Cabrera Herrera Nathaniel

0\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defmacro prueba (x lista) `(x (first lista) (second lista)))

(prueba + '(1 2))

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

(defun collect (predicado lista &optional (answ ()))

(if (and (functionp predicado) (listp lista));;valido el predicado y lista no vacia

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

(if (not (equal aux ()));;caso base

(if (funcall predicado (first lista))

(progn

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

(setq lol (collect predicado (rest aux) resultado))

lol

)

(progn

(setq lol (collect predicado (rest aux) resultado))

lol

)

)

resultado

)

)

)

)

2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(defun palindromo (lista &optional (answ t))

(if (and (listp lista ) (equal answ t) (> (length lista) 0))

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

(if (equal (list (first aux)) (last aux))

(progn

(pop aux)

(setq aux (reverse aux))

(pop aux)

(setq lol (palindromo aux resultado))

)

(setq lol (palindromo aux (not resultado)))

)

lol

)

answ

)

)

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

(defun 2palindrome (cadena &optional (answ "")(n 0))

(if (and (stringp cadena) (not (equal cadena "")))

(progn

(let ((aux cadena)(resultado answ) (lol ""))

(progn

(setq lol (2palindrome (subseq aux 1) (concatenate 'string (subseq aux 0 1) resultado ) (1+ n)))

(if (= n 0)

(progn

(setq lol (concatenate 'string cadena lol))

)

)

lol

)

)

)

answ

)

)

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

(defun i-palindrome (cadena &optional (answ ""))

(if (and (stringp cadena) (not (equal cadena "")))

(let ((aux cadena)(resultado answ) (lol ""))

(do

((n 0 (1+ n ))) ((<= (length cadena) n) resultado)

(progn

(setq resultado (concatenate 'string (subseq aux 0 1) resultado))

(setq aux (subseq aux 1))

)

)

(setq lol (concatenate 'string cadena resultado))

lol

)

)

)

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

(defun l-rotate (cadena num &key (right nil)(left nil))

(if (and (stringp cadena) (not (equal cadena "")) (numberp num) (or (not (equal right NIL)) (not (equal left NIL))));;verifico tipos de dato

(let ((aux cadena))

(if (and right left)

"Solo selecciona una dirección"

(if (not (equal right NIL))

(progn

(print 'Derecha)

(setq aux (reverse aux))

(do ((n 0 (1+ n))) ((eql n num) aux)

(setq aux (concatenate 'string (subseq aux 1) (subseq aux 0 1)))

)

(setq aux (reverse aux))

)

(progn

(print 'Izquierda)

(do ((n 0 (1+ n))) ((eql n num) aux);;caso derecha

(setq aux (concatenate 'string (subseq aux 1) (subseq aux 0 1)))

)

)

)

)

aux

)

'Error!\_verifica\_los\_parametros

)

)

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

(defun max&pos (lista)

(if (listp lista)

(let ((aux lista)(resultado ())(long 0)(valm 0))

(do ((n 0 (1+ n ))) ((<= (length lista) n) resultado)

(setq valm 0)

(setq long (length (nth n aux)))

(do ((m 0 (1+ m))) ((<= long m) valm)

(if (> (nth m (nth n aux)) valm)

(setq valm (nth m (nth n aux)))

)

)

(push (cons n valm) resultado)

)

(setq resultado (reverse resultado))

)

'Error\_argumentos

)

)

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

(defun combine (funcion lista &optional (answ ()))

(if (and (listp lista) (functionp funcion))

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

(if (not (equal aux ()))

(progn

(if (equal resultado ())

(if (> (length aux) 1)

(progn

(setq resultado (funcall funcion (first aux) (second aux)))

(setq lol (combine funcion (rest (rest aux)) resultado))

)

(progn

(setq resultado (funcall funcion (first aux)))

(setq lol (combine funcion (rest aux) resultado))

)

)

(progn

(setq resultado (funcall funcion (first aux) resultado))

(setq lol (combine funcion (rest aux) resultado))

)

)

lol

)

resultado

)

)

'Error\_argumentos

)

)

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

(defun level (cadena lista &optional (answ nil)(nivel 0))

(if (and (stringp cadena) (listp lista))

(let ((aux lista) (resultado answ)(lol nil)(lev nivel))

(if (not (equal aux ()))

(progn

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

(progn

(setq resultado (level cadena (first aux) resultado (1+ lev)))

(setq lol (level cadena (rest aux) resultado lev))

)

(progn

(if (equal cadena (first aux))

(progn

(setq lol lev)

)

(progn

(setq lol (level cadena (rest aux) resultado lev))

)

)

)

)

lol

)

resultado

)

)

'Error\_argumentos

)

)

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

(defun strencode (cadena &optional (answ ()))

(if (stringp cadena)

(let ((cad cadena)(resultado answ))

(cond ((string= cad "") resultado)

(T (let ((caracter (assoc (char cad 0) resultado)) )

(cond

( (not (null caracter))

(setq resultado (cons (list (char cad 0) (1+ (second caracter))) resultado))

(setq resultado (remove-if (lambda (x) (equal x caracter) )resultado))

(strencode (subseq cad 1) resultado)

)

( T

(setq resultado (cons (list (char cad 0) 1) resultado))

(strencode (subseq cad 1) resultado)

)

)

)

)

)

)

'Error\_argumentos

)

)

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

(defmacro if-positive (expresion &key (then-do) (else-do))

`(cond ((> ,expresion 0) ,then-do)

(T ,else-do)

)

)