Seminar 6 - Functii MAP

Rolul functiilor MAP este de a aplica o functie in mod repetat asupra elementelor (sau sublistelor successive) listelor care sunt date ca si argumente.

Functia MAPCAR este o functie care se aplica *pe rand asupra elementelor listelor* date ca argument, *rezultatele fiind culese intr-o lista intoarsa ca valoare a apelului functiei MAPCAR*. Evaluarea se incheie la terminarea listei celei mai scurte.

Sintaxa: (**MAPCAR function set_of_parameters) : list**

Sa presupunem ca avem implementata functia **triple** care are ca si parametru un numar si ne returneaza numarul inmultit cu valoarea 3.

(DEFUN triple (x) (x 3))*

Cu ajutorul functiei **MAPCAR** vom putea aplica functia **triple** asupra fiecarui element dintr-o lista data ca si parametru.

Astfel, daca avem lista L = (1 2 3 4 5) si scriem:

(MAPCAR triple L) – atunci este echivalent cu aplicarea ***succesiva*** a functiei ***triple asupra elementelor*** listei si ***gruparea rezultatelor*** obtinute intr-o lista.

ð (list (triple 1) (triple 2) … (triple 5)), deci rezultatul va fi: (3 6 … 15)

In situatia in care, in lista noastra vom avea atat atomi nenumerici cat si liste, trebuie sa modificam astfel incat:

- ü daca avem atomi nenumerici sa ne returneze atomii
- ü daca avem subliste, sa se tripleze atomii numerici din subliste – In acest caz, pentru aceasta ramura, vom utiliza functia MAPCAR aplicata asupra elementelor sublistelor

```
(DEFUN triple (x)
(COND
        ((numberp x) (* x 3))
        ((atom x) x)
        (t (mapcar 'triple x))
) )
```

In acest caz, modelul mathematic este:

Daca se doreste ca rezultatul sa fie un numar, se utilizeaza functia **APPLY** care permite aplicarea unei functii asupra unei liste rezultat pentru a calcula un singur rezultat.
Deci, functia APPLY permite aplicarea unei functii asupra unor parametrii furnizati sub forma de lista.

Sintaxa: **(APPLY function set_of_parameters) : val**

Exemple:

(apply ‘+ (1 2 3 4 5)) => 15

(apply ‘max (1 2 3 4 5)) => 5 ; max este lisp functia built-in care returneaza maximul

1. Sa se determine produsul elementelor dintr-o lista, la orice nivel:

L= (2 a 4 (2 b)) => produs = 16

```
(DEFUN product (x)
(COND
        ((numberp x) x)
        ((atom x) 1)
        (t (apply ' * (mapcar  ' product x)))
)
)
```

2. Se da o lista neliniara. Sa se determine numarul sublistelor in care, primul atom numeric, la orice nivel, este numarul 5.

Pentru rezolvare, avem nevoie de o functie care verifica daca primul atom numeric este 5 si de o functie care sa numere sublistele care indeplinesc conditia.

Pentru a verifica daca primul atom numeric este numarul 5, in primul rand trasformam lista noastra neliniara intr-o lista liniara formata doar din valori numerice (eliminam atat atomii cat si liniarizam lista).

```
(DEFUN transform(l)
(COND
        ((null l) nil)
        ((numberp (car l)) (cons (car l) (transform (cdr l))))
        ((atom (car l)) (transform (cdr l)))
        (T (APPEND (transform (car l)) (transform (cdr l))))
)
)
```

Dupa ce am transformat lista, ne trebuie o alta functie care sa verifice daca primul element din lista indeplineste conditia noastra (are valoarea 5)

```
(DEFUN verif (l)
(COND
        ((null (transform l)) nil)
        ((equal (car (transform l)) 5) T)
        (T nil)
```

```
)
)
```

Avand si functia pentru verificare, trebuie sa scriem functia care numara sublistele in care primul atom numeric are valoarea 5.

```
(DEFUN numara (L)
(COND
        ((atom L) 0)
        ((verif L) (+ 1 (apply ' + (mapcar ' numara L))))
        (T (apply ' + (mapcar ' numara L)))
)
)
```

3. Sa se determine numarul nodurilor de pe nivelele pare dintr-un arbore N-ar, reprezentat sub forma: (radacina (arbore1) (arbore2) ... (arboren))

Nivelul radacinii se considera 1.

Deci, pentru rezolvare:

Avem nevoie de un parametru care sa ne determine (arate) nivelul curent:

- Daca nivelul curent este par si avem atom (adica nod), numaram acest nod gasit.

- Daca am gasit nod (atom) dar nivelul nu este par, atunci nu numaram acest nod

- Daca gasim sublista (adica subarbore) atunci continuam in subarbore (cu ajutorul functiei MAPCAR) si incrementam nivelul curent.

Deci modelul matematic este:

In acest caz, pentru aceasta ultima ramura, apelul MAPCAR ar trebui sa arate asa:

MAPCAR ‘noduriPare arbore (+1 nivel) – dar daca vom scrie sub aceasta forma vom obtine

eroare, deoarece in acest caz functia MAPCAR se aplica asupra unei functii cu 2 parametrii.

Când sunt mai mulți parametri, MAPCAR va aplica funcția respectivă pe perechi de elemente din parametri (primul element din prima lista cu primul element din a 2-a listă, al 2-lea element din prima listă cu al 2-lea element din a 2-a listă, etc.) până când se termină lista mai scurtă.

De ex: (mapcar ’list ’(A B C) ’(D E F)) = ((AD) (BE) (CF))

(mapcar ’list ’(A B C) ’(D E)) = ((AD) (BE))

Problema este că la noi parametrul al doilea (*nivel*) e un număr, nu e listă, și dacă încercăm ceva de genul (MAPCAR ‘noduriPare x (+ 1 nivel)) vom avea o eroare, deoarece parametrul al doilea va fi tratat ca listă, și MAPCAR va încerca să ia CAR din parametru.

Ar trebui deci sa avem: (MAPCAR ‘AltaFunctie Arbore)

Soluția în asemenea situații este să folosim o **expersie lambda.**

Expresiile lambda sunt niște funcții, în general simple, care nu au nume, și sunt definite direct acolo unde avem nevoie de ele.

O expresie lambda ne poate ajuta să ”transformăm” funcția noastră care primește 2 parametri, într-o funcție cu un singur parametru.

Ex: lambda (a) (noduriPare a (+ 1 nivel)))

In acest caz, forma functiei noastre este:

```
(DEFUN noduriPare(arb nivel)
(COND
    ((and (atom arb) (= (mod nivel 2) 0)) 1)
    ((atom arb) 0)
    (T (apply '+ ( mapcar ' ( lambda ( a ) ( noduriPare a (+ 1 nivel ) ) )
arb ) ) )
)
)
```

Ne mai trebuie o funcție care să apeleze noduriPare inițializând valoarea parametrului pentru nivel.

```
(DEFUN noduri (arb) (noduriPare arb 0))
```