Cabrera Herrera Nathaniel

1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun cic (elem lista pos (&optional (posact 0)))

(let ((aux lista) (ene posact))

(if (/= ene pos)

(progn

(setq ene (1+ ene))

(cic elem (rest aux) pos ene)

)

(if (equal elem (first aux))

T

NIL

)

)

)

) 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun cic (elem lista)

(let ((resultado ())(aux lista))

(if (equal elem (first aux))

(setq resultado aux)

(cic elem (append (rest aux) (list (first aux))))

)

)

)

3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun teren (elem lista &optional (lol 0))

(let ((resultado ())(aux lista))

(if (= lol 0)

(progn

(setq aux (reverse lista))

(setq lol 1)

)

)

(if (and (equal elem (first aux)))

(setq resultado (reverse aux))

(teren elem (append (rest aux) (list (first aux))) lol)

)

)

)

4\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun pimpar (lista &optional (cont 1))

(let ((aux lista)(resultado ()))

(if (oddp (first aux))

(setq resultado (append (list (first aux)) (list cont) ))

(pimpar (rest aux) (1+ cont))

)

)

)

5\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun uelto (lista &optional (answ ()) (man 0) )

(let ((aux lista)(resultado answ) (lol ()))

(if (not (equal aux ()))

(progn

(if (= 0 man)

(progn

(setq man 1)

(setq aux (reverse lista))

)

)

(if (not (equal resultado ()));;si ya tengo el número

(if (equal (first aux) (first resultado))

(progn

(setq resultado (append (list (first aux)) (list (1+ (second resultado)))))

(setq lol (uelto (rest aux) resultado man ))

)

(setq lol (uelto (rest aux) resultado man))

)

)

(if (equal resultado ());;si aun no encuentro el número

(if (and (realp (first aux)) (>= (first aux) 0));;al encuentrar el número

(progn

(setq resultado (append (list (first aux)) (list 1)))

(setq lol (uelto (rest aux) resultado man))

)

(setq lol (uelto (rest aux) resultado man))

)

)

lol

)

resultado

)

)

)

6\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun conteo (lista &optional (answ ())(man 0)) ;;entero - lista

(let ((aux lista)(resultado answ)(lol ()))

(if (not (equal aux ()))

(progn

(if (= 0 man)

(progn

(setq man 1)

(setq resultado (append (list 0) (list 0)))

)

)

(if (integerp (first aux))

(progn

(setq resultado (append (list (1+ (first resultado))) (list (second resultado))))

(setq lol (conteo (rest aux) resultado man))

)

(if (listp (first aux))

(progn

(setq resultado (append (list (first resultado)) (list (1+ (second resultado)))))

(setq lol (conteo (rest aux) resultado man ))

)

(progn

(setq lol (conteo (rest aux) resultado man))

)

)

)

lol

)

resultado

)

)

)

7\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun aplana (lista &optional (answ ()))

(let ((aux lista) (resultado answ)(lol ()))

(if (not (equal aux ()))

(progn

(if (and (listp (first aux)) (> (length (first aux)) 0))

(progn

(setq resultado (append resultado (aplana (first aux))))

(setq lol (aplana (rest aux) resultado ))

)

(progn

(setq resultado (append resultado (list (first aux))))

(setq lol (aplana (rest aux) resultado ))

)

)

lol

)

resultado

)

)

)

8\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun diagonal (lista &optional (n 0)(answ ()))

(let ((aux lista)(resultado answ)(lol ()))

(if (not (equal aux ()))

(progn

(setq resultado (append resultado (list (nth n (first aux)))))

(setq n (1+ n))

(setq lol (diagonal (rest aux) n resultado))

)

resultado

)

)

)

9\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun ver (lista &optional (answ ()))

(let ((aux lista)(resultado answ)(lol ()))

(if (not (equal aux ()))

(progn

(cond

((and (listp (first aux)) (> (length (first aux)) 0)) (setq resultado (append resultado (list 'L))))

((and (atom (first aux)) (not (equal (first aux) ()))) (setq resultado (append resultado (list 'A))))

(T (setq resultado (append resultado (list 'N))))

)

(setq lol (ver (rest aux) resultado))

)

resultado

)

)

)

10\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun sumn (lista &optional (answ 0))

(let ((aux lista)(resultado answ)(lol 0))

(if (not (equal aux ()))

(progn

(setq resultado (+ resultado (first aux)))

(setq lol (sumn (rest aux) resultado))

)

resultado

)

)

)

11\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun fvocales (lista &optional (answ ()))

(let ((aux lista) (resultado answ)(lol ()))

(if (not (equal aux ()))

(progn

(if (and (listp (first aux)) (> (length (first aux)) 0))

(progn

(setq resultado (fvocales (first aux) resultado ))

(setq lol (fvocales (rest aux) resultado ))

)

(if (characterp (first aux))

(progn

(if (or (char-equal (first aux) #\a) (char-equal (first aux) #\e) (char-equal (first aux) #\i) (char-equal (first aux) #\o) (char-equal (first aux) #\u))

()

(setq resultado (append resultado (list (first aux))))

)

(setq lol (fvocales (rest aux) resultado ))

)

(progn

(setq resultado (append resultado (list (first aux))))

(setq lol (fvocales (rest aux) resultado ))

)

)

)

lol

)

resultado

)

)

)

12\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun fm (lista num &optional (answ ())) "filtra multiplos"

(let ((aux lista)(resultado answ)(lol ()))

(if (not (equal aux ()))

(progn

(if (numberp (first aux))

(cond

((/= (mod (first aux) num) 0)

(progn

(setq resultado (append resultado (list (first aux))))

(setq lol (fm (rest aux) num resultado))

)

)

(T

(setq lol (fm (rest aux) num resultado))

)

)

(progn

(setq resultado (append resultado (list (first aux))))

(setq lol (fm (rest aux) num resultado))

)

)

lol

)

resultado

)

)

)

13\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun celdas (lista &optional (answ 0))

(let ((aux lista) (resultado answ)(lol 0))

(if (not (equal aux ()))

(progn

(if (and (listp (first aux)) (> (length (first aux)) 0))

(progn

(setq resultado (celdas (first aux) resultado ))

(setq lol (celdas (rest aux) resultado ))

)

(progn

(setq resultado (1+ resultado))

(setq lol (celdas (rest aux) resultado ))

)

)

lol

)

resultado

)

)

)

14\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

15\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun matrix (lista1 lista2 &optional (answ ())(man 0)) ;;entero - lista

(let ((aux1 lista1)(aux2 lista2)(resultado answ)(lol ())(error 0))

(if (not (equal aux1 ()))

(progn

(if (= 0 man);;verifico que sean compatibles

(progn

(setq man 1)

(if (= (length aux1) (length aux2))

(do ((n 0 (1+ n ))) ((< (1- (length lista1)) n) resultado)

(if (= (length (nth n lista1)) (length (nth n lista2)))

()

(progn

(setq lol ())

(setq error 1)

)

)

)

)

)

)

(if (/= error 1)

(progn

(setq resultado (append resultado (list (mapcar #'\* (first aux1) (first aux2)))))

(setq lol (matrix (rest aux1) (rest aux2) resultado man))

)

lol

)

)

resultado

)

)

)

16\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun busca (elem lista)

(let ((resultado ())(aux lista))

(if (not (equal aux ()))

(if (equal elem (first aux))

(progn

(setq resultado aux)

resultado

)

(busca elem (rest aux))

)

NIL

)

)

)

17\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun cambia (elem1 elem2 lista &optional (answ ()))

(let ((resultado answ)(aux lista))

(if (not (equal aux ()))

(if (equal elem1 (first aux))

(progn

(push elem2 resultado)

(cambia elem1 elem2 (rest lista) resultado)

)

(progn

(push (first aux) resultado)

(cambia elem1 elem2 (rest lista) resultado)

)

)

(reverse resultado)

)

)

)

18\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Naive recursive computation

Evaluation took:

628.177 seconds of real time

625.312500 seconds of total run time (625.234375 user, 0.078125 system)

99.54% CPU

1,316,697,035,834 processor cycles

3,570,512 bytes consed

1. Tail-recursive

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

3,415 processor cycles

0 bytes consed

1. loop-based iterative

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

3,909 processor cycles

0 bytes consed

1. do-based iterative

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

2,823 processor cycles

0 bytes consed

1. CPS computation with the continuation left implicit

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

3,460 processor cycles

0 bytes consed

1. Successive squaring method from SICP

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

5,537 processor cycles

0 bytes consed

1. Dijkstra

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

18,320 processor cycles

0 bytes consed

1. Tail-recursive version, w/o an auxiliary function

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

4,251 processor cycles

0 bytes consed

1. Fast Fibonacci Pair

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

3,647 processor cycles

0 bytes consed

1. Formula de Binet

Evaluation took:

0.000 seconds of real time

0.000000 seconds of total run time (0.000000 user, 0.000000 system)

100.00% CPU

43,167 processor cycles

0 bytes consed

19\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

20\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun aplana (lista &optional (answ ())(nan 0))

(let ((aux lista) (resultado answ)(lol ()))

;;(if (= 0 nan) (progn (setq nan 1) (setq aux (reverse lista))))

(if (not (equal aux ()))

(progn

(if (and (listp (first aux)) (> (length (first aux)) 0))

(progn;;si es lista

(setq resultado (aplana (first aux) resultado nan))

(setq lol (aplana (rest aux) resultado nan))

)

(progn

(setq resultado (push (first aux) resultado ))

(setq lol (aplana (rest aux) resultado nan))

)

)

lol

)

resultado

)

)

)

(defun m-aplana (lista)

(let((aux ()))

(setq aux (reverse (aplana lista)))

aux

)

)