**Combine-elt-lst**

**Pseudocódigo**

**Entrada:** elt: elemento a combinar con la lista

lst: lista a combinar con el elemento

**Salida:** Devuelve las listas resultantes de combinar el elemento con cada uno de los elementos de la lista.

**Procesamiento:**

Si el elemento o la lista es null

Nil

En otro caso, usando mapcar combinamos uno a uno cada uno de los elementos de la lista con el elemento pasado como argumento.

**Código**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; combine-elt-lst

;;; Combina un elemento dado con todos los elementos de una lista

(defun combine-elt-lst (elt lst)

(cond ((or (null elt) (null lst))

nil)

(t (mapcar #'(lambda(x) (list elt x)) lst))))

**Comentarios**

En lugar de utilizar la recursión, a partir de la función mapcar y una función lambda que crea una lista a partir de dos elementos vamos recorriendo la lista elemento a elemento y generando todas las listas demandadas.

**Ejemplos**

**Combine-lst-lst**

**Pseudocódigo**

**Entrada:** lst1: primera lista

lst2: segunda lista

**Salida:** lista con las combinaciones del elemento con cada uno de los de la lista

**Procesamiento:**

Si alguna de las listas es null

Nil

En otro caso, utilizando mapcan y una función lambda que va elemento a elemento de la primera lista aplicando combine-elt-lst sobre la segunda lista

**Código**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; combine-lst-lst

;;; Calcula el producto cartesiano de dos listas

(defun combine-lst-lst (lst1 lst2)

(cond ((or (null lst1) (null lst2))

nil)

(t (mapcan #'(lambda(x) (combine-elt-lst x lst2)) lst1))))

**Comentarios**

De igual manera que en el 3.1 aplicamos mapcan y vamos creando todos los pares de combinaciones de las dos listas.

**Ejemplos**

**Combine-list-of-lsts**

**(utilizamos dos funciones auxiliares iguales que las explicadas en el 3.1 y 3.2 pero en lugar de construir los pares con *list* los construimos con *cons*)**

**Pseudocódigo**

**Entrada:** lstolsts: lista de listas

**Salida:** lista con todas las posibles combinaciones de elementos

**Procesamiento:**

Si la lista de listas es null

Nil

En otro caso, llamamos a la función que combina dos listas, cuyos argumentos serán la primera lista y la recursión de la función pero respecto al *rest*  de la lista de listas.

**Código**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; combine-list-of-lsts

;;; Calcula todas las posibles disposiciones de elementos

;;; pertenecientes a N listas de forma que en cada disposicion

;;; aparezca unicamente un elemento de cada lista

(defun combine-list-of-lsts (lstolsts)

(cond ((null lstolsts)

(list nil))

(t (combine-cons-lst-lst (first lstolsts)

(combine-list-of-lsts (rest lstolsts))))))

**Comentarios**

En este caso hemos utilizado la recursión para poder ir combinando todas las listas a partir de la función que da todas las combinaciones de dos listas.

**Ejemplos**