;PROBLEMA 1

; insereaza2(l1l2...ln, elem, pozitie) =

; = NIL, if n = 0

; = {l1} U {elem} U insereaza2(l2...ln, elem, pozitie + 1) , daca pozitie % 2 = 0

; = {l1} U insereaza2(l2...ln, elem, pozitie + 1) , altfel

(defun insereaza2(l elem pozitie)

(cond

((null l) nil)

((equal (mod pozitie 2) 0) (cons (car l) (cons elem (insereaza2 (cdr l) elem (+ 1 pozitie)))))

(t (cons (car l) (insereaza2 (cdr l) elem (+ 1 pozitie))))

)

)

(defun main1(l elem)

(insereaza2 l elem 1)

)

;b

(defun inverseaza\_total(l)

(cond

((null l) nil)

((listp (car l)) (append (inverseaza\_total(cdr l)) (inverseaza\_total(car l))))

(T (append (inverseaza\_total(cdr l)) (list(car l))))

)

)

;c

(defun cmmdc(a b)

(cond

((not (numberp a)) b)

((not (numberp b)) a)

((= 0 b) a)

((= a b) a)

((> a b) (cmmdc b (- a b)))

(t (cmmdc a (- b a)))

)

)

(defun cmmdc\_lista(l)

(cond

((null l) nil)

((listp (car l)) (cmmdc (cmmdc\_lista (car l)) (cmmdc\_lista (cdr l))))

((not (numberp (car l))) (cmmdc\_lista (cdr l)))

(t (cmmdc (car l) (cmmdc\_lista(cdr l))))

)

)

;d nr\_aparitii\_neliniara

(defun nr\_aparitii(l elem)

(cond

((null l) 0)

((listp (car l)) (+ (nr\_aparitii (car l) elem) (nr\_aparitii (cdr l) elem)))

((equal (car l) elem) (+ 1 (nr\_aparitii (cdr l) elem)) )

(t (nr\_aparitii (cdr l) elem))

)

)

;PROBLEMA 2

;a

(defun elem\_n(l n poz)

(cond

((null l) nil)

((equal n poz) (car l))

(t (elem\_n (cdr l) n (+ poz 1)))

)

)

(defun main1(l n)

(elem\_n l n 1)

)

;b

(defun apartine\_neliniara(l elem)

(cond

((null l) nil)

((listp (car l)) (OR (apartine\_neliniara (cdr l) elem) (apartine\_neliniara (car l) elem)))

((equal (car l) elem) t)

(t (apartine\_neliniara (cdr l) elem))

)

)

;c

(defun subliste (l)

(cond

((atom l) nil)

(T (apply 'append (list l) (mapcar 'subliste l)))

)

)

;d

(defun sterge\_elem(l elem)

(cond

((null l) nil)

((equal (car l) elem) (sterge\_elem (cdr l) elem))

(t (append (list (car l)) (sterge\_elem (cdr l) elem)))

)

)

(defun lista\_multime(l)

(cond

((null l) nil)

(t (cons (car l) (lista\_multime (sterge\_elem l (car l)))))

)

)

;PROBLEMA 3

;a

(defun produs\_vectori(l1 l2)

(cond

((null l1) 0)

(t (+ (\* (car l1) (car l2)) (produs\_vectori (cdr l1) (cdr l2))))

)

)

;b

(defun maxim(a b)

(cond

((> a b) a)

(t b)

)

)

(defun adancime(l contor)

(cond

((null l) contor)

((listp (car l)) (maxim (adancime (car l) (+ contor 1)) (adancime (cdr l) contor)))

(t (adancime (cdr l) contor))

)

)

(defun main2(l)

(adancime l 1)

)

;c

(defun sterge\_elem(l elem)

(cond

((null l) nil)

((equal (car l) elem) (sterge\_elem (cdr l) elem))

(t (append (list (car l)) (sterge\_elem (cdr l) elem)))

)

)

(defun obtine\_min(l min)

(cond

((null l) min)

((< (car l) min) (obtine\_min (cdr l) (car l)))

(t (obtine\_min (cdr l) min))

)

)

(defun main\_min(l)

(obtine\_min l (car l))

)

(defun sort\_fara\_dubluri(l)

(cond

((null l) nil)

(t (append (list (main\_min l)) (sort\_fara\_dubluri (sterge\_elem l (main\_min l)))))

)

)

;d apare\_liniara

(defun apare(l elem)

(cond

((null l) nil)

((equal (car l) elem) t)

(t (apare (cdr l) elem))

)

)

(defun intersectie(l1 l2)

(cond

((null l1) nil)

((apare l2 (car l1)) (cons (car l1) (intersectie (cdr l1) l2)))

(t (intersectie (cdr l1) l2))

)

)

;PROBLEMA 4

;a

(defun lungime(l)

(cond

((null l) 0)

(t (+ 1 (lungime (cdr l))))

)

)

(defun suma\_vectori(l1 l2)

(cond

((not (equal (lungime l1) (lungime l2))) nil)

((null l1) nil)

(t (cons (+ (car l1) (car l2)) (suma\_vectori (cdr l1) (cdr l2))))

)

)

;b

(defun obtine\_atomi(l)

(cond

((null l) nil)

((listp (car l)) (append (obtine\_atomi (car l)) (obtine\_atomi (cdr l))))

(t (cons (car l) (obtine\_atomi (cdr l))))

)

)

;c

(defun inverseaza\_continuele (l aux)

(cond

((null l) aux)

;daca e lista adaug la final de aux, am terminat cu actuala inversare si lipesc car l inversat si cdr l inversat (appenduite)

((listp (car l)) (append aux (append (list (inverseaza\_continuele (car l) nil)) (inverseaza\_continuele (cdr l) nil))))

;adaug practic la inceput de aux -> 1 2 3 va fi 3 2 1 ca adaug la inceput de fiecare data

(t (inverseaza\_continuele (cdr l) (append (list (car l)) aux)))

)

)

;d

(defun maxim(a b)

(cond

((not (numberp a)) b)

((not (numberp b)) a)

((> a b) a)

(t b)

)

)

(defun maxim\_superficial(l)

(cond

((null l) nil)

((numberp (car l)) (maxim (car l) (maxim\_superficial (cdr l))) )

(t (maxim\_superficial (cdr l)))

)

)

;PROBLEMA 5

;a

(defun interclasare\_dubluri(l1 l2)

(cond

((null l1) l2)

((null l2) l1)

((< (car l1) (car l2)) (cons (car l1) (interclasare\_dubluri (cdr l1) l2)))

(t (cons (car l2) (interclasare\_dubluri l1 (cdr l2))))

)

)

;b

(defun inlocuire\_elem(l elem\_vechi lista\_noua)

(cond

((null l) nil)

((listp (car l)) (cons (inlocuire\_elem (car l) elem\_vechi lista\_noua) (inlocuire\_elem (cdr l) elem\_vechi lista\_noua)))

((equal (car l) elem\_vechi) (cons lista\_noua (inlocuire\_elem (cdr l) elem\_vechi lista\_noua)))

(t (cons (car l) (inlocuire\_elem (cdr l) elem\_vechi lista\_noua)))

)

)

;c bafta frt ca e lunga suma

;prima data fac numerele sa aiba acelasi nr de cifre

(defun adauga\_0(l nr\_zerouri)

(cond

((equal nr\_zerouri 0) l)

(t (append '(0) (adauga\_0 l (- nr\_zerouri 1)) ))

)

)

(defun lungime(l)

(cond

((null l) 0)

(t (+ 1 (lungime (cdr l))))

)

)

;l1 e mai scurta si o egalam cu l2

(defun egaleaza(l1 l2)

(cond

((equal (lungime l1) (lungime l2) ) l1)

(t (egaleaza (adauga\_0 l1 1) l2 ))

)

)

;o sa lucrez cu listele inversate

(defun inverseaza\_liniara(l)

(cond

((null l) nil)

(t (append (inverseaza\_liniara (cdr l)) (list (car l)) ))

)

)

;suma\_dummy adica doar adun element cu element, corectez dupa

(defun suma\_dummy(l1 l2)

(cond

((null l1) nil)

(t (append (list (+ (car l1) (car l2)) ) (suma\_dummy (cdr l1) (cdr l2)) ) )

)

)

;inversez liste, le egalez si fac suma\_dummy

(defun pregatire(l1 l2)

(cond

((< (lungime l1) (lungime l2)) (suma\_dummy (inverseaza\_liniara (egaleaza l1 l2)) (inverseaza\_liniara l2) ) )

((> (lungime l1) (lungime l2)) (suma\_dummy (inverseaza\_liniara (egaleaza l2 l1)) (inverseaza\_liniara l1) ) )

((= (lungime l1) (lungime l2)) (suma\_dummy (inverseaza\_liniara l1) (inverseaza\_liniara l2) ) )

)

)

;corectez cele de >9

(defun corecteaza(l)

(cond

((null l) NIL)

((and (> (car l) 9) (null (cdr l))) (append (list (- (car l) 10)) '(1) ) )

((< (car l) 10) (cons (car l) (corecteaza (cdr l))))

(t (cons (- (car l) 10) (corecteaza (cons (+ 1 (car (cdr l))) (cddr l)))))

)

)

;transform din lista in numar, vezi ca lista tre sa fie inversata

(defun lista\_numar(l)

(cond

((null l) 0)

(t (+ (car l) (\* 10 (lista\_numar (cdr l)))))

)

)

;ok acum combin toate prostiile si obtin suma invers sub forma de lista, apoi o transform in nr

(defun main(l1 l2)

(lista\_numar (corecteaza (pregatire l1 l2)))

)

;d

(defun cmmdc(a b)

(cond

((not (numberp a)) b)

((not (numberp b)) a)

((= 0 b) a)

((= a b) a)

((> a b) (cmmdc b (- a b)))

(t (cmmdc a (- b a)))

)

)

(defun cmmdc\_lista(l)

(cond

((null l) nil)

((numberp (car l)) (cmmdc (car l) (cmmdc\_lista(cdr l))))

(t (cmmdc\_lista (cdr l)))

)

)

;PROBLEMA 6

;a

(defun dublare\_n(l n poz)

(cond

((null l) nil)

((equal poz n) (append (list (car l) (car l)) (cdr l)))

(t (append (list (car l)) (dublare\_n (cdr l) n (+ 1 poz))))

)

)

(defun main1(l n)

(dublare\_n l n 1)

)

;b

(defun asociere (l1 l2)

(cond

((null l1) nil)

(t (append (list(cons (car l1) (car l2))) (asociere (cdr l1) (cdr l2))))

)

)

;c

(defun numara\_subliste(l)

(cond

((null l) 1)

((listp (car l)) (+ (numara\_subliste (car l)) (numara\_subliste (cdr l))))

(t (numara\_subliste (cdr l)))

)

)

;d

(defun numara\_numere (l)

(cond

((null l) 0)

((numberp (car l)) (+ 1 (numara\_numere (cdr l))))

(t (numara\_numere (cdr l)))

)

)

;PROBLEMA 7

;a

(defun liniara(l)

(cond

((null l) t)

((listp (car l)) nil)

(t (liniara (cdr l)))

)

)

;b

(defun apare\_neliniara(l elem)

(cond

((null l) nil)

((equal (car l) elem) t)

((listp (car l)) (OR (apare (car l) elem) (apare (cdr l) elem)))

(t (apare (cdr l) elem))

)

)

(defun inlocuire(l el\_v el\_n)

(cond

((null l) nil)

((equal (car l) el\_v) (cons el\_n (cdr l)))

((and (listp (car l)) (apare\_neliniara (car l) el\_v)) (cons (inlocuire (car l) el\_v el\_n) (cdr l)))

(t (cons (car l) (inlocuire (cdr l) el\_v el\_n)))

)

)

;c

(defun inverseaza\_total(l)

(cond

((null l) nil)

((listp (car l)) (append (inverseaza\_total(cdr l)) (inverseaza\_total(car l))))

(T (append (inverseaza\_total(cdr l)) (list(car l))))

)

)

(defun main3(l)

(cond

((null l) nil)

((listp (car l)) (append (list (car (inverseaza\_total (car l)))) (main3 (cdr l)) ))

(t (append (list (car l)) (main3 (cdr l))))

)

)

;d interclasare\_fara\_dubluri

(defun interclasare(l1 l2)

(cond

((null l1) l2)

((null l2) l1)

((< (car l1) (car l2)) (cons (car l1) (interclasare (cdr l1) l2)))

(t (cons (car l2) (interclasare l1 (cdr l2))))

)

)

(defun elimina\_dubluri(l)

(cond

((null l) nil)

((null (cdr l)) (list (car l)))

((equal (car l) (car (cdr l))) (elimina\_dubluri (cdr l)))

(t (append (list (car l)) (elimina\_dubluri (cdr l))))

)

)

(defun main4(l1 l2)

(elimina\_dubluri (interclasare l1 l2))

)

;PROBLEMA 8

;a

(defun elimina\_n(l n poz)

(cond

((null l) nil)

((equal poz n) (elimina\_n (cdr l) n (+ 1 poz)) )

(t (append (list (car l)) (elimina\_n (cdr l) n (+ 1 poz)) ) )

)

)

(defun main1(l n)

(elimina\_n l n 1)

)

;b succesor

(defun inverseaza\_liniara(l)

(cond

((null l) nil)

(t (append (inverseaza\_liniara (cdr l)) (list (car l)) ))

)

)

;cu lista inversata

(defun succesor(l)

(cond

((null l) nil)

((equal (car l) 9) (cons '0 (succesor (cdr l)) ) )

(t (cons (+ 1 (car l)) (cdr l) ))

)

)

;in caz ca trec la nr diferit de cifre

(defun verifica\_0(l)

(cond

((equal (car l) 0) (cons '1 l))

(t l)

)

)

(defun succesor(l)

(verifica\_0 (inverseaza\_liniara (succesor (inverseaza\_liniara l))))

)

;c

(defun liniarizare(l)

(cond

((null l) nil)

((listp (car l)) (append (liniarizare(car l)) (liniarizare(cdr l)) ))

(T (append (list(car l)) (liniarizare (cdr l)) ))

)

)

(defun sterge\_elem(l elem)

(cond

((null l) nil)

((equal (car l) elem) (sterge\_elem (cdr l) elem))

(t (append (list (car l)) (sterge\_elem (cdr l) elem)))

)

)

(defun elimina\_dubluri(l)

(cond

((null l) nil)

((not (equal (cdr l) nil)) (append (list (car l)) (elimina\_dubluri (sterge\_elem (cdr l) (car l))) ))

(t (append (list (car l)) (elimina\_dubluri (cdr l)) ) )

)

)

(defun main(l)

(elimina\_dubluri (liniarizare l))

)

;d

(defun apare(l elem)

(cond

((null l) nil)

((equal (car l) elem) t)

(t (apare (cdr l) elem))

)

)

(defun este\_multime(l)

(cond

((null l) t)

((AND (not (equal (cdr l) nil)) (apare (cdr l) (car l))) nil)

(t (este\_multime (cdr l)))

)

)

;PROBLEMA 9

;a

(defun contine (l e)

(cond

((null l) nil)

((equal (car l) e) T)

(T (contine (cdr l) e))

)

)

(defun diferenta (l k)

(cond

((null l) nil)

((not (contine k (car l))) (cons (car l) (diferenta (cdr l) k)))

(T (diferenta (cdr l) k))

)

)

;b

(defun inverseaza\_neliniara (l)

(cond

((null l) nil)

((listp (car l)) (append (inverseaza\_neliniara (cdr l)) (list (inverseaza\_neliniara (car l)))))

(T (append (inverseaza\_neliniara (cdr l)) (list (car l))))

)

)

;c

(defun lungime(l)

(cond

((null l) 0)

(t (+ 1 (lungime (cdr l))))

)

)

(defun construieste\_lista(L len f)

(cond

((null L) nil)

((atom (car L))

(cond

; lungime impara (si este primul element din lista) => adaug in rezultat elementul

((and (= f 1) (= (mod len 2) 1))

(cons (car L) (construieste\_lista (cdr L) len 0))

)

; trec mai departe

(t (construieste\_lista (cdr L) len f) )

)

)

; primul element e o lista => merg in adancime si continui

(t

(append

(construieste\_lista (car L) (lungime (car L)) 1)

(construieste\_lista (cdr L) len f)

)

)

)

)

(defun main(L)

(construieste\_lista L (lungime L) 1)

)

;d

(defun suma(l)

(cond

((null l) 0)

((listp (car l)) (+ (suma (cdr l)) (suma(car l)) ))

(t (+ (car l) (suma (cdr l))))

)

)

;PROBLEMA 10

;a

(defun suma\_superficial(l)

(cond

((null l) 0)

((numberp (car l)) (+ (car l) (suma\_superficial (cdr l))))

(t (suma\_superficial (cdr l)) )

)

)

;b

(defun fa\_perechi(l elem)

(cond

((null l) nil)

(t (cons (list elem (car l) ) (fa\_perechi (cdr l) elem) ) )

)

)

(defun perechi(l)

(cond

((null l) nil)

((not (equal (cdr l) nil)) (append (fa\_perechi (cdr l) (car l)) (perechi (cdr l)) ))

(t (perechi (cdr l)))

)

)

;c mereu ma uit dupa combinatia semn op1 op2 si inlocuiesc cu rezultatul pana ajung la un sg numar

(defun fa\_operatie(semn nr1 nr2)

(cond

((equal semn '+) (+ nr1 nr2))

((equal semn '\*) (\* nr1 nr2))

((equal semn '-) (- nr1 nr2))

)

)

(defun operatie\_mica(l)

(cond

((equal (cdr l) nil) (car l))

((AND (not (numberp (car l))) (numberp (cadr l)) (numberp (caddr l)) ) ( append (list (fa\_operatie (car l) (cadr l) (caddr l))) (cdddr l) ))

(t (append (list (car l)) (operatie\_mica (cdr l)) ))

)

)

(defun expresie\_preordine(l)

(cond

((null (cdr l)) (car l))

(t (expresie\_preordine (operatie\_mica l) ))

)

)

;d nr\_aparitii\_liniara

(defun nr\_aparitii(l elem)

(cond

((null l) 0)

((equal (car l) elem) (+ 1 (nr\_aparitii (cdr l) elem)))

(t (nr\_aparitii (cdr l) elem))

)

)

(defun sterge\_elem(l elem)

(cond

((null l) nil)

((equal (car l) elem) (sterge\_elem (cdr l) elem))

(t (append (list (car l)) (sterge\_elem (cdr l) elem)))

)

)

(defun perechi\_nr\_aparitii(l)

(cond

((null l) nil)

(t (cons (list (car l) (nr\_aparitii l (car l))) (perechi\_nr\_aparitii (sterge\_elem (cdr l) (car l)) ) ) )

)

)

;PROBLEMA 11

;a

(defun cmmdc(a b)

(cond

((not (numberp a)) b)

((not (numberp b)) a)

((= 0 b) a)

((= a b) a)

((> a b) (cmmdc b (- a b)))

(t (cmmdc a (- b a)))

)

)

(defun cmmmc(a b)

(cond

((not (numberp a)) b)

((not (numberp b)) a)

(t (/ (\* a b) (cmmdc a b)) )

)

)

(defun cmmmc\_lista(l)

(cond

((null l) nil)

((listp (car l)) (cmmmc (cmmmc\_lista (car l)) (cmmmc\_lista (cdr l))))

((not (numberp (car l))) (cmmdc\_lista (cdr l)))

(t (cmmmc (car l) (cmmmc\_lista(cdr l))))

)

)

;b

;0 inseamna ca suntem in scadere 1 in crestere

(defun lungime(l)

(cond

((null l) 0)

(t (+ 1 (lungime (cdr l))))

)

)

(defun munte(l ok)

(cond

((equal (lungime l) 1) t)

((AND (< (car l) (car (cdr l))) (equal ok 1)) (munte (cdr l) 1))

((> (car l) (car (cdr l))) (munte (cdr l) 0))

(t nil)

)

)

;vedem sa nu scadem din prima

(defun main2(l)

(cond

((< (car l) (car (cdr l))) (munte l 1))

(t nil)

)

)

;c

(defun maxim(a b)

(cond

((not (numberp a)) b)

((not (numberp b)) a)

((> a b) a)

(t b)

)

)

(defun maxim\_neliniara(l)

(cond

((null l) nil)

((listp (car l)) (maxim (maxim\_neliniara (car l)) (maxim\_neliniara (cdr l))))

(t (maxim (car l) (maxim\_neliniara (cdr l))))

)

)

;elimina\_elem\_neliniara - sterge\_elem\_neliniara

(defun elimina\_elem(l elem)

(cond

((null l) nil)

((listp (car l)) (cons (elimina\_elem (car l) elem) (elimina\_elem (cdr l) elem )) )

((equal (car l) elem) (elimina\_elem (cdr l) elem))

(t (cons (car l) (elimina\_elem (cdr l) elem) ))

)

)

(defun main3 (l)

(elimina\_elem l (maxim\_neliniara l))

)

;d

(defun e\_par(nr)

(cond

(( AND (numberp nr) (equal (mod nr 2) 0 ) ) t)

(t nil)

)

)

(defun produs\_pare(l)

(cond

((null l) 1)

((listp (car l)) (\* (produs\_pare (car l)) (produs\_pare (cdr l))) )

((e\_par (car l)) (\* (car l) (produs\_pare (cdr l))))

(t (produs\_pare (cdr l)))

)

)

;PROBLEMA 12

;a

(defun produs\_vectori(l1 l2)

(cond

((null l1) 0)

(t (+ (\* (car l1) (car l2)) (produs\_vectori (cdr l1) (cdr l2))))

)

)

;b

(defun maxim(a b)

(cond

((and (not (numberp a)) (not (numberp b))) nil)

((not (numberp a)) b)

((not (numberp b)) a)

((> a b) a)

(t b)

)

)

(defun max\_lista(l)

(cond

((null l) nil)

((AND (equal (lungime l) 1) (atom (car l))) (car l))

((listp (car l)) (maxim (max\_lista (car l)) (max\_lista (cdr l))))

(t (maxim (car l) (max\_lista (cdr l))))

)

)

;c e nasol, intreaba-ma

;ins poz n a unei liste

(defun ins(e n l)

(cond

((= n 1) (cons e l))

(t (cons (car l) (ins e (- n 1) (cdr l))))

)

)

;ins pe toate poz de la n+1 la 1 si combina liste

(defun insert (e n l)

(cond

((= n 0) nil)

(t (cons (ins e n l) (insert e (- n 1) l)))

)

)

;wrapper

(defun inserare(e l)

(insert e (+ (length l) 1) l)

)

;; l- lista de liste!!!

(defun inserareFiecareLista(e l)

(cond

((null l) nil)

;;lista de lista + lista de lista

(t (append (inserare e (car l)) (inserareFiecareLista e (cdr l))))

)

)

(defun permutari(l)

(cond

((null l) (list nil))

(t (inserareFiecareLista (car l) (permutari (cdr l))))

)

)

;d

(defun nr\_par(l)

(cond

((null l) t)

;aici verific sa existe si al doilea element si sa nu fie nil

((not (null (cadr l)) ) (nr\_par (cddr l)))

(t nil)

)

)

;PROBLEMA 13

;a

(defun intercalare\_n(l n elem poz)

(cond

((null l) nil)

((equal poz n) (cons elem l))

(t (cons (car l) (intercalare\_n (cdr l) n elem (+ poz 1))))

)

)

(defun main1(l n elem)

(intercalare\_n l n elem 1)

)

;b

(defun suma(l)

(cond

((null l) 0)

((listp (car l)) (+ (suma (cdr l)) (suma(car l)) ))

((numberp (car l)) (+ (car l) (suma (cdr l))))

(t (+ 0 (suma (cdr l))))

)

)

;c

(defun subliste(l)

(cond

((atom l) nil)

(T (apply 'append (list l) (mapcar 'subliste l)))

)

)

;d

(defun sterge\_prima\_aparitie(l elem)

(cond

((null l) nil)

((equal (car l) elem) (cdr l))

(t (append (list (car l)) (sterge\_prima\_aparitie (cdr l) elem)))

)

)

(defun apare(l elem)

(cond

((null l) nil)

((equal (car l) elem) t)

(t (apare (cdr l) elem))

)

)

(defun egalitate(l1 l2)

(cond

((AND (null l1) (null l2)) t)

((apare l2 (car l1)) (egalitate (cdr l1) (sterge\_prima\_aparitie l2 (car l1))))

(t nil)

)

)

;PROBLEMA 14

;a

(defun elinima\_din\_n\_in\_n(l n ok)

(cond

((null l) nil)

((equal ok 1) (elinima\_din\_n\_in\_n (cdr l) n n))

(t (cons (car l) (elinima\_din\_n\_in\_n (cdr l) n (- ok 1))))

)

)

(defun main1(l n)

(elinima\_din\_n\_in\_n l n n)

)

;b

;0 inseamna ca suntem in scadere 1 in crestere

(defun lungime(l)

(cond

((null l) 0)

(t (+ 1 (lungime (cdr l))))

)

)

(defun vale(l ok)

(cond

((equal (lungime l) 1) t)

((AND (> (car l) (car (cdr l))) (equal ok 0)) (vale (cdr l) 0))

((< (car l) (car (cdr l))) (vale (cdr l) 1))

(t nil)

)

)

;vedem sa nu crestem din prima

(defun main2(l)

(cond

((> (car l) (car (cdr l))) (vale l 0))

(t nil)

)

)

;c

(defun minim(a b)

(cond

((and (not (numberp a)) (not (numberp b))) nil)

((not (numberp a)) b)

((not (numberp b)) a)

((< a b) a)

(t b)

)

)

;min\_lista minim\_lista min\_neliniara minim\_neliniara

(defun min\_lista(l)

(cond

((null l) nil)

((AND (equal (lungime l) 1) (atom (car l))) (car l))

((listp (car l)) (minim (min\_lista (car l)) (min\_lista(cdr l))))

(t (minim (car l) (min\_lista (cdr l))))

)

)

;d elimina\_elem\_liniara - sterge\_elem\_liniara

(defun sterge\_elem(l elem)

(cond

((null l) nil)

((equal (car l) elem) (sterge\_elem (cdr l) elem))

(t (append (list (car l)) (sterge\_elem (cdr l) elem)))

)

)

;max\_lista max\_neliniara maxim\_lista maxim\_neliniara

(defun maxim(a b)

(cond

((and (not (numberp a)) (not (numberp b))) nil)

((not (numberp a)) b)

((not (numberp b)) a)

((> a b) a)

(t b)

)

)

(defun max\_lista(l)

(cond

((null l) nil)

;am ajuns la lungimea 1

((AND (null (cdr l)) (atom (car l))) (car l))

((listp (car l)) (maxim (max\_lista (car l)) (max\_lista(cdr l))))

(t (maxim (car l) (max\_lista (cdr l))))

)

)

(defun main3(l)

(sterge\_elem l (max\_lista l))

)

;PROBLEMA 15

;a

(defun sterge\_prima\_aparitie(l elem)

(cond

((null l) nil)

((equal (car l) elem) (cdr l))

(t (append (list (car l)) (sterge\_prima\_aparitie (cdr l) elem)))

)

)

(defun reuniune(l1 l2)

(cond

((null l1) l2)

((null l2) l1)

(t (cons (car l1) (reuniune (cdr l1) (sterge\_prima\_aparitie l2 (car l1)))))

)

)

;b

(defun produs(l)

(cond

((null l) 1)

((listp (car l)) (\* (produs (cdr l)) (produs(car l)) ))

((numberp (car l)) (\* (car l) (produs (cdr l))))

(t (\* 1 (produs (cdr l))))

)

)

;c

(defun minim(a b)

(cond

((not (numberp a)) b)

((not (numberp b)) a)

((< a b) a)

(t b)

)

)

(defun minim\_neliniara(l)

(cond

((null l) nil)

((listp (car l)) (minim (minim\_neliniara (car l)) (minim\_neliniara (cdr l))))

(t (minim (car l) (minim\_neliniara (cdr l))))

)

)

(defun sort\_dubluri(l)

(cond

((null l) nil)

(t (append (list (minim\_neliniara l)) (sort\_dubluri (sterge\_prima\_aparitie l (minim\_neliniara l)))))

)

)

;d

(defun poz\_elem(l elem poz)

(cond

((null l) nil)

((equal (car l) elem) (append (list poz) (poz\_elem (cdr l) elem (+ 1 poz))))

(t (poz\_elem (cdr l) elem (+ 1 poz)))

)

)

(defun main\_poz\_elem(l elem)

(poz\_elem l elem 0)

)

(defun poz\_min(l)

(poz\_elem l (main\_min l))

)

;extra

;div e 2 la inceput

;nr\_prim

(defun prim(n div)

(cond

((equal div n) t)

((equal (mod n div) 0) nil)

(t (prim n (+ 1 div)))

)

)

(defun main\_prim(n)

(cond

((< n 2) nil)

(t (prim n 2))

)

)