Lisp 2

Dragea Larisa Dragea - 923

```
(print "==========  L2  ==========")

; For a given tree of type (1) return the path from the root node to a certain given node X.

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




; returns the left/right subtree of the tree
(defun left(l)
  (left_traversal (cddr l) 0 0)
)

(defun right(l)
  (right_traversal (cddr l) 0 0)
)




; extract the left child of the current node
(defun left_traversal(l n m)
;l = lista care reprezinta arborele curent
;n = index curent in recursivitate
;m = numarul de noduri deja procesate
    (
        cond
        ((null l) nil)
        ((= n (+ 1 m)) nil)  ; am ajuns la finalul subarborelui stang
        (t (cons (car l) (cons (cadr l) (left_traversal (cddr l) (+ 1 n) (+ (cadr l) m)))))
    )
)

; skips the left subtree and returns the rest of the tree after the current node's children.
(defun right_traversal(l n m)
    (
        cond
        ((null l) nil)
        ((= n (+ 1 m)) l)
        (t (right_traversal (cddr l) (+ 1 n) (+ (cadr l) m)))
    )
)
; checks if an element exists in a list
(defun checkExistence(l elem)
  (cond
    ((null l) nil)
    ((equal (car l) elem) t)
    (t (checkExistence (cdr l) elem))
  )
)

; checks if an element exists in the left subtree of the tree
(defun checkExistenceLeft(l elem)
  (checkExistence (left l) elem)
)

; checks if an element exists in the right subtree of the tree
(defun checkExistenceRight(l elem)
  (checkExistence (right l) elem)
)


; returns the path from the root node to a certain given node X
(defun path(l elem)
  (cond
    ((null l) nil)
    ((equal (car l) elem) (list elem))
    ((checkExistenceRight l elem) (cons (car l) (path (right l) elem)))
    ((checkExistenceLeft l elem) (cons (car l) (path (left l) elem)))
  )
)
(print (path '(A 2 B 0 C 2 D 0 E 0) 'D))
(write-char #\Newline)

;          A
;        /   \
;       B     C
;           / \
;          D   E
```

```
"==========  L2  =========="
(A C D)
```