



Universidad
Andrés Bello

Lisp y Prolog

...

Jean Contreras
Sistemas Inteligentes

Instalación en Linux

LISP:

```
$ sudo apt-get install clisp
```

PROLOG:

```
$ sudo add-apt-repository ppa:swi-prolog/stable
```

```
$ sudo apt-get install swi-prolog
```

Lisp

Iniciar lisp:

\$ clisp

```
i i i i i i i      00000 0      0000000 00000 00000
I I I I I I I      8      8 8      8      8 o 8 8
I \ \ '+ ' / I      8      8      8      8      8 8
 \ \ \ '+ ' / /      8      8      8      00000 80000
  \ \ \ '+ ' / /      8      8      8      8 8
   \ \ \ '+ ' / /      8 o 8      8      o 8 8
  -----          00000 8000000 0008000 00000 8

Welcome to GNU CLISP 2.49 (2010-07-07) <http://clisp.cons.org/>

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010

Type :h and hit Enter for context help.

[1]> █
```

Cargar programa lisp:

> (load "cuadrado.lisp")

```
[1]> (load "cuadrado.lisp")  
;; Loading file cuadrado.lisp ...  
;; Loaded file cuadrado.lisp  
T
```

*Ejecutar programa:

> (cuad 5)

```
[2]> (cuad 5)  
25
```

Salir:

> (quit)

```
Break 1 [2]> (quit)  
Adiós.
```

*Se llama a la función principal del programa, no al nombre del archivo.

Hola Mundo:

```
; hello world lisp program.  
(print "Hello World")
```

```
[1]> (load "hello.lisp")  
;; Loading file hello.lisp ...  
"Hello World"  
;; Loaded file hello.lisp  
T
```

Cuadrado de un número:

```
(DEFUN CUAD (N)  
  (* N N))
```

```
[1]> (load "cuadrado.lisp")  
;; Loading file cuadrado.lisp ...  
;; Loaded file cuadrado.lisp  
T  
[2]> (cuad 5)  
25
```

Mover la cabeza de una lista al final:

ABCD -> BCDA

```
(DEFUN ROTAR-IZQ (L)
  (APPEND (CDR L) (LIST (CAR L))))
```

```
[1]> (load "rotarIzq.lisp")
;; Loading file rotarIzq.lisp ...
;; Loaded file rotarIzq.lisp
T
[2]> (rotar-izq '(A B C D))
(B C D A)
```

Las funciones CAR y CDR devuelven la cabeza y la cola de la lista, respectivamente.

APPEND une dos listas.

Factorial:

```
(DEFUN FACT (N)
  (COND ((= N 1) 1)
        (T (* N (FACT (- N 1))))))
```

```
[1]> (load "factorial.lisp")
;; Loading file factorial.lisp ...
;; Loaded file factorial.lisp
T
[2]> (fact 5)
120
```

COND: Condición de término, retorna 1.

T: “Else”.

Lista:

```
(DEFUN SEGUNDO (L)  
  (CAR (CDR L)))
```

```
(DEFUN TERCERO (L)  
  (CAR (CDR (CDR L))))
```

```
(DEFUN PENULTIMO (L)  
  (NTH (- (LENGTH L) 2) L))
```

ABCD -> BCD -> B

ABCD -> BCD -> CD -> C

LENGTH = 4

NTH(4-2) = NTH(2) = C

NTH devuelve el nth elemento de una lista partiendo desde cero.

Lista:

```
(DEFUN SEGUNDO (L)
  (CAR (CDR L)))

(DEFUN TERCERO (L)
  (CAR (CDR (CDR L))))

(DEFUN PENULTIMO (L)
  (NTH (- (LENGTH L) 2) L))
```

```
[1]> (load "listas.lisp")
;; Loading file listas.lisp ...
;; Loaded file listas.lisp
T
[2]> (segundo '(A B C D))
B
[3]> (tercero '(A B C D))
C
[4]> (penultimo '(A B C D))
C
```

NTH devuelve el nth elemento de una lista partiendo desde cero.

Prolog

Iniciar prolog:

\$ prolog

```
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.
```

```
For help, use ?- help(Topic). or ?- apropos(Word).
```

```
?- █
```

Prolog

Cargar programa prolog:

?- consult("parentezco").

```
?- consult("parentezco").  
true.
```

Ejecutar programa:

?- progenitor(pedro,jose).

```
?- progenitor(pedro,jose).  
true .
```

Pertenencia de un valor en una lista:

```
progenitor(pilar,belen).  
progenitor(tomas,belen).  
progenitor(belen,pedro).  
progenitor(pedro,jose).  
progenitor(pedro,maria).
```

```
?- consult("parentezco").  
true.  
  
?- progenitor(pedro,jose).  
true .  
  
?- progenitor(belen,jose).  
false.  
  
?- progenitor(tomas,pedro).  
false.  
  
?- progenitor(pedro,maria).  
true.
```

Longitud de una lista:

```
longitud([],0).
```

```
longitud([_|L],N):-longitud(L,N1), N is N1 + 1.
```

`[_|L]` es una Lista, separado en cabeza y cola.

Hecho, condición de término.

Regla, :- (si ...)

```
?- consult("longitud").  
true.  
  
?- longitud([a,b,c],N).  
N = 3.  
  
?- longitud([a,b,c,d],N).  
N = 4.  
  
?- longitud([1,2,3,4,6],N).  
N = 5.
```

Pertenencia de un valor en una lista:

```
pertenece(X,[X|_]).
```

```
pertenece(X,[_|L]):- pertenece(X,L).
```

X: variable a buscar.

pertenece(X,[X|_]). : Objetivo (True or False).

```
?- consult("pertenece").  
true.  
  
?- pertenece(1, [1,2,3,4,6]).  
true .  
  
?- pertenece(7, [1,2,3,4,6]).  
false.  
  
?- pertenece(a, [a,b,c]).  
true .  
  
?- pertenece(d, [a,b,c]).  
false.
```

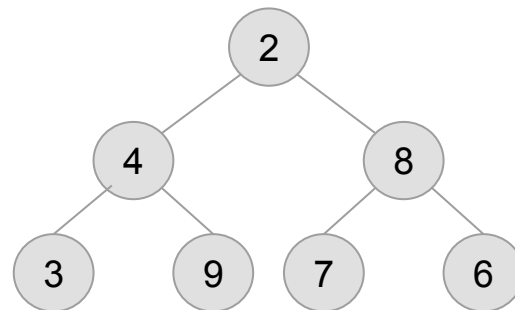
Máximo valor de un árbol:

```
maximo(nil,0).
```

```
maximo(t(I,R,D),M):- maximo(I,MI),maximo(D,MD),M1 is max(MI,MD), M is max(R,M1).
```

nil = Árbol vacío

```
t( t( t(nil,3,nil), 4, t(nil,9,nil)), 2, t( t(nil,7,nil), 8, t(nil,6,nil)))
```



Máximo valor de un árbol:

```
maximo(nil,0).
```

```
maximo(t(I,R,D),M):- maximo(I,MI),maximo(D,MD),M1 is max(MI,MD), M is max(R,M1).
```

```
?- consult("maximo_arbol").
```

```
true.
```

```
?- maximo(t( t( t(nil,3,nil), 4, t(nil,9,nil)), 2, t( t(nil,7,nil), 8, t(nil,6,nil))),M).  
M = 9.
```