Lisp 3

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.

(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:

daca avem atomi nenumerici sa ne returneze atomii

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

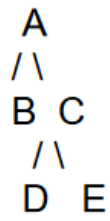
```
(DEFUN triple (x)
(COND
        ((numberp x) (* x 3))
        ((atom x) x)
        (t (mapcar 'triple x))
) )
```

Lisp 2

A binary tree is memorised in the following two ways:

(node no-subtrees list-subtree-1 list-subtree-2 ...) (1)
(node (list-subtree-1) (list-subtree-2) ...) (2)

As an example, the tree



is represented as follows:

(A 2 B 0 C 2 D 0 E 0) (1)
(A (B) (C (D) (E))) (2)

Problema 1 – arbore in forma (1)

Compute the list of nodes accesed in postorder in a tree of type (1).

;parcurg_st(l1l2...ln, nv, nm):{

; Ø, n=0

; Ø, nv = 1 + nm

; l1 U l2 U parcurg_st(l3l4...ln, nv + 1, nm + l2), otherwise

; }

;parcurg_dr(l1l2...ln, nv, nm):{

; Ø, n=0

; l1l2...ln, nv = 1 + nm

; parcurg_dr(l3l4...ln, nv + 1, nm + l2), otherwise

; }

```
;postorder(l1l2...ln):{
;                       Ø, n=0
;                       stang(l1l2...ln) U drept(l1l2...ln) U l1, otherwise
;                   }

;returns the left side of a Type 1 Tree
(defun parcurg_st(arb nv nm)
   (cond
      ((null arb) nil)
      ((= nv (+ 1 nm)) nil)
      (t (cons (car arb)(cons (cadr arb) (parcurg_st (cddr arb) (+ 1 nv) (+ (cadr arb) nm)))))
   )
)

;returns the left side of a Type 1 Tree, skipping the first pair (node, nr_children)
(defun stang(arb)
   (parcurg_st (cddr arb) 0 0)
)

;returns the right side of a Type 1 Tree, skipping the first pair (node, nr_children)
(defun parcurg_dr(arb nv nm)
   (cond
      ((null arb) nil)
      ((= nv (+ 1 nm)) arb)
      (t (parcurg_dr (cddr arb) (+ 1 nv) (+ (cadr arb) nm)))
   )
)

;returns the right side of a Type 1 Tree
(defun drept(arb)
   (parcurg_dr (cddr arb) 0 0)
```

```
)

;returns the postorder traversal of a Type 1 tree
(defun postorder(l)
   (cond
      ((null l) ())
      (t (append (postorder(stang l)) (postorder(drept l)) (list (car l))))
   )
)
(setq l '(a 2 b 2 c 1 i 0 f 1 g 0 d 2 e 0 h 0) )

(print(postorder l))
```

Problema 2 – arbore in forma (2)

. 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.

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)

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))
```