LECTURE 11 - Examples of list processing using MAP functions

MAP Functions - Recap

Usage:

(MAP-function function list-1 list-2 … list-n)

How-to:

MAP-function	get parameters	pack results
MAPCAR	CAR	LIST
MAPLIST	CDR	LIST
MAPCAN	CAR	NCONC
MAPCON	CDR	NCONC
MAPC	CAR	list-1
MAPL	CDR	list-1

1. Consider the following definitions:

```
(defun f (L e)
        (list e L)
)

(setq L '(1 2 3))
(setq e 4)
```

(mapcar #'*f* L e) evaluates to NIL

(mapcar #'(lambda (L) (*f* L e)) L) evaluates to ((4 1) (4 2) (4 3))

2. Consider the following definition:

```
(defun f (L)
        (list L)
)
```

(mapcar #'*f* '(1 2 3)) evaluates to ((1) (2) (3))

(mapcan #'*f* '(1 2 3)) evaluates to (1 2 3)

Notice the equivalence of the following

(mapcan #'*f* L)

(apply #'append (mapcar #'*f* L))

3. Define a function that returns the length of a nonlinear list (in number of atoms at any level).

(LG '(1 (2 (a) c d) (3))) = 6

$$\lg(L) = \begin{cases} 1 & daca \quad L\ e\ atom \\ \sum_{i=1}^{n} \lg(L_i) & daca\ L\ e\ lista\ (L_1 \ldots L_n) \end{cases}$$

```
(DEFUN LG (L)
        (COND
                ((ATOM L) 1)
                (T (APPLY #'+ (MAPCAR 'LG L)))
        )
)
```

4. Define a function that, given a nonlinear list returns the number of sublists (including the list) with even length (at the superficial level).

(NR '(1 (2 (3 (4 5) 6)) (7 (8 9)))) = 4

We will use a auxiliary function that returns T if the argument list has an even number of elements at the surface level, NIL otherwise.

$$nr(L) = \begin{cases} 0 & daca\ L\ e\ atom \\ 1 + \sum_{i=1}^{n} \lg(L_i) & daca\ L\ e\ lista\ (L_1 \ldots L_n)\ si\ n\ e\ par \\ \sum_{i=1}^{n} \lg(L_i) & altfel \end{cases}$$

```
(DEFUN EVEN (L)
    (COND
        ((= 0 (MOD (LENGTH L) 2)) T)
        (T NIL)
    )
)

(DEFUN NR (L)
    (COND
        ((ATOM L) 0)
        ((EVEN L) (+ 1 (APPLY #'+ (MAPCAR #'NR L))))
        (T (APPLY #'+ (MAPCAR #'NR L)))
    )
)
```

5. Define a function that, given a nonlinear list, returns the list of atoms (from any level) in the list.

(ATOMI '(1 (2 (3 (4 5) 6)) (7 (8 9)))) = (1 2 3 4 5 6 7 8 9)

$$atomi(L) = \begin{cases} (L) & daca \quad L \ e \ atom \\ \bigcup_{i=1}^{n} atomi(L_i) & daca \ L \ e \ lista \ (L_1 \ldots L_n) \end{cases}$$

```
(DEFUN ATOMI (L)
    (COND
        ((ATOM L) (LIST L))
        (T (MAPCAN #'ATOMI L))
    )
)
```

Remark: The same requirement could be solved using the MAPCAR function.

```
(DEFUN ATOMI (L)
    (COND
        ((ATOM L) (LIST L))
        (T (APPLY #'APPEND (MAPCAR #'ATOMI L)))
    )
)
```

For the following examples, we let the reader deduce the recursive solution formulas.

6. Define a function that, given a nonlinear list returns the list with all negative numerical atoms at any level removed (keeping the list structure).

```
(ELIMIN '(A (1 B (-1 3 C)) 2 -3)) = (A (1 B (3 C)) 2)
```

Remark: An auxiliary ELIM function will be used (let the reader notice the need to use this function)

```
(DEFUN ELIM (L)
        (COND
                ((AND (ATOM L) (MINUSP L)) NIL)
                ((ATOM L) (LIST L))
                (T (LIST (MAPCAN #'ELIM L)))
        )
)

(DEFUN ELIMIN (L)
        (CAR (ELIM L))
)
```

7. Define a function which, given a nonlinear list, returns T if all sublists (including the list) have even length (at the surface level), or NIL otherwise.

(VERIF '(1 (2 (3 (4 5))))) = T

(VERIF '(1 (2 (3 (4 5 6))))) = NIL

Remark: A function (EVEN L) (defined above) and an auxiliary function (MYAND L) will be used, which having as argument a list consisting only of the values T and NIL checks if all the elements in the list are T.

```
(DEFUN VERIF (L)
    (DEFUN MYAND (L)
        (COND
            ((NULL L) T)
            ((NOT (CAR L)) NIL)
            (T (MYAND (CDR L)))
        )
    )
    (COND
        ((ATOM L) T)
        ((NOT (EVEN L)) NIL)
        (T (FUNCALL #'MYAND (MAPCAR #'VERIF L)))
    )
)
```

8. We could represent a general tree in Lisp as a list of the form

(root subtree1 subtree2) ...)

Define a function which, given a tree, returns the number of nodes in the tree.

```
(NR '(1 (2) (3 (5) (6)) (4))) = 6
```

```
(DEFUN NR (L)
    (COND
        ((NULL (CDR L)) 1)
        (T (+ 1 (APPLY #'+ (MAPCAR #'NR (CDR L)))))
    )
)
```

9. Define a function which, given a tree represented as above, returns the depth of the tree (maximum level - root level is assumed 0).

```
(AD '(1 (2) (3 (5) (6)) (4))) = 2

(DEFUN AD (L)
        (COND
                ((NULL (CDR L)) 0)
                (T (+ 1 (APPLY #'MAX (MAPCAR #'AD (CDR L))))
        )
)
```

10. Define a function which, given a nonlinear list, returns the list of atoms that appear on any level, but in reverse order.

(INVERS '(A (B C (D (E))) (F G))) = (G F E D C B A)

a. recursive, without MAP functions

```
(DEFUN INVERS (L)
   (COND
        ((NULL L) NIL)
        ((ATOM (CAR L)) (APPEND (INVERS (CDR L)) (LIST (CAR L))))
        (T (APPEND (INVERS (CDR L)) (INVERS (CAR L))))
   )
)
```

b. using MAP functions

```
(DEFUN INVERS (L)
        (COND
                ((ATOM L) (LIST L))
                (T (MAPCAN #'INVERS (REVERSE L)))
        )
)
```

11. A matrix can be represented in Lisp as a list whose elements are lists representing the lines of the matrix.

((line1) (line2)….)

Define a function which, given two matrices of order n return their product (as a matrix).

```
(PRODUCT '((1 2) (3 4)) '((2 -1) (3 1))) = ((8 1) (18 1))
```

Remark: We will use two auxiliary functions: a function (COLUMNS L) that returns the list of columns of the parameter matrix L and a function (PR L1 L2) that returns as a matrix the result of multiplying the matrix L1 (list of rows) with the list L2 (a list of columns of a matrix).

```
(DEFUN COLUMNS (L)
    (COND
        ((NULL (CAR L)) NIL)
        (T (CONS (MAPCAR #'CAR L) (COLUMNS (MAPCAR #'CDR L))))
))

(DEFUN PR (L1 L2)
    (COND
        ((NULL (CAR L1)) NIL)
        (T (CONS (MAPCAR #'(LAMBDA (L)
            (APPLY #'+ (MAPCAR #'* (CAR L1) L)) )
            L2)
            (PR (CDR L1) L2)))
))

(DEFUN PRODUCT (L1 L2) (PR L1 (COLUMNS L2)) )
```

12. Write a function to return the number of occurrences of a certain element in a nonlinear list at any level.

(*nrap* 'a '(1 (a (3 (4 a) a)) (7 (a 9)))) = 4

$$nrap(e, l) = \begin{cases} 1 & daca\ l = e \\ 0 & dacă\ l\ e\ atom \\ \sum_{i=1}^{n} nrap(e, l_i) & altfel, l = (l_1 l_2 \dots l_n)\ e\ lista \end{cases}$$

```
(defun nrap(e L)
    (cond
        ((equal L e) 1)
        ((atom L) 0)
        (t (apply #'+ (mapcar #'(lambda(L) (nrap e L) ) l ) ))
    )
)
```

13. Given a nonlinear list, write a function to return the list with all negative numeric atoms removed. Use a MAP function.

Ex: (*stergere* '(a 2 (b -4 (c -6)) -1)) → (a 2 (b (c)))

$$sterg(l) = \begin{cases} \emptyset & daca\ l\ numeric\ negativ \\ l & dacă\ l\ e\ atom \\ sterg(l_i) & altfel, l = (l_1 l_2 \ldots l_n)\ e\ lista \end{cases}$$

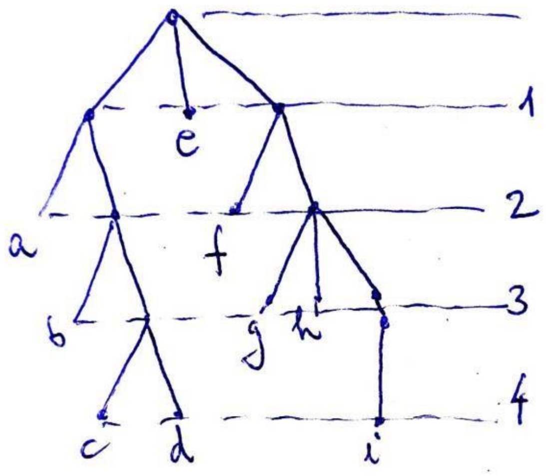
```
(defun sterg(L)
	(cond
		((and (numberp L) (minusp L)) nil)
		((atom L) (list L))
		(t (list (apply #'append (mapcar #'sterg L) ) ))
	)
)

(defun stergere (L)
	(car (sterg L))
)
```

14. Write a function to return the list of atoms at depth n from a non-linear list. The superficial level is assumed 1.

(lista '((a (b (c d))) e (f (g h (i)))) 3) returns (b g h)

(lista '((a (b (c d))) e (f (g h (i)))) 4) returns (c d i)

(lista '((a (b (c d))) e (f (g h (i)))) 5) returns NIL



Recursive model

$$lista(l,n) = \begin{cases} (l) & \text{dacă } n = 0 \text{ si } l \text{ atom} \\ \emptyset & \text{dacă } n = 0 \\ \emptyset & \text{dacă } l \text{ atom} \\ \bigcup_{i=2}^{k} lista(l_i, n-1) & \text{altfel}, l = (l_1 l_2 \dots l_k) \text{ e lista} \end{cases}$$

```
(defun lista(L n)
(cond
    ((and (= n 0) (atom L)) (list L))
    ((= n 0) nil)
    ((atom L) nil)
    (t (mapcan #'(lambda(L) (lista L (- n 1)) ) L))
)
)
```

15. Se dă o mulțime reprezentată sub forma unei liste liniare. Se cere să se genereze lista submulțimilor mulțimii. Se va folosi o funcție MAP.

Ex: (*subm* ‘(1 2)) → (nil (1) (2) (1 2))

<u>Remark</u>

(setq e 1)

(mapcar #' (lambda(L) (cons e L)) '(2 3)) returns ((1 2) (1 3))

```
(defun subm (L)
 (cond
        ((null L) (list nil))
        (t ((lambda (s)
                (append s
                        (mapcar #'(lambda (sb) (cons (car L) sb)) s )
                )
            ) (subm (cdr L) )
           )
        )
  )
)
```

16. Given a set represented as a linear list, write a function to generate the list of permutations of that set. Use a MAP function.

Ex: (*permutari* ‘(1 2 3)) → ((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2) (3 2 1))

```
(defun permutari (L)
 (cond
      ((null (cdr L)) (list L))
      (t (mapcan #'(lambda (e)
                      (mapcar #'(lambda (p) (cons e p) )
                           (permutari (remove e L))
                      )
                 )
                 L
      ))
  )
)
```