquierda a derecha, si algún argumento no es **nil** devuelve el argumento en otro caso, **nil**.	(or A B) devuelve **5**
not	Toma un argumento y retorna **t** si el argumento es evaluado como **nil**	(not A) devuelve **t**

3.4. Llamadas a funciones

En Lisp una función, que normalmente se denota como funcion(x), se escribe mediante la forma **(funcion x)**.

3.5. Variables Globales

En Lisp, cada variable es representada por un símbolo. Las variables globales son por lo general declaradas usando el constructo **defvar**. Como no hay declaración de tipo para variables en LISP, también se puede especificar directamente un valor para un símbolo con el constructo **setq**.

Cuadro 6. Declaración de variables globales

```
(defvar x 234)   ; x = 234
(setq y 10)      ; y = 10
```

3.6. Variables Locales

Las variables locales también pueden ser declaradas mediante el constructo **setq**. También hay dos constructos más: **let** y **prog**:

Cuadro 7. Declaración de variables locales

```
(let ((x 'a) (y 'b)(z 'c))
;x = A y = B z = C

(prog ((x '(a b c))(y '(1 2 3))(z '(p q 10)))
;x = (A B C) y = (1 2 3) z = (P Q 10)
```

3.7. Macros

Una macro es definida por otra macro llamada **defmacro**. La sintaxis para definir una macro es la siguiente:

Cuadro 8. Estructura de una macro

```
(defmacro nombre-macro (lista-parametros))
"Texto_opcional_para_documentacion"
cuerpo-de-la-macro
```

Donde el cuerpo de una macro consiste en expresiones que definen las tareas de la macro. Por ejemplo:

Cuadro 9. Ejemplo de una macro

```
(defmacro setTo10(num)
(setq num 10)(print num))
; definicion de la macro: toma una variable
;la convierte a 10 y la imprime

(setq x 25)      ;se define x=25
(print x)        ;se imprime x
(setTo10 x)      ;se llama la macro
                 ;con parametro x
                 ;salida:
                 25
                 10
```

3.8. Definición de funciones

Las funciones se definen por medio del constructo **defun** (ver Cuadro 10).

3.9. Definición de constantes

Las constantes se definen por medio del constructo **defconstant**. Ejemplo:

Cuadro 10. Definir funciones y constantes

```
(defconstant PI 3.141592)
(defun area-circle(rad)
   (terpri);funcion terminate print-line
   (format t "Radius:~5f" rad)
   (format t "Area:~10f" (* PI rad rad)))
(area-circle 10)  ;llamada a la funcion
```

3.10. Decisiones

Los siguientes constructos se utilizan para definir decisiones:

- **cond**
- **if**
- **when**
- **case**

Cuadro 11. Ejemplos de decisiones

```
(setq a 10)
(cond ((> a 20)
   (format t "a␣is␣greater␣than␣20"))
   (t (format t "value␣of␣a␣is␣~d␣" a)))

(setq a 10)
(if (> a 20)
   (format t "a␣is␣less␣than␣20"))
   (format t "value␣of␣a␣is␣~d␣" a)

(setq a 100)
(when (> a 20)
   (format t "a␣is␣greater␣than␣20"))
   (format t "value␣of␣a␣is␣~d␣" a)

(setq day 4)
(case day
(1 (format t "Monday"))
(2 (format t "Tuesday"))
(3 (format t "Wednesday"))
(4 (format t "Thursday"))
(5 (format t "Friday"))
(6 (format t "Saturday"))
(7 (format t "Sunday")))
```

3.11. Ciclos

Los siguientes constructos se utilizan para definir ciclos:

- **loop**
- **loop for**
- **do**
- **dotimes**
- **dolist**

Cuadro 12. Ejemplos de ciclos

```
(setq a 10)
(loop
   (setq a (+ a 1))
   (write a)
   (terpri)
   (when (> a 17) (return a)))

(loop for a from 10 to 20
   do (print a))

(dotimes (n 11)
   (print n) (print (* n n)))

(dolist (n '(1 2 3 4 5 6 7 8 9))
   (print n))
```

4. CARACTERÍSTICAS

Lisp incorpora 9 nuevas ideas en su momento:

4.1. Condicionales

Un condicional es un constructo if-then-else. Ahora se toma esto como un hecho. Fueron inventados por McCarthy en el curso del desarrollo de Lisp. (Fortran en ese momento sólo tenía un goto condicional, estrechamente basado en la instrucción de la rama en el hardware subyacente.) McCarthy, que estaba en el comité de Algol, consiguió condicionales en Algol, de donde se extendieron a la mayoría de los otros idiomas.

4.2. Un tipo Función

En Lisp, las funciones son objetos de primera clase: son un tipo de dato como lo son los enteros, strings, etc, y tienen una representación literal, se pueden almacenar en variables, se pueden pasar como argumentos, etc.

4.3. Recursión

La recursión existió como un concepto matemático antes de Lisp, pero Lisp fue el primer lenguaje de programación que lo soportó. (Es indiscutiblemente implícito en hacer que las funciones sean objetos de primera clase.)

4.4. Un nuevo concepto de variables

En Lisp, todas las variables son efectivamente punteros. Los valores son lo que tienen tipos, no variables, y asignar o enlazar variables significa copiar punteros, no a lo que apuntan.

4.5. Garbage-collection

Aunque había que llamar el garbage collector por medio del comando **(garbage-collect)**, Lisp fue el primero en implementar este concepto.

4.6. Programas compuestos de expresiones

Los programas Lisp son árboles de expresiones, cada uno de los cuales devuelve un valor. (En algunas expresiones de Lisp puede devolver varios valores.). Esto es en contraste con Fortran y la mayoría de los lenguajes que vinieron después, que distinguen entre expresiones y declaraciones.

Era natural tener esta distinción en Fortran porque (no es de extrañar en un idioma donde el formato de entrada era tarjetas perforadas) el lenguaje estaba orientado a líneas. No se podían anidar declaraciones. Y así, mientras se necesitaban expresiones para que las matemáticas funcionaran, no tenía sentido hacer que algo más devolviera un valor, porque no podía haber nada esperando dicho valor.

Esta limitación se fue con la llegada de lenguajes estructurados en bloques, pero para entonces ya era demasiado tarde. La distinción entre expresiones y declaraciones estaba arraigada. Se extendió de Fortran a Algol y de allí a sus dos descendientes.

Cuando un lenguaje está hecho completamente de expresiones, se puede componer expresiones de cualquier forma.

Cuadro 13. Dos maneras de escribir la misma expresión en Lisp

```
(if foo (= x 1) (= x 2))
;or
(= x (if foo 1 2))
```

4.7. Un tipo símbolo

Los símbolos difieren de los string en que se puede probar la igualdad mediante la comparación de un puntero.

4.8. Una notación para código

Usando árboles de símbolos.

4.9. El lenguaje completo está siempre disponible

No hay una distinción real entre tiempo de lectura, tiempo de compilación y tiempo de ejecución. Se puede compilar o ejecutar código mientras se lee, leer o ejecutar código mientras se compila, y leer o compilar código en tiempo de ejecución. Ejecutar código en tiempo de lectura permite reprogramar la sintaxis de Lisp; ejecutar código en tiempo de compilacipn es la base de macros; compilar en tiempo de ejecución es la base del uso de Lisp como un lenguaje de extensión en programas como Emacs (variación de Lisp); y la lectura en tiempo de ejecución permite a los programas comunicarse utilizando Expresiones-S, una idea reinventada conocida como XML.

5. VENTAJAS DE LISP

- Código Homoicónico. Esto permite código de automodificación estructurado.
- Sintaxis de macros. Permite la reescritura de código repetitivo.
- Pragmatismo. Lisp está diseñado para hacer que las cosas sean hechas por profesionales que trabajan. La mayoría de los lenguajes funcionales no lo son, por regla general.
- Flexibilidad. Puede hacer muchas cosas diferentes, todas a velocidades razonables.
- Dureza. El mundo real es desordenado. La codificación pragmática termina por tener que usar o inventar construcciones desordenadas. Lisp tiene la suficiente dureza como para resolver problemas sin importar el desorden.

REFERENCIAS

[1] P. Graham, (2002). The Roots of Lisp. Recuperado de http://ep.yimg.com/
[2] Anónimo. Lisp Data Types. Recuperado de https://www.gnu.org/
[3] P. Graham, (2001). What made Lisp Different. Recuperado de http://www.paulgraham.com/
[4] Anónimo. Lisp. Recuperado de https://es.wikipedia.org/
[5] Anónimo. Expresión S. Recuperado de https://es.wikipedia.org/
[6] Anónimo. Lisp - Quick Guide. Recuperado de https://www.tutorialspoint.com/
[7] Anónimo. Garbage Collection. Recuperado de https://www.gnu.org/