**D.** Given a nonlinear list, write a Lisp function to replace the numerical values on off levels and greater than a given value k to their natural predecessor. The superficial level is assumed 1. **A** **MAP function shall be used.**

**Example** for the list (1 s 4 (3 f (7))) and

**a)** k=0 the result is (0 s 3 (3 f (6))) **b)** k=8 the result is (1 s 4 (3 f (7)))

; f58(l, lvl, k) =

; { l-1 , if l is a number and the level is odd and l>k

; { l, if l is an atom (and the level is even or l<=k)

; { , if l is list, where l=l1l2..ln

; f58(L: list or atom, lvl: integer, k: integer)

(defun f58(L lvl k)

(cond

((AND (numberp L) (> L k) (equal (mod lvl 2) 1)) (- L 1))

((atom L) L)

(t (mapcar #'(lambda (X) (f58 X (+ lvl 1) k) ) L) )

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

; f58Main(tree, k) = f58(tree, 0, k)

; f58Main(tree: list, k: integer)

(defun f58Main(L k)

(f58 L 0 k)

)

**D.** An n-ary tree is represented in Lisp as (node subtree1 subtree2 ...). Write a function to replace all nodes on odd levels with a given value **e**. The root level is assumed zero. **A MAP function** **shall be used.**

**Example** for the tree (a (b (g)) (c (d (e)) (f))) and **e**=h => (a (h (g)) (h (d (h)) (h)))

; Mathematical model

; replaceNode(l, elem, lvl) =

; { elem, if elem is an atom and lvl % 2 = 1

; { l, if elem is an atom (and lvl % 2 = 0)

; { (uniune de la i=1 la n)replaceNode(l\_i, elem, lvl+1), where l = l1l2..ln, if l is list.

; replaceNode(l: list or atom, elem: atom, lvl: integer)

(defun replaceNode(l elem lvl)

(cond

((and (atom l) (equal 1 (mod lvl 2))) elem )

((atom l) l)

(t

(mapcar #'(lambda (X) (replaceNode X elem (+ 1 lvl))) l)

)

)

)

; Mathematical model

; replaceNodeMain(tree, e) = replaceNode(tree, e, -1)

; replaceNodeMain(tree: list, e: atom)

; We start from -1 because we first call the function on the entire tree, which has lvl -1, and then when we reach the superficial level(the root), it will become 0

(defun replaceNodeMain(l e)

(replaceNode l e -1)

)

**D.** An n-ary tree is represented in Lisp as ( node subtree1 subtree2 ...). Write a Lisp program to return the **height** of a node of a tree. **A MAP function shall be used.**

**Example** for the tree (a (b (g)) (c (d (e)) (f)))

**a)** nod=e => the height is 0 **b)** nod=v => the height is -1 c**)** nod=c => the height is 2.

; height(l, elem, found)

; { -1, l is atom

; {max(height(l1,elem,fals),...,height(ln,elem,fals)) , l is list and found=false and l1 != e

; {1+max(height(l1,elem,true),...,height(ln,elem,true)) , l is list si found=false and l1=e

; {1+max(height(l1,elem,true),...,height(ln,elem,true)) , otherwise (l is list and ;found=true)

; height(L: list or atom, e: atom, found: T/NIL)

(defun height(L e found)

(cond

( (atom L) -1 )

( (AND (listp L) (equal found NIL) (not (equal e (car L))))

(apply #'max ( mapcar #'(lambda (x) ( height x e NIL ) ) L ) )

)

( (AND (listp L) (equal found NIL) (equal e (car L)))

(+ 1 (apply #'max ( mapcar #'(lambda (x) ( height x e T ) ) L ) ) )

)

( T (+ 1 (apply #'max( mapcar #' (lambda (x) ( height x e T ) ) L ) )))

)

)

; main(tree, e) = height(tree, e, false)

; main(tree:list, e:element)

(defun main(l e)

(height l e NIL)

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all non-numerical atoms on even levels removed. The superficial level is assumed 1. **A MAP function shall be used.**

**Example** for the list (a (1 (2 b)) (c (d))) the result is (a (1 (2 b)) ((d)))

; Mathematical model

; f61(l, lvl) =

; { [], if l is an atom, not a number, and lvl is even

; { [l], if l is an atom (and lvl%2=1)

; { f61(l\_i, lvl+1)), otherwise (l is list, l = l1l2..ln)

; f61(L:list or atom, lvl: integer)

(DEFUN f61 (L lvl)

(cond

( (AND (atom L) (NOT (NUMBERP L)) (equal 0 (MOD lvl 2))) NIL)

( (atom L) (list L) )

( T (list (mapcan #'(lambda (X) (f61 X (+ 1 lvl))) L) ))

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

; f61Main(tree) = f61(tree, 0)

; f61Main(tree: list)

(defun f61Main(L)

(car (f61 L 0))

)

**D.** An n-ary tree is represented in Lisp as ( node subtree1 subtree2 ...). Write a Lisp function tonreturn the list of nodes on the given level **k**. The root level is assumed zero. **A MAP function** **shall be used. Example** for the tree (a (b (g)) (c (d (e)) (f)))

**a)** k=2 => (g d) **b)** k=5 => ()

; Mathematical model

; f62(l, lvl, k) =

; { [l], if l is an atom, and k=lvl

; { [], if l is an atom (and k!=lvl)

; { f62(l\_i, lvl+1, k)), otherwise (l is list, l = l1l2..ln)

; f62(L:list or atom, lvl: Integer, k: Integer)

(DEFUN f62 (L lvl k)

(cond

( (AND (equal k lvl) (atom L)) (list L))

( (atom L) NIL )

( T (mapcan #'(lambda (X) (f62 X (+ 1 lvl) k)) L))

)

)

; We start from -1 because we first call the function on the entire tree, which has lvl -1, and then when we reach the superficial level(the root), it will become 0

; f62Main(tree, k) = f62(tree, -1, k)

; f62Main(tree: list, k:Integer)

(defun f62Main(L k)

(f62 L -1 k)

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all even numerical atoms from an odd level removed. The superficial level is assumed 1. **A MAP function shall be used.** **Example a)** if the list is (1 (2 A (4 A)) (6)) => (1 (2 A (A)) (6))

**b)** if the list is (1 (2 (C))) => (1 (2 (C)))

; checkCond(l, lvl)=

; {[], l is number and even, lvl is odd

; {[l], l is atom (not number or odd number or even level)

; { checkCond(li, lvl+1)), if l=l1l2l3..ln, otherwise

; checkCond(l: list or atom, lvl: Integer)

(defun checkCond (l lvl)

(cond

((and (numberp l) (=(mod l 2) 0) (= (mod lvl 2) 1) ) nil)

((atom l) (list l))

(t (list(mapcan #' (lambda (v) ( checkCond v (+ 1 lvl))) l )))

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

; mainFunction(tree) = l1, where l1 is the first element of checkCond(tree, 0)

; mainFunction(tree: List)

(defun mainFunction (l)

(car (checkCond l 0))

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all the numerical atoms that are multiple of 3 removed. **A MAP function shall be used.**

**Example a)** if the list is (1 (2 A (3 A)) (6)) => (1 (2 A (A)) NIL)

**b)** if the list is (1 (2 (C))) => (1 (2 (C)))

;Mathematical model: f64(l) =

; { [], if l is atom and l%3==0

; { [l], if l is atom (and l%3 != 0)

; { (U de la i=1 la n)(f64(l\_i)), where l = l1l2..ln

; f64(l: List or atom)

(DEFUN f64 (L)

(cond

( (AND (numberp L) (equal 0 (MOD L 3))) NIL)

( (atom L) (list L) )

( T (list (mapcan #'(lambda (X) (f64 X)) L) ))

)

)

; f64Main(tree) = l1, where l1 is the first element of f64(tree)

; f64(tree: list)

(DEFUN f64Main (L)

(car (f64 L))

)

**D.** Given a nonlinear list, write a Lisp function to replace all the odd values from even levels with their natural successor. The superficial level is assumed 1. **A MAP function shall be used.**

**Example** for the list (1 s 4 (3 f (7))) the result is (1 s 4 (4 f (7))).

;Mathematical model: f65(l, lvl) =

; { l+1, if l is atom and l%2==1 and lvl% 2==0

; { l, if l is atom (and l%2 == 0 or lvl%2 == 1)

; { (U de la i=1 la n)(f65(l\_i, lvl+1)), where l = l1l2..ln, otherwise

; f65(L:list or atom, lvl: Integer)

(DEFUN f65 (L lvl)

(cond

( (AND (numberp L) (equal 1 (mod L 2)) (equal 0 (mod lvl 2))) (+ L 1))

( (atom L) L)

( T (mapcar #'(lambda (X) (f65 X (+ 1 lvl))) L))

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become -1

; f65Main(tree) = f65(tree, 0)

(DEFUN f65Main (L)

(f65 L 0)

)

**D.** An n-ary tree is represented in Lisp as ( node subtree1 subtree2 ...). Write a Lisp function to verify whether a node **x** occurs on an even level of the tree. The root level is assumed zero. **A** **MAP function shall be used.**

**Example** for the tree (a (b (g)) (c (d (e)) (f))) **a) x**=g => T **b)** x=h => NIL

; Mathematical model: f69(l, elem, lvl) =

; { true, if l is atom and l = elem and lvl%2==0

; { false, if l is atom

; { (U de la i=1 la n)(f69(l\_i, elem, lvl+1 )), if l=l1l2..ln is list

; f69(l: list, e: atom, lvl: integer)

; mapcan only appends T to a list, so if x never respects the condition, we will have NIL, otherwise we have (T)

(defun f69(l e lvl)

(cond

( (AND (atom l) (equal e l) (equal 0 (mod lvl 2))) (list T))

( (atom l) NIL )

( T (mapcan #'(lambda (x) (f69 x e (+ 1 lvl))) l ))

)

)

; We start from -1 because we first call the function on the entire tree, which has lvl -1, and then when we reach the superficial level(the root), it will become 0

; f69Main(tree, e) = l1, where l1 is the first element of f69(tree, e, -1)

; f69Main(tree: list, e:atom)

(defun f69Main(tree e)

(car (f69 tree e -1))

)

**D.** Write a Lisp function to substitute all numerical values at any level of a given nonlinear list with a given value **e**. **A MAP function shall be used.**

**Example**, for the list (1 d (2 f (3))), **e**=0 the result is (0 d (0 f (0))).

; Mathematical model f70(l, e) =

; { e , l is atom and number

; { l , l is atom (and not a number)

; { (U de la i=1 la n)(f70(l\_i,e)) , if l=l1l2...ln is list

; f70(l:list, e:element)

(defun f70(l e)

(cond

( (AND (atom l) (numberp l)) e )

( (AND (atom l) (not (numberp l))) l )

( T (mapcar #'(lambda (x) ( f70 x e ) ) l ) )

)

)

**D.** An n-ary tree is represented in Lisp as ( node subtree1 subtree2 ...). Write a Lisp function to determine the path from the root to a given node. **A MAP function shall be used.**

**Example** for the tree (a (b (g)) (c (d (e)) (f)))

**(a)** nod=e => (a c d e) (**b)** nod=v => ()

; Mathematical model isin(elem, l1l2..ln) =

; { false, if n = 0

; { true, if l1 is atom and l1 = elem

; { isin(elem, l1) OR isin(elem, l2..ln), if l1 is list

; { isin(elem, l2..ln), otherwise (l1!=elem)

; isin(elem: atom, l:list)

(defun isin (e l)

(cond

((null l) nil)

((and (atom (car l)) (equal (car l) e)) t)

((listp (car l)) (or (isin e (car l)) (isin e (cdr l))))

( t (isin e (cdr l)))

)

)

; Mathematical model path(l,e) =

; { [l], if l is an atom(node) and l = e

; { [], if l is an atom (and l != e)

; { (uniune de la i=1 la n)(path(l\_i)), if l=l1l2..ln is a list and e is in that list

; { [], otherwise (l is a list that does not contain e)

; path(l: list or atom, e: atom)

(defun path (l e)

(cond

( (and (atom l) (equal l e)) (list l) )

( (atom l) nil)

( (and (listp l) (isin e l ))

(cons (car l) (mapcan #'(lambda (x) (path x e)) (cdr l)))

)

( (listp l) nil)

)

)

**D.** An n-ary tree is represented in Lisp as ( node subtree1 subtree2 ...).. Write a function to return the list of nodes on even levels, in increasing level order (0, 2, …). The root level is assumed zero. **A MAP function shall be used.**

**Example** for the tree (a (b (g)) (c (d (e (h))) (f))) => (a g d f h)

; elim(l, niv) =

; { [l] , l is atom and niv % 2 = 0

; { [] , l is atom (and niv % 2 = 1)

; { (uniune de la i=1 la n)(elim(l\_i, niv+1)), is l=l1l2..ln is list

; elim(l:list or atom, niv:Integer)

(defun elim (l niv)

(cond

( (AND (atom l) (equal (mod niv 2) 0) ) (list l) )

( (atom l) NIL )

(T (mapcan #'(lambda (x) ( elim x (+ niv 1) ) ) l))

)

)

; We start from -1 because we first call the function on the entire tree, which has lvl -1, and then when we reach the superficial level(the root), it will become 0

; main(tree) = elim(tree,-1)

; main(tree:list)

(defun main(tree)

(elim tree -1)

)

**D.** An n-ary tree is represented in Lisp as ( node subtree1 subtree2 ...). Write a Lisp function to replace all nodes on the given level **k** with a given value **e**. The root level is assumed zero. **A** **MAP function shall be used. Example** for the tree (a (b (g)) (c (d (e)) (f))) and **e**=h

**(a)** k=2 => (a (b (h)) (c (h (e)) (h))) **(b)** k=4 => (a (b (g)) (c (d (e)) (f)))

; Mathematical model

; replaceNode(l, elem, lvl, k) =

; { elem, if elem is an atom and lvl = k

; { l, if elem is an atom (and lvl != k)

; { (uniune de la i=1 la n)replaceNode(l\_i), where l = l1l2..ln, if l is list.

; replaceNode(l:list or atom, elem: atom, lvl: integer, k:integer)

(defun replaceNode(l elem lvl k)

(cond

((and (atom l) (equal lvl k)) elem )

((atom l) l)

(t

(mapcar #'(lambda (X) (replaceNode X elem (+ 1 lvl) k)) l)

)

)

)

; We start from -1 because we first call the function on the entire tree, which has lvl -1, and then when we reach the superficial level(the root), it will become 0

; replaceNodeMain(tree, e, k) = replaceNode(tree, e, -1, k)

; replaceNodeMain(tree:list, e: atom, k: integer)

(defun replaceNodeMain(l e k)

(replaceNode l e -1 k)

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all atoms on level **k** removed. The superficial level is assumed 1. **A MAP function shall be used.**

**Example** for the list (a (1 (2 b)) (c (d)))

**a)** k=2 => (a ((2 b)) ((d))) **b)** k=1 => ((1 (2 b)) (c (d))) **c)** k=4 => the list does not change

; Mathematical model

; f75(l, lvl, k) =

; { [], if l is an atom and lvl = k

; { [l], if l is an atom (and lvl != k)

; { uniune( f75(l\_i, lvl+1, k)), otherwise (l is list, l = l1l2..ln)

; f75(L: list, lvl: Integer, k: Integer)

(DEFUN f75 (L lvl k)

(cond

( (AND (atom L) (equal lvl k)) NIL )

( (atom L) (list L) )

( T (list (mapcan #'(lambda (X) (f75 X (+ 1 lvl) k)) L) ))

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

; f75Main(tree, k) = l1, where l1 is the first element of f75(tree,0,k)

; f75Main(tree: list, k: integer)

(defun f75Main(tree k)

(car (f75 tree 0 k))

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all atoms on even levels replaced by zero. The superficial level is assumed 1. **A MAP function shall be used.**

**Example** for the list (a (1 (2 b)) (c (d))) the result is (a (0 (2 b)) (0 (d))).

; Mathematical model

; f76(l, lvl) =

; { 0, if elem is an atom and lvl % 2 = 0

; { l, if elem is an atom (and lvl % 2 != 0)

; { (uniune de la i=1 la n)f76(l\_i, lvl), where l = l1l2..ln, if l is list.

; f76(l: list, lvl: integer)

(defun f76(l lvl)

(cond

((and (atom l) (equal 0 (mod lvl 2))) 0 )

((atom l) l)

(t

(mapcar #'(lambda (X) (f76 X (+ 1 lvl))) l)

)

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

; Mathematical model: replaceNodeMain(tree, e) = replaceNode(tree, e, 0)

; replaceNodeMain(tree: list, e: atom)

(defun f76Main(tree)

(f76 tree 0)

)

**D.** Given a nonlinear list, write a Lisp function to replace all even numerical values with their natural successor. **A MAP function shall be used.**

**Example** for the list (1 s 4 (2 f (7))) the result is (1 s 5 (3 f (7))).

;Mathematical model: f78(l) =

; { l+1, if l is number and l%2==0

; { l, if l is atom (and l not a number or l%2!= 0)

; { (U de la i=1 la n)(f78 l\_i), where l = l1l2..ln, otherwise

; f78(l: list or atom)

(DEFUN f78 (L)

(cond

( (AND (numberp L) (equal 0 (mod L 2)) ) (+ L 1))

( (atom L) L)

( T (mapcar #'(lambda (X) (f78 X)) L))

)

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all atoms on level **k** replaced by **0**. The superficial level is assumed 1. **A MAP function shall be used.**

**Example** for the list (a (1 (2 b)) (c (d))) **(a)** k=2 => (a (0 (2 b)) (0 (d)))

**(b)** k=1 => (0 (1 (2 b)) (c (d))) **(c)** k=4 => the list does not change

; Mathematical model

; f82(l, lvl, k) =

; { 0, if l is an atom and lvl = k

; { l, if l is an atom (and lvl != k)

; { uniune( f82(l\_i, lvl+1, k)), otherwise (l is list, l = l1l2..ln)

; f82(L: list or atom, lvl: integer, k: integer)

(DEFUN f82 (L lvl k)

(cond

( (AND (atom L) (equal lvl k)) 0 )

( (atom L) list L )

( T (mapcar #'(lambda (X) (f82 X (+ 1 lvl) k)) L) )

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

; f82Main(tree, k) = f82(tree, 0, k)

; f82Main(tree: list, k: integer)

(defun f82Main(tree k)

(f82 tree 0 k)

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all occurrences of the element **e** replaced by the value **e1**. **A MAP function shall be used.**

**Example a)** if the list is (1 (2 A (3 A)) (A)), **e** is A and **e1** is B => (1 (2 B (3 B)) (B))

**b)** if the list is (1 (2 (3))) and **e** is A => (1 (2 (3)))

; Mathematical model

; f87(l, e, e1) =

; { e1, if l is an atom and l = e

; { l, if l is an atom (and l != e)

; { uniune( f87(l\_i, e, e1)), otherwise (l is list, l = l1l2..ln)

; f87(l: list or atom, e: atom, e1: atom)

(DEFUN f87 (L e e1)

(cond

( (AND (atom L) (equal l e)) e1 )

( (atom L) l )

( T (mapcar #'(lambda (X) (f87 X e e1)) L) )

)

)

**D.** Write a Lisp function to substitute an element **e** by other element **e1** at all odd levels of a nonlinear list. The superficial level is assumed 1. **A MAP function shall be used.**

**Example**, for the list (1 d (2 d (d))), **e**=d and **e1**=f the result is (1 f (2 d (f))).

; Mathematical model

; f91(l, e, e1, lvl) =

; { e1, if l is an atom and l = e and lvl % 2 == 1

; { l, if l is an atom (and l != e)

; { uniune( f91(l\_i, e, e1, lvl+1)), otherwise (l is list, l = l1l2..ln)

; f91(l: list or atom, e: atom, e1: atom, lvl: integer )

(DEFUN f91 (L e e1 lvl)

(cond

( (AND (atom L) (equal l e) (equal 1 (mod lvl 2))) e1 )

( (atom L) l )

( T (mapcar #'(lambda (X) (f91 X e e1 (+ lvl 1)) ) L) )

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

; f91Main(tree, e, e1) = f91(tree, e, e1, 0)

; f91Main(tree: list, e:atom, e1:atom)

(DEFUN f91Main (tree e e1)

( f91 tree e e1 0 )

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all occurrences of an element **e** removed. **A MAP function shall be used.**

**Example a)** if the list is (1 (2 A (3 A)) (A)) and **e** is A => (1 (2 (3)) NIL)

**b)** if the list is (1 (2 (3))) and **e** is A => (1 (2 (3)))

;Mathematical model: f102(l, e) =

; { [], if l is atom and l=e

; { [l], if l is atom (and l != e)

; { (U de la i=1 la n)(f102(l\_i, e)), where l = l1l2..ln

; f102(L: list, e: atom)

(DEFUN f102 (L e)

(cond

( (AND (atom L) (equal L e)) NIL)

( (atom L) (list L) )

( T (list (mapcan #'(lambda (X) (f102 X e)) L) ))

)

)

; f102Main(tree, e) = l1, where l1 is the first element of f102(tree, e)

; f102Main(tree: list, e:atom)

(DEFUN f102Main (L e)

(car (f102 L e))

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all atoms on the level **k** replaced by 0. The superficial level is assumed 1. **A MAP function shall be used.**

**Example** for the list (a (1 (2 b)) (c (d))) **a)** k=2 => (a (0 (2 b)) (0 (d)))

**b)** k=1 => (0 (1 (2 b)) (c (d))) **c)** k=4 => the list does not change

; Mathematical model

; f103(l, lvl, k) =

; { 0, if l is an atom and lvl=k

; { l, if l is an atom (and lvl != k)

; { (uniune de la i=1 la n)f103(l\_i, lvl+1, k), where l = l1l2..ln, if l is list.

; f103(l:List, lvl:integer, k:integer)

(defun f103(l lvl k)

(cond

((and (atom l) (equal lvl k)) 0 )

((atom l) l)

(t

(mapcar #'(lambda (X) (f103 X (+ 1 lvl) k)) l)

)

)

)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

; Mathematical model: f103Main(tree, k) = f103(tree, 0, k)

; f103Main(tree: list, k: integer)

(defun f103Main(tree k)

(f103 tree 0 k)

)

**D.** Given a nonlinear list, write a Lisp function to return the list with all non-numerical atoms on even levels removed. The superficial level is assumed 1. **A MAP function shall be used.**

**Example** for the list (a (1 (2 b)) (c (d))) the result is (a (1 (2 b)) ((d)))

; Mathematical model

; f109(l, lvl) =

; { [], if l is an atom and not a number and lvl % 2 = 0

; { l, if l is an atom (and lvl % 2 != 0)

; { (uniune de la i=1 la n) f109(l\_i, lvl+1), where l = l1l2..ln, if l is list.

; f109(l:list, lvl: integer)

(defun f109(l lvl)

(cond

((and (atom l) (not (numberp l)) (equal 0 (mod lvl 2))) NIL )

((atom l) (list l))

(t

(list (mapcan #'(lambda (X) (f109 X (+ 1 lvl))) l))

)

)

)

; Mathematical model: f109Main(tree) = l1, where l1 is the first element of f109(tree, 0)

; f109Main(tree: list)

; We start from 0 because we first call the function on the entire tree, which has lvl 0, and then when we reach the superficial level(the root), it will become 1

(defun f109Main(l)

(car (f109 l 0))

)

**D.** An n-ary tree is represented in Lisp as ( node subtree1 subtree2 ...). Write a Lisp function to determine the number of nodes on level **k**. The root level is assumed zero. **A MAP function** **shall be used. Example** for the tree (a (b (g)) (c (d (e)) (f)))

**a)** k=2 => nr=3 (g d f) **b)** k=4 => nr=0 ()

; Mathematical model countNodes(l, lvl, k) =

; {1 , if l is an atom and lvl = k

; {0, if l is an atom (and lvl != k)

; {(suma de la i=1 la n) (countNodes(l\_i,lvl+1,k)), if l=l1l2..ln is list

(defun countNodes(l lvl k)

(cond

((and (atom l) (equal lvl k)) 1)

((atom l) 0)

(t (apply '+ (mapcar #'(lambda (x) (countNodes x (+ 1 lvl) k)) l)))

)

)

; We start from -1 because we first call the function on the entire tree, which has lvl -1, and then when we reach the superficial level(the root), it will become 0

; Mathematical model: countNodesMain(tree, k) = countNodes(tree, -1, k)

; countNodesMain(tree:list, k:integer)

(defun countNodesMain(l k)

(countNodes l -1 k)

)