Autor: Yolanda Aparicio Ramos
Correo: alu2938@etsii.ull.es

Autor: Yolanda Aparicio Ramos, alu2938@etsii.ull.es
Revisado por: Patricio García Báez. patricio@etsii.ull.es
Asignatura: Introducción a la inteligencia Artificial. 3º Informática Gestión. La Laguna.

= FUNCIONES PRIMITIVAS DE LISP =

== Ayudas y depuración ==

describe: es útil si quieres saber qué argumentos tiene una función, ó a qué paquete corresponde
Ejemplo de uso: (describe 'first)

documentation: Extrae el string de documentación del elemento pasado.

apropos: lista el conjunto de símbolos relativos al argumento del que queremos información.
Ejemplo de uso: (apropos “first”)

trace: Realiza una traza de la función que se le indica cuando es llamada y cuando retorna.

untrace: Desactiva la traza de la función previamente marcada con trace.

step: Ejecuta paso a paso una forma evaluable (expresión que puede ser una lista, una variable, etc.)

== Funciones y Macros ==

defun: crea funciones
Ejemplo de uso: (defun square (x) (* x x))

defmacro: crea macros
Ejemplo de uso: (defmacro square (X) `(* ,X ,X))
Ejemplo de uso: (defmacro square2 (X) `(let ((Temp ,X)) (* Temp Temp)))

lambda: define procedimientos anónimos, es un defun sin nombre del procedimiento, útil para dejar más claro lo que va a hacer el procedimiento. No debe usarse si el procedimiento puede usarse unas cuantas veces.
Ejemplo de uso: (mapcar #'(lambda (x) (eq x 2)) '(3 4 2 5)) -> haz la macro memberp

Ejemplo de uso: ((lambda (x) (list x x)) '(lambda (x) (list x x))) -> programa q se escribe a si mismo

== Asignación ==

setf[1]**:** Asigna o modifica los valores de una variable.
Ejemplo de uso: (setf nueve '9)
(setf l '(a b c))

Autor: Yolanda Aparicio Ramos, alu2938@etsii.ull.es
Revisado por: Patricio García Báez. patricio@etsii.ull.es
Asignatura: Introducción a la inteligencia Artificial. 3º Informática Gestión. La Laguna.

let y let*: Asigna valores a variables y genera listas de tal modo que en let las modificaciones en la muestra de datos en pantalla de los valores son locales y en let* son globales.

Ejemplo de uso: (setf nueve '9)
```
(let ((nueve 'interior)
      (y nueve))
     (list nueve y))
```

Ejemplo de uso: (setf nueve '9)
```
(let* ((nueve 'interior)
       (y nueve))
      (list nueve y))
```

== Aritméticas ==

+, -, *, / : Son las operaciones para sumar, restar, multiplicar y dividir
Un ejemplo de uso es: (+ (* 2 3) 5)

mod : Devuelve el resto de una división en punto flotante.

rem: Devuelve el resto de una división entera

1+, 1- : Incrementa/Decrementa en uno el valor. Es una operación unaria.
Un ejemplo de uso es: (1+ 9)

max, min : Devuelve el valor máximo de una lista
Un ejemplo de uso es: (max 23 4 5 234 654)

float: Devuelve el valor en coma flotante
Ejemplo de uso: (float (/ 22 7))

round: redondea un valor al entero más cercano
Ejemplo de uso: (round (/ 22 7))

expt: calcula potencias elevando su primer argumento a su segundo
Ejemplo de uso: (expt 2 3)

sqrt: calcula raices cuadradas
Ejemplo de uso: (sqrt 9)

abs: calcula el valor absoluto
Ejemplo de uso: (abs -3)

random: calcular un número aleatorio de 0 a N-1
Ejemplo de uso: (random '5)

== Comparación ==

>, <, >=, <= : Mayor que, Menor que, Mayor o igual que, Menor o igual que. Sólo números
Un ejemplo de uso: (< (max 5 6) 6)

/ = : Distinto que. Sólo para números
Ejemplo de uso: (/= 5 4)

equal: Evalúa si el valor de los dos argumentos es la misma expresión
Ejemplo de uso: (equal (+ 2 2) 4)

eql: Evalúa si el valor de los dos argumentos es el mismo símbolo o número.
Ejemplo: Fallará si intentamos (eql '(x y) '(x y))

eq: Evalúa si el valor de los dos argumentos es el mismo símbolo.
Ejemplo: Fallará si intentamos (eq '(x y) '(x y))

=: Evalúa si el valor de los dos argumentos es el mismo número.
Ejemplo: Falla si intentamos ver si (= 'x 'x)

atom: Nos dice si el elemento es un átomo o si es una lista.

endp: Comprueba si la lista está vacía

null: Igual que ende pero su argumento puede no ser una lista.

numberp: Evalúa si es un número.

symbolp: Evalúa si es un símbolo.

zerop: Evalúa si un número es cero.

plusp: Evalúa si un número es positivo.

evenp: Evalúa si un número es par.

oddp: Evalúa si un número es impar.

and, or y not: Son las expresiones lógicas de toda la vida
Ejemplo de uso: (and (< 5 6) T)

if, when, unless y case: Sirven para ejecutar o no acciones si se cumple una condición.
Ejemplos de uso:

```
(setf dia-o-fecha 'lunes)
(if (symbolp dia-o-fecha) 'dia 'fecha)
(when (symbolp dia-o-fecha) 'dia)
(cond ((symbolp dia-o-fecha) 'dia)
      ((numberp dia-o-fecha) 'fecha))
(case figura
      (circulo (* pi r r))
      (esfera (* 4 pi r r)))
```

Autor: Yolanda Aparicio Ramos, alu2938@etsii.ull.es
Revisado por: Patricio García Báez. patricio@etsii.ull.es
Asignatura: Introducción a la inteligencia Artificial. 3º Informática Gestión. La Laguna.

cond: Tiene las mismas características que un case en C/C++. Es lo mismo que poner varias sentencias if anidadas.

== Manejo de Secuencias (tanto listas como cadenas) ==

elt: devuelve el elemento que ocupa la posición x de la secuencia.
Ejemplo de uso: (elt '(a b c) 0)

length: cuenta el número de elementos de nivel superior que hay en una lista.
Ejemplo de uso: (length '(platon socrates aristóteles))

reverse: invierte el orden de los elementos de nivel superior de la secuencia.
Ejemplo de uso: (reverse '(platon socrates aristóteles))

sort[1]: ordena una secuencia en el orden que se le dicta como argumento.
Ejemplo de uso: (sort '(3 1 4 1 5 9) #'<)
Al igual que nconc y delete sort altera el contenido de la lista.

=== Manejo de cadenas ===

string=, string-equal: nos dice si dos cadenas son iguales, la diferencia entre uno y otro es que string= diferencia minúsculas y mayúsculas y string-equal, no.
Ejemplo de uso: (string= "abc" "ABC")

char=, char-equal: nos dice si dos caracteres son iguales, la diferencia entre uno y otro es que char= diferencia minúsculas y mayúsculas y char-equal, no.
Ejemplo de uso: (char= #\a #\a)

search: nos sirve para determinar si una cadena está contenida en otra. Si lo está devolverá la posición donde empieza la correspondencia, en otro caso devuelve NIL.
Ejemplo de uso: (search "Márquez" "García Márquez")
Ejemplo de uso: (search "MÁRQUEZ" "García Márquez" :test #'char-equal) ;; así se evita que distinga entre mayúsculas y minúsculas.

concatenate: concatena cadenas
Ejemplo de uso: (concatenate 'string "hola" "adios")

=== Manejo de listas ===

first: Devuelve el primer elemento de una lista. Es lo mismo que car
Ejemplo de uso: (first amigos)

rest: Devuelve la lista sin el primer elemento. Es lo mismo que cdr
Ejemplo de uso: (rest amigos)

cadr: Entre c y r puede haber una combinación a y d que denotan el encadenamiento de first y rest.
Ejemplo de uso: (caddr amigos) = (first (rest (rest amigos)))

second, third, fourth,...: Devuelve el segundo, tercer, o cuarto elemento de una lista (hay hasta el tenth)
Ejemplo de uso: (third amigos)

nth: Devuelve el n-ésimo elemento de una lista.

nthcdr: Devuelve la lista a partir del número de elementos indicados. Es equivalente a hacer tantos rest como elementos le hemos indicado.
Ejemplo de uso: (nthcdr 3 amigos)

last: devuelve una lista en la que se han eliminado todos los elementos menos el último.
Ejemplo de uso: (last amigos)

cons: Inserta un nuevo elemento en la primera posición de una lista.
Ejemplo de uso: (cons 'edu amigos)
Lo que realmente hace cons es hacer que el primer apuntador de la lista sea 'edu.

list: Elabora una lista con sus argumentos
Ejemplo de uso: (list '1 '2 '3)

append: Combina dos listas en una
Ejemplo de uso: (setf friends (append amigos '(b c)))
Lo que realmente hace append es copiar amigos en la nueva lista (en nuestro caso friends) y, luego anexa '(b c).

nconc[1]: fusiona dos listas en una. La diferencia con append es que nconc lo hace haciendo que el último apuntador de la primera lista apunte al primero de la segunda.
Ejemplo de uso: (nconc amigos '(b c))
Si ahora llamamos a amigos desde clisp el resultado no es el que tal vez esperábamos.

push y pop: meter y sacar un elemento a una lista. Fíjate que podemos usar cons y rest para esos menesteres, la diferencia es que aquí los cambios alteran el contenido de la variable.
Ejemplo de uso: (pop amigos)

butlast: elimina los n últimos elementos de una lista
Ejemplo de uso: (butlast amigos 2)

assoc: Sirve para recuperar elementos de una lista de asociación. Una lista de asociación es una lista de sublistas, en la que el primer elemento de cada sublista se utiliza como una clave para recuperar la sublista completa.
Ejemplo de uso: (setf sara '((estatura 1.7) (peso 65)))
(assoc 'peso sara)

member: comprueba si un elemento pertenece a una lista y devuelve una lista con los elementos que hay desde el elemento coincidente
Ejemplo de uso: (member 'c '(b c a))

delete[1]: se deshace de las ocurrencias del primer argumento que aparezcan en el nivel superior del segundo.
Ejemplo de uso: (delete 'jorge amigos)
Nótese que delete ha borrado del todo a jorge de amigos.

remove: se deshace de las ocurrencias del primer argumento que aparezcan en el nivel superior del segundo. Aparentemente hace lo mismo que delete, pero en la práctica delete cambia el contenido de la lista y remove no.
Ejemplo de uso: (remove 'jorge amigos)
Ejemplo avanzado de uso: (remove '(cara cruz) '((alfa omega) (cara cruz) (zenit nadir)) :test #'equal)
Ejemplo avanzado de uso: (remove 'cruz '((alfa omega) (cara cruz) (zenit nadir)) :test #'member)

== Manejo de expresiones evaluables ==

mapcar: Simplifica las operaciones de transformación de listas. Se proporciona el procedimiento de transformación y la lista de elementos a transformar.
Ejemplo de uso: (mapcar #'oddp '(1 2 3))
(mapcar #'- '(1 2 3 4))

remove-if, remove-if-not: Simplifica las operaciones de filtración de listas. De esta manera, remove-if elimina todos elementos que satisfacen un predicado dado.
Ejemplo de uso: (remove-if #'evenp '(1 2 3 4))

count-if, find-if: Simplifican las operaciones de conteo y localización.
Ejemplo de uso: (count-if #'evenp '(1 2 3 4))
(find-if #'evenp '(1 2 3 4))

funcall: Permite definir procedimientos que tengan procedimientos como argumentos.
Ejemplo de uso: (funcall #'first '(1 2 3))

apply: usa el valor de su primer argumento sobre los elementos de su segundo argumento, el cual debe ser una lista.
Ejemplo de uso: (apply #'append '((e1 e2) (e3 e4)))
(apply #'+ 1 2 3 '(4 5 6))

eval: Función que evalúa una forma y devuelve su resultado.
Ejemplo de uso: (eval (read))

prog1, prog2 y progn: Interpretan secuencias, progn devuelve el resultado de la n-ésima función y prog1 de la primera.

Ejemplo de uso: (progn (setf a 'x) (setf b 'y) (setf c 'z))

quote: Evita que la forma que sigue sea evaluada. Normalmente se simplifica utilizando el apóstrofe ‘ .

backquote: ` Similar al quote, pero permite evaluar ciertas partes de la forma si están precedidas por comas.

== Iteración sobre números y listas ==

dotimes:

```
(dotimes (<contador> <límite-superior> <resultado>)
         <cuerpo del bucle>)
```

dolist:

```
(dolist (<contador> <lista> <resultado>)
        <cuerpo del bucle>)
```

Ejemplo de uso:

```
(dolist (i '(2 3 5 6))
        (if (equal 5 i) (format t "encontrado ~a" i)))
```

do y do*: Son mas generales que dolist y dotimes

Ejemplo de uso:

```
(defun nuevo-expt (m n)
      (do ((resultado 1)                ;;se inicializa parámetros
           (exponente n))
          (when (zerop exponente)       ;;condición de terminación
             (return resultado))
          (setf resultado (* m resultado))  ;;cuerpo del bucle
          (setf exponente (- exponente 1))))
```

loop: Sólo se detiene al encontrarse con un return

```
(loop <cuerpo>)
```

== Funciones de Lectura y Escritura ==

print, format: son operaciones de salida de texto.

Ejemplo de uso: (format t "~%¡Hola!~%Estoy listo para empezar.")

read: es una operación de lectura de texto.

Ejemplo de uso: (setf dato-usuario (read))

read-line: absorbe caracteres hasta donde aparece un retorno de carro o un fin de archivo. Luego produce una cadena con los caracteres q preceden el retorno de carro o el final del archivo, seguido de NIL, al menos que read-line encuentre el final del archivos mientras está leyendo una línea, en ese caso es T.

Ejemplo de uso: (read-line)

Ejemplo de uso:

```
(with-open-file (flujo-de-pacientes "pacientes.lsp" :direction :input)
(dotimes (n 4) (print (read-line flujo-de-pacientes))))
```

Ejemplo de uso:

```
(setq a "line 1
```

```
line2")
        (read-line (setq input-stream (make-string-input-stream a)))
```

read-char: lee un carácter
Ejemplo de uso: (read-char)

with-open-file: permite leer y escribir en archivos
plantilla:

```
(with-open-file (<nombre del flujo>
                <"ruta del archivo">
                :direction <:input o :output>)
    ...)
```

Ejemplo de uso:

```
(with-open-file (flujo-de-pacientes "pacientes.lsp"
                                    :direction :input)
                (do ((paciente (read flujo-de-pacientes nil 'eof)
                               (read flujo-de-pacientes nil 'eof)))
                    ((eq paciente 'eof))
                    (format t "~%¡Hola!~%Estoy listo para empezar. ~a"
(first paciente))))
```

open: permite escribir en archivos
Ejemplo de uso: (defvar *st-local* (open "/tmp/local1" :direction :output :if-exists :rename-and-delete))

== Propiedades ==

get: establece (con setf) y recupera el valor de una propiedad de un símbolo.
Ejemplo de uso: (setf (get 'luis 'padres) '(alfonso monica)) ;; establece
(get 'luis 'padres) ;; recupera

== Estructuras ==

defstruct: Permite definir tipos de datos estructurados basados en números fijos de campos. Como los record de pascal o los struct de c.

== Matrices ==

make-array: crea una matriz con la ayuda de setf
Ejemplo de uso: (setf matriz (make-array '(4 4)))

aref: sirve para recuperar un valor de una matriz
Ejemplo de uso: (setf (aref matriz 0 0) 3)

== Entorno ==

get-universal-time: nos devuelve la fecha actual en formato unix
Ejemplo de uso: (get-universal-time)

machine-type: nos devuelve la familia de maquina que usamos
Ejemplo de uso: (machine-type)

lisp-implementation-version: Devuelve una cadena describiendo la version del lisp utilizada.

lisp-implementation-type: Devuelve una cadena describiendo el tipo de implementación utilizado.

time: nos devuelve el coste computacional de una función.
Ejemplo de uso: (time (+ 2 3))

getenv: nos devuelve el valor de una variable de entorno
Ejemplo de uso: (port:getenv "HOME")

load: Carga el archivo indicado evaluando todas las formas de Lisp incluidas dentro del mismo.

compile-file: Compila un fichero generando código binario. La evaluación de código compilado puede ser unas diez veces más rápida que la del código interpretado

Ejemplo de uso:
(COMPILE-FILE fichero-entrada &key (output-file T) (listing NIL) (warnings *compile-warnings*) (verbose *compile-verbose*) (print *compile-print*))

Para obtener más información, visitar:
http://www.lcc.uma.es/~iaic/funciones-utiles.pdf

UN EJEMPLO DE APLICACIÓN DE LISP

"El problema de los misioneros y los caníbales"

La principal estructura de datos para este problema es una cola con todos los caminos desde el estado inicial al estado final. Un "estado" en este contexto es un registro del número de misioneros y caníbales que están en coda orilla y un conmutador que dice si la barca está en la orilla izquierda o en la derecha. Cada estado está compuesto de una lista de dos tripletas como sigue:

Orilla izquierda Orilla derecha

((M C B) (M C B))

M y C son el número de misioneros y el número de caníbales sobre coda orilla y B es 1 ó 0, dependiendo de si la barca está o no en una orilla particular. Por tanto, el estado original del juego es el siguiente:

((331) (000))

con todos los misioneros, caníbales y la barca en la orilla izquierda. El estado final deseado se representa como:

((000) (331))

Conforme progresa el juego, crece la cola cada vez que se encuentra una transición de estados posibles, y en cada etapa se realiza una comprobación para ver que no ha tenido lugar canibalismo como resultado de la transición realizada. El nombre de esta cola en el programa es q.

Un movimiento, denotado por la variable "movimiento", se da también mediante una tripleta, la cual contiene el número de misioneros en la barca, el número de caníbales en la barca y la constante 1 denotando a la propia barca. Por tanto, por ejemplo, (111) denota a un movimiento en el que la barca contiene un misionero y un caníbal.

La variable "historia" contiene la historia de todos los estados de la orilla izquierda, Los cuales han sido ya tratados anteriormente en el juego. La variable "posible" es una lista de constantes que contiene todas las posibles alternativas para "un movimiento"; esto es, todas las posibles combinaciones de misioneros y caníbales que pueden cruzar el río de una vez.

El programa consta de una función principal "myc" y varias funciones auxiliares "comido", "expandir", "movcorrecto", "mover" y "presentar". Cada una de estas funciones se documenta brevemente en el texto del programa.

---

[1] Esta función viola los principios de la programación funcional ya que puede modificar el valor de las variables.

Autor: Yolanda Aparicio Ramos, alu2938@etsii.ull.es
Revisado por: Patricio García Báez. patricio@etsii.ull.es
Asignatura: Introducción a la inteligencia Artificial. 3º Informática Gestión. La Laguna.

PROGRAMA

```
(defun myc ()

  (prog (q historia) ; inicializa la cola y los posibles movimientos
      (setq posibles '((0 2 1)(0 1 1)(1 1 1)(1 0 1)(2 0 1))))

   (setq q (list (list (list '(3 3 1) '(0 0 0)))))
   repeat ; este bucle se repite hasta que está vacía la orilla izquierda
   (cond ((equal (caaar q) '(0 0 0))
      (return (display (reverse (car q)))))
; desecha un camino si da lugar a canibalismo
; o representa un bucle
      ((or (comido (caar q)) (member (casar q) historia))
         (setq q (cdr q))
         (go repeat))
   )
|
; ahora añade este estado a la historia y pasa
; al siguiente estado
   (setq historia (cons (caar q) historia))
   (setq q (append (expandir (car q) posibles) (cdr q)))
   (go repeat)
]
(defun comido (estado)

; esta función comprueba si existe canibalismo examinando
; la orilla izquierda (car estado). Si allí M es 1 0 2
; y M <>C, entonces hay canibalismo en una u otra orilla.
; Si no, no hay en ninguna.
   (and (or (equal (caar estado) 1) (equal (caar estado) 2))
   (not (equal (caar estado) (cadar estado)))
]
(defun expandir (caminos posibles)
; esta función desarrolla todos los posibles movimientos
; a partir del estado actual.
   (cond ((null posibles) nil)
      ((movcorrecto (car mover) (car posibles))
      (cons (cons (camino (mover (car camino) (car posibles)) camino)
(expandir camino (cdr posibles))))
   (t (expandir camino (cdr posibles)))

   (defun movcorrecto (estado unmovimiento)
; aquí se resta el número de misioneros y caníbales
; que hay en el bote del número que queda
; en la orilla actual, para asegurarse que no se cogen
; más de los que hay.
       (cond ((zerop (caddar estado)) ; ve si bate en la derecha
           (restatodo (cadr estado) unmovimiento))
       (t (restatodo (car estado) unmovimiento))
       (defun restatodo (triple unmovimiento)
; esta función resta loa tres números de un movimiento
; del bate del contenido de una orilla y devuelve
; nil si cualquiera de Las diferencias es <0
           (not (minusp (apply 'min (mapcar ' - triple unmovimiento)
]
(defun mover (estado unmovimiento)
; esta función realiza un movimiento restando
; los números de un movimiento del bote de una orilla
; y sumándolos a la otra.
   (cond ((zerop (caddar estado))
; comprueba si bote en la derecha
    (list (mapcar '+ (car estado) unmovimiento)(mapcar '- (cadr estado) unmovimiento)))
    (t (list (mapcar '- (car estado) unmovimiento)(mapcar '+ (cadr estado) unmovimiento)))
]

(defun display (path)
; esta función presenta la solución resultante
   (con ((null camino) 'end)
   (t (print (car camino))
   (terpri)
   (display (cdr camino))
]
```