LCC 2019

Mauro Montano, Rafael Aloisio

[PROGRAMACIÓN EN LISP]

A continuación se detallarán la implementación de las funciones Lisp requeridas: decisiones de diseño, respecto a las entradas y a su funcionalidad, específicas tanto de la función principal como también de aquellas auxiliares.

INDICE

	Pág.
FUNCIÓN TRANS	
Decisiones de diseño	3
Sobre la entrada	3
Sobre la funcionalidad	3
transAux	3
longitudLista	3
elemLista	3
matrizColumna	4
matrizElem	4
Ejemplos	4
FUNCION SUMAPRIMOS	
Sobre la entrada	5
Sobre la funcionalidad	5
sumaPrimos2	5
esPrimo	6
Ejemplos	6
FUNCIÓN PERMLEX	
Decisiones de diseño	7
Sobre la entrada	7
Sobre la funcionalidad	7
permLex2	7
unir	7
eliminar	8
Ejemplos	8

Función *trans*

DECISIONES DE DISEÑO

Sobre la entrada.

La función principal trans recibe como argumento una matriz. Si es vacía retorna nulo.

Sobre la funcionalidad.

La función trans a partir de una matriz M debe retornar su transpuesta Mt. Para esto delega la función de retornar la traspuesta a una función auxiliar *trans2* llamándola con la matriz, el número 0 y la cantidad de filas de la matriz.

trans2

Objetivo: Función que recibe como argumentos la matriz, la posición de la primer columna (0) y la cantidad de columnas.
Caso base: si hay una única columna, entonces el resultado es una matriz con una sola fila (la columna transpuesta).
Caso recursivo: si hay más de una columna, la transpuesta será agregar una nueva fila al final de Mt la cual será la primer columna de M, y luego repetir el proceso con M sin su primer columna (instancia reducida).
Para lograr esto se van llamando a otras funciones auxiliares:

longitudLista

Objetivo: Función que calcula la longitud de la lista L recibida como argumento.
Caso base: si la lista es vacía el resultado es 0.
Caso recursivo: si la lista no está vacía la longitud de L es la longitud de L más 1 y se repite el proceso con la cola de la lista.

elemLista

Objetivo: Función que obtiene el elemento en la posición P de la lista L.
Caso base: si la posición es igual a 0, devuelve el primer elemento de la lista L
Caso recursivo: si la posición no es 0, llamo recursivamente con la cola de la lista L y con la posición P decrementada en 1.

matrizColumna

Objetivo: Función que devuelve en una lista los elementos de la columna número CP de la matriz M.
Caso base: Si la cola de la matriz es vacía, devuelvo como resultado una lista con el elemento en la posición de CP de la única lista que posee.
Caso recursivo: Sino agrego en la lista un elemento de la columna de número CP y luego llamo recursivamente a matriColumna con la cola de M.

matrizElem

Objetivo: Función que obtiene el elemento de la fila F y la columna C de la matriz M. Para lograrlo se llama a la función elemLista.

EJEMPLOS:

```
[25]> (trans `())
NIL
[26]> (trans (trans `((1 2 3) (4 5 6))))
((1 2 3) (4 5 6))
[27]> (trans `((1 2 3 4) (5 6 7 8)))
((1 5) (2 6) (3 7) (4 8))
[28]> (trans `((1) (2)))
((1 2))
[29]>
```

Función *sumaPrimos*

DECISIONES DE DISEÑO

Sobre la entrada.

Se recibe como único argumento un numero N. Este será controlado por la función *sumaPrimos* de tal forma que se reciba un número N mayor o igual a 0 y menor a 1000 (lo pusimos como tope máximo), en caso de no se cumpla con la entrada requerida se avisará con un cartel.

Sobre la funcionalidad.

Luego de controlar la entrada, si es todo correcto, se llama a la función sumaPrimos2:

sumaPrimos2

La función principal *sumaPrimos2* es la que calcula la suma de todos los números naturales primos hasta un N dado.

Caso base: si N es 0 se detiene la recursión y da como resultado 0.

Caso recursivo 1: si N es primo, la suma de primos de N es igual a N más la suma de primos de N-1.

Caso recursivo 2: si N no es primo, la suma de primos de N es igual a la suma de primos de N-1.

Para saber si N es un número primo, le asigna a X el número 2 y se llama a la función esPrimo con N y X:

esPrimo:

Objetivo: recibe como argumentos el número N y el numero X que es 2 para realizar el módulo de N con X y saber si N es primo.

Caso Base: si N es 1, el resultado es nulo.

Caso Recursivo: se evalúa a N si no existe X tq sea mayor a N/2 y X es divisor de X, en caso contrario no es primo. Entonces si caigo en el primer caso, y luego, si ocurre que el módulo de N y X es 0, entonces N es divisible por X por lo tanto no es primo. En caso de que N no es divisible por X debo verificar que no sea divisible por X+1.

Ejemplos:

```
[2]> (sumaPrimos 10)
17
[3]> (sumaPrimos -1)
(INGRESAR UN NUMERO N VALIDO)
[4]> (sumaPrimos 1500)
(INGRESAR UN NUMERO N VALIDO)
[5]> (sumaPrimos 15)
41
```

Función *permLex*

DECISIONES DE DISEÑO

Sobre la entrada.

Se recibirá como único argumento una lista, que representará un conjunto de letras ya ordenadas, por lo tanto, no se debe hacer ningún control sobre esta.

Sobre la funcionalidad.

La función *permLex*, dada una lista de letras L debe obtener la lista con todas las permutaciones de dichas letras (lista de listas) en orden lexicográfico. Si la lista recibida está vacía retornara vacío, sino delegara su función a permLex2:

permLex2

Objetivo: recibe una lista L que representa a un conjunto de letras, una lista auxiliar vacía, una posición inicialmente en 0 y la posición del último elemento de la lista L, y devuelve la lista de listas que se pide.

Caso Base: si nos encontramos en el último elemento, entonces el resultado es lo que produzca la función 'unir'.

Caso Recursivo: si no estamos en el último elemento, entonces el resultado es concatenar la permutación del primero y las permutaciones de los restantes (instancia reducida). Para lograr esto se llama a la función unir que devuelve una lista con otra lista adentro, y luego con append se forma una lista de listas.

Para lograr la permutación entre los símbolos se utiliza el método auxiliar unir, descripto a continuación.

unir

Objetivo: recibe una lista L que representa a un conjunto de letras, un lista auxiliar vacía, una posición inicialmente en 0 y la posición del último elemento de la lista L. Permuta el primer símbolo de una lista y luego llama a permLex2 con una instancia reducida para que permute esta instancia.

Caso Base: si la lista tiene un único elemento, entonces sus permutaciones son sólo el elemento, entonces devuelve una lista con otra lista dentro, la cual tiene al elemento ese.

Caso Recursivo: si la longitud de la lista es mayor a 1, removemos el elemento en la posición P y lo agregamos al final de lista auxiliar Aux la cual va guardando el resultado de la permutación. Al eliminar un elemento se produce una instancia reducida, y la recursividad cruzada llama a permLex2 el cual permutará los n-1 elementos restantes.

Para eliminar a un elemento de la lista se llama a:

eliminar

Objetivo: Función que elimina el elemento de la posición P de la lista L.
Caso base: si la posición P es igual a 0, llegamos al elemento deseado por lo tanto lo descartamos y retornamos la cola de L.
Caso recursivo: sino agrego el primer elemento de la lista en la nueva lista y llamo a eliminar con la cola de L y la posición decrementada en 1.

Ejemplos:

```
[29]> (permLex `(a b c d))
((A B C D) (A B D C) (A C B D) (A C D B) (A D B C) (A D C B) (B A C D) (B A D C) (B C A D) (B C D A) (B D A C)
 (B D C A) (C A B D) (C A D B) (C B A D) (C B D A) (C D A B) (C D B A) (D A B C) (D A C B) (D B A C) (D B C A)
 (D C A B) (D C B A))
[30]> (permLex `())
NIL
[31]> (permLex `(a b c))
((A B C) (A C B) (B A C) (B C A) (C A B) (C B A))
[32]> (permLex `(a b))
((A B) (B A))
```