

Programación Funcional en LISP

Lógica para Ciencias de la Computación

Primer Cuatrimestre de 2009

– Material Adicional –

Introducción

- A lo largo de la carrera estudiaremos diversos paradigmas de programación:
 - ➡ Programación Orientada a Objetos.
 - ➡ Programación Lógica.
 - ➡ Programación Funcional.
- En esta clase, abordaremos uno de los lenguajes que pertenecen al paradigma funcional, el lenguaje **LISP**.

Origen

- **LISP** se originó en la década del '50, de la mano de John McCarthy.
- Se trata de un lenguaje maduro, aplicable a diversos escenarios, tales como:
 - ➡ Robótica.
 - ➡ Inteligencia Artificial.
 - ➡ Procesamiento de lenguaje natural.
 - ➡ Demostración automática de teoremas.

Principales Características

- La característica principal de **LISP** es que todo programa consiste de una función.
- Otras peculiaridades:
 - ➡ No posee asignaciones.
 - ➡ Su principal estructura de control es la recursión.
 - ➡ Los programas y los datos son equivalentes.
 - ➡ Su principal estructura de datos es la lista.
 - ➡ La memoria es asignada por demanda.

LISP vs. LISP Puro

- **LISP** incorpora ciertos vicios tomados del paradigma imperativo, que permiten expresar aspectos ajenos al paradigma funcional.
- El subconjunto de **LISP** netamente funcional se lo suele denominar **LISP puro**.
- En lo que resta, nos concentraremos exclusivamente en **LISP puro**.

Expresiones en LISP

- Todo programa es una **s-expresión**.
- Las s-expresiones pueden ser **átomos** o **listas**.
- Un átomo puede ser un **átomo numérico** o un **átomo simbólico**.
- Una lista es una secuencia de s-expresiones delimitadas por paréntesis, con sus elementos separados por espacios en blanco.

Representación de Funciones

• La aplicación de una función **LISP** se nota en forma prefija, mediante una **lista**:

• La función **f** aplicada a los argumentos $a_1, a_2, a_3, \dots, a_n$, se denota $(f\ a_1\ a_2\ a_3\ \dots\ a_n)$, en vez del usual $f(a_1, a_2, a_3, \dots, a_n)$.

• Por ejemplo, la **suma** de 2 con el **producto** de 5 y -3 se denota como:

átomos simbólicos

$(+ 2 (* 5 -3))$

átomos
numéricos

Evaluación de s-expresiones

- Como vimos antes, es posible expresar aplicaciones anidadas de funciones.
- Básicamente, el intérprete de **LISP** evalúa una **s-expresión** S , representando posiblemente aplicaciones anidadas de funciones, de la siguiente forma:
 - ➡ Si S es un átomo numérico a , entonces el resultado es simplemente dicho átomo numérico a .
 - ➡ Si S es de la forma $(f\ a_1\ a_2\ a_3\ \dots\ a_n)$, entonces el intérprete evalúa recursivamente cada argumento a_i , obteniendo valores v_1, v_2, \dots, v_n y luego calcula el resultado de aplicar f a dichos valores.

Funciones Aritméticas

■ LISP cuenta, entre otras, con las siguientes funciones aritméticas:

- ➡ $(+ \text{ Arg}_1 \text{ Arg}_2 \dots \text{ Arg}_n)$
- ➡ $(- \text{ Arg}_1 \text{ Arg}_2 \dots \text{ Arg}_n)$
- ➡ $(* \text{ Arg}_1 \text{ Arg}_2 \dots \text{ Arg}_n)$
- ➡ $(/ \text{ Arg}_1 \text{ Arg}_2 \dots \text{ Arg}_n)$
- ➡ $(1+ \text{ Arg})$
- ➡ $(1- \text{ Arg})$
- ➡ $(\text{MAX } \text{ Arg}_1 \text{ Arg}_2 \dots \text{ Arg}_n)$
- ➡ $(\text{MIN } \text{ Arg}_1 \text{ Arg}_2 \dots \text{ Arg}_n)$

QUOTE: “ ` ”

- ❖ Como dijimos anteriormente, cuando **LISP** evalúa una aplicación, evalúa recursivamente cada argumento de la misma.
- ❖ Existen situaciones en las que resulta fundamental **evitar** la evaluación de ciertos argumentos en una aplicación.
- ❖ Por ejemplo, cuando dichos argumentos son listas, pero que no denotan la aplicación de una función sino simplemente una **lista de elementos como estructura de datos**.
- ❖ A tal efecto, **LISP** cuenta con la función **QUOTE** de aridad 1.

Función QUOTE: “ ` ”

❖ Sintaxis:

➡ (QUOTE Arg)

➡ o simplemente puede abreviarse 'Arg

❖ Esta función evita que se evalúe su argumento, y lo retorna intacto como resultado.

> (+ 1 2)

3

> (2 3 5)

ERROR: 2 no es una
función

> `(+ 1 2) (o (QUOTE (+ 1 2)))

(+ 1 2)

> `(2 3 5)

(2 3 5)

Función EVAL

- **EVAL** tiene el efecto contrario que la función **QUOTE**.
- **(EVAL Arg)** fuerza una evaluación adicional de su argumento **Arg**.
- Ejemplo:

<pre>> (QUOTE (+ 2 2)) (+ 2 2)</pre>	<pre>> (EVAL (QUOTE (+ 2 2)) 4</pre>
---	---

Funciones Elementales sobre Listas

■ **LISP** dispone de las siguientes funciones para manipular listas:

➡ **(CAR List)** Retorna el primer elemento de la lista **List**.

➡ **(CDR List)** Retorna la cola de la lista **List**.

■ Por ejemplo:

```
> (CAR `(1 2 3))
```

```
1
```

```
> (CDR `(1 2 3))
```

```
(2 3)
```

Constructores de Listas

❖ Como las listas desempeñan un papel central en **LISP**, existen múltiples formas de construir una lista:

- ➡ **(APPEND List₁ List₂ ... List_n)** Esta función concatena las listas **List₁, List₂, ..., List_n**.
- ➡ **(CONS Elem List)** Retorna una lista cuyo primer elemento es **Elem** y cuya cola es **List**.
- ➡ **(LIST Elem₁ Elem₂ ... Elem_n)** Esta función retorna una lista formada por los elementos **Elem₁, Elem₂, ..., Elem_n**.

Átomos Especiales

- En **LISP**, los siguientes átomos presentan un comportamiento especial.
 - ➡ **T** (verdadero).
 - ➡ **NIL** (falso, indefinido, lista vacía).
 - ➡ Números en punto fijo.
 - ➡ Números en punto flotante.
- Estos átomos son idempotentes con respecto a la evaluación.

La sobrecarga de significados de NIL constituye un aspecto cuestionable del lenguaje.

Predicados en LISP

- Un predicado puede representarse en **LISP** como una función que retorna un valor de verdad.
- Se adopta la convención de que **NIL** denota **falso**, y que cualquier otra s-expresión denota **verdadero**.
- Por ejemplo:
 - ➡ **NIL** equivale a falso.
 - ➡ **T** equivale a verdadero.
 - ➡ **(+ 2 2)** equivale a verdadero.

Predicados en LISP

• LISP dispone de diversos predicados:

- ➡ (ATOM Arg)
- ➡ (EQUAL Arg₁ Arg₂)
- ➡ (MEMBER Elem List)
- ➡ (ZEROP Arg)
- ➡ (MINUSP Arg)
- ➡ (< Arg₁ Arg₂ ... Arg_n)
- ➡ (= Arg₁ Arg₂ ... Arg_n)
- ➡ (> Arg₁ Arg₂ ... Arg_n)

Predicados Lógicos

❖ **LISP** también cuenta con los predicados lógicos tradicionales:

- ➡ **(NOT Arg)** Retorna **T** si, y sólo si, la valuación de **Arg** es **NIL**. En caso contrario retorna **NIL**.
- ➡ **(AND Arg₁ Arg₂ ... Arg_n)** Evalúa sus argumentos de izquierda a derecha. Si encuentra alguno que evalúa a **NIL**, retorna **NIL**, pero si todos evalúan a algo distinto a **NIL** retorna el último valor obtenido.
- ➡ **(OR Arg₁ Arg₂ ... Arg_n)** Análoga a la anterior, sólo retorna **NIL** cuando todos sus argumentos evalúan a **NIL**, retornando en caso contrario el valor del primer argumento que evalúe a algo distinto a **NIL**.

Función Condicional

- Una de las funciones más importantes es la función condicional, pues recordemos que la principal estructura de control en **LISP** es la recursión:

```
(COND (Test1 Result1)  
      (Test2 Result2)  
      ⋮  
      (Testn Resultn))
```

- Esta función va evaluando las diferentes condiciones. En caso de encontrar algún **Test_k** que evalúe a algún valor distinto de **NIL**, retorna la evaluación de **Result_k**.
- Como particularidad, en caso de que todas las condiciones evalúen a **NIL**, entonces retornará **NIL**.
- Si las condiciones **Test_i** son exhaustivas, entonces se suele reemplazar la última (**Test_n**) por **T**.

Definición de Funciones

- Como los datos y programas no difieren entre sí, incluso la definición de nuevas funciones es llevada a cabo mediante la invocación de una función:

(DEFUN FunctionName (Par₁ Par₂ ... Par_n)
FunctionDescription)

- Esta función define una función de nombre **FunctionName**, con argumentos **Par₁**, **Par₂**, ..., **Par_n**, cuyo comportamiento sigue la descripción **FunctionDescription**.

Definición de Funciones

- La función **DEFUN** presenta un comportamiento especial, a saber:
 - ➡ No evalúa sus argumentos.
 - ➡ Retorna como resultado el nombre de la función que acaba de ser definida.
 - ➡ El nombre de la función debe ser un átomo simbólico.
- Ejemplo: definición de la función **suc**.
(DEFUN suc (N)
 (+ N 1)
)

Ejemplos

- ❖ Ejemplo: definición de la función **factorial**.

(DEFUN factorial (N)

(COND

((ZEROP N) 1)

(T

(*

N

(factorial (- N 1))

)

)

)

)

Ejemplos

- ❖ Ejemplo: definición de la función **fibonacci**.

(DEFUN fibo (N)

(COND

((< N 2) 1)

(T (+

(fibo (- N 1))

(fibo (- N 2))

)

)

)

)

Ejemplos

- Ejemplo: definición de la función **largo**.

```
(DEFUN largo (L)
```

```
  (COND
```

```
    ((NULL L) 0)
```

```
    (T      (+ 1 (largo (CDR L)))))
```

```
  )
```

```
)
```