; 8. Return the list of nodes of a tree of type (2) accessed inorder.

; (car l) - the first element of the list is the root of the tree

; (cadr l) - the second element of the list, at superficial level, is the left subtree

; (caddr l) - the third element of the list, at the superficial level, is the right subtree

; myAppend(l1l2...ln, p1p2...pm) =

; = p1p2...pm, if n = 0

; = {l1} U myAppend(l2...ln, p1p2...pm), otherwise

(defun myAppend (l p)

(cond

((null l) p)

(t (cons (car l) (myAppend (cdr l) p)))

)

)

; inorder(l1l2l3) =

; = nil, if n = 0

; = myAppend(inorder(l2), myAppend(list(l1), inorder(l3))), otherwise

(defun inorder(l)

(cond

((null l) nil)

(t (myAppend (inorder (cadr l)) (myAppend (list (car l)) (inorder (caddr l)))))

)

)

(print (inorder '(A (B) (C (D) (E)))))

L3.

; 8. Write a function to determine the number of nodes on the level k from a n-tree represented as follows:

; (root list\_nodes\_subtree1 ... list\_nodes\_subtreen)

; Eg: tree is (a (b (c)) (d) (e (f))) and k=1 => 3 nodes

; nrNodes(l, level, counter) =

; = 1, if level = counter

; = 0, if level != counter

; = nrNodes(l1, level, counter) + nrNodes(l2, level, counter) + ... + nrNodes(ln, level, counter), otherwise

(defun nrNodes(l level counter)

(cond

((and (atom l) (equal counter level)) 1)

((atom l) 0)

(t (apply '+ (mapcar #' (lambda (a) (nrNodes a level (+ counter 1))) l)))

)

)

(defun main(l level)

(nrNodes l level -1)

)

(print(main '(a (b (c)) (d) (e (f))) 1))