; 1. Написать хвостовую рекурсивную функцию my-reverse, которая развернет

; верхний уровень своего списка-аргумента lst

(defun myreverse (lst res)

(cond ((null lst) res )

( T (myreverse (cdr lst) (cons (car lst) res)))

)

)

; (format t "~a~%" (myreverse '(1 (2 11) (3 22) (4 33) 5) Nil) )

; --> (5 (4 33) (3 22) (2 11) 1)

; 2. Написать функцию, которая возвращает первый элемент списка-аргумента,

; который сам является непустым списком.

(defun frst\_el (lst res\_el)

(cond ( (null lst) ())

( (atom (car lst)) (frst\_el (cdr lst) res\_el))

( T (car lst))

)

)

; (format t "~a~%" (frst\_el '(Nil () 1 (2 11) a) Nil) ) ; --> (2 11)

; (format t "~a~%" (frst\_el '(() (2 3 4)) Nil)) ; --> (2 3 4)

; (format t "~a~%" (frst\_el '(()) Nil)) ; --> nil

; 3. Напишите рекурсивную функцию, которая умножает на заданное число-аргумент

; все числа из заданного списка-аргумента, когда

; a) все элемента списка - числа,

; б) элементы списка - любые объекты.

(defun mul (lst num)

(mapcar #'(lambda (x) (\* x num)) lst)

)

; (format t "~a~%" (mul '(1 2 3) 10)) ; --> (10 20 30)

(defun allnums (lst)

(cond ((null lst) Nil)

((not (numberp (car lst))) Nil)

((and (numberp (car lst)) (null (cdr lst))) T)

(T (allnums (cdr lst)))

)

)

; a) все элемента списка - числа,

(defun f3a (lst num)

(cond ((allnums lst) (mul lst num))

(T lst)

)

)

; (format t "~a~%" (f3a '(1 2 3) 10)) ; --> (10 20 30)

; (format t "~a~%" (f3a () 10)) ; --> ()

; (format t "~a~%" (f3a '(1 2 a ) 10)) ; --> (1 2 a)

(defun mulNum (el num)

(cond ((numberp el) (\* el num))

(T el)

)

)

; (format t "~a~%" (mulNum 1 10)) ; --> 10

; (format t "~a~%" (mulNum 'a 10)) ; --> a

; (format t "~a~%" (mulNum '(1 2) 10)) ; --> (1 2)

; б) элементы списка - любые объекты.

(defun f3b (lst num res)

(cond ((null lst) res)

(T (cons (mulNum (car lst) num)

(f3b (cdr lst) num res)))

)

)

; (format t "~a~%" (f3b '(1 2 3) 10 nil)) ; --> (10 20 30)

; (format t "~a~%" (f3b () 10 nil)) ; --> ()

; (format t "~a~%" (f3b '(1 (4) 2 a ) 10 nil)) ; --> (10 (4) 20 a)

; 4. Напишите функцию, select-between, которая из списка-аргумента,

; содержащего только числа, выбирает только те, которые расположены

; между двумя указанными границами аргументами и возвращает их в виде списка

(defun between (x st end)

(cond ( (< st x)

(cond ( (< x end) T)

))

)

)

(defun select\_between (lstnums st end res)

(sort (cond ((null lstnums) res)

( (between (car lstnums) st end)

(select\_between (cdr lstnums)

st

end

(cons (car lstnums) res)) )

(T (select\_between (cdr lstnums) st end res))

) #'<)

)

; (format t "~a~%" ( select\_between '(3 6 4 7) 1 10 nil)) ; --> (3 4 6 7)

; (format t "~a~%" ( select\_between '(30 -6 4 7) 1 10 nil)) ; --> (4 7)

; (format t "~a~%" ( select\_between '(30) 1 10 nil)) ; --> Nil

; 5. Написать рекурсивную версию (с именем rec-add) вычисления суммы чисел заданного списка:

; а) одноуровневого смешанного,

; б) структурированного.

(defun rec\_add\_a (lst sum)

(cond ((null lst) sum)

( (numberp (car lst))

(rec\_add\_a (cdr lst)

(+ (car lst) sum)))

(T (rec\_add\_a (cdr lst)

sum))

)

)

; (format t "~a~%" ( rec\_add\_a '(a 1 b 2 3) 0)) ; --> 6

; (format t "~a~%" ( rec\_add\_a () 0)) ; --> 0

; (format t "~a~%" ( rec\_add\_a '(a b c) 0)) ; --> 0

(defun rec\_add\_b (lst sum)

(cond ((null lst) sum)

( (not (atom (car lst)))

(rec\_add\_b (cdr lst)

(+ (rec\_add\_b (car lst) 0)

sum)))

; (+ (rec\_add\_b (car lst) 0)

; (rec\_add\_b (cdr lst) sum)))

( (numberp (car lst))

(rec\_add\_b (cdr lst)

(+ (car lst) sum)))

(T (rec\_add\_b (cdr lst)

sum))

)

)

; (format t "~a~%" ( rec\_add\_b '(a 1 b 2 3) 0)) ; --> 6

; (format t "~a~%" ( rec\_add\_b () 0)) ; --> 0

; (format t "~a~%" ( rec\_add\_b '(a b c) 0)) ; --> 0

; (format t "~a~%" ( rec\_add\_b '(1 (2) (3 (4 5))) 0)) ; --> 15

; (format t "~a~%" ( rec\_add\_b '( () (a b 1) ((((10))))) 0)) ; --> 11

; 6. Написать рекурсивную версию с именем recnth функции nth.

(defun recnth\_ (n lst i)

(cond ((null lst) nil)

((eql n i) (car lst))

(T (recnth\_ n (cdr lst) (+ i 1)))

)

)

(defun recnth (n lst)

(recnth\_ n lst 0)

)

; (format t "~a~%" ( recnth 0 '(a 1 b 2 3))) ; --> a

; (format t "~a~%" ( recnth 2 '(a 1 b 2 3))) ; --> b

; (format t "~a~%" ( recnth 4 '(a 1 b 2 3))) ; --> 3

; (format t "~a~%" ( recnth 20 '(a 1 b 2 3))) ; --> nil

; 7. Написать рекурсивную функцию allodd, которая возвращает t,

; когда все элементы списка нечетные.

(defun allodd (lst)

(cond ( (and (atom (car lst))

(numberp (car lst))

(oddp (car lst)))

(cond ((null (cdr lst)) T)

(T (allodd (cdr lst)))

))

(T nil)

)

)

; (format t "~a~%" ( allodd '(a 1 b 5 (11 b) 3))) ; --> Nil

; (format t "~a~%" ( allodd ())) ; --> Nil

; (format t "~a~%" ( allodd '(1 11 13 15))) ; --> T

; (format t "~a~%" ( allodd '(1 11 2 13 15))) ; --> Nil

; 8. Написать рекурсивную функцию, которая возвращает первое нечетное число

; из списка (структурированного), возможно создавая некоторые

; вспомогательные функции.

(defun ifnumodd (a)

(and (numberp a) (oddp a))

)

(defun f8 (lst)

(cond ((null lst) nil)

( (atom (car lst))

(cond ((ifnumodd (car lst)) (car lst))

(T (f8 (cdr lst)))

))

(T (or (f8 (car lst))

(f8 (cdr lst))))

)

)

; (format t "~a~%" ( f8 '(a 1 b 5 (11 b) 3))) ; --> 1

; (format t "~a~%" ( f8 ())) ; --> Nil

; (format t "~a~%" ( f8 '(2 22 13 15))) ; --> 13

; (format t "~a~%" ( f8 '(2 22 15))) ; --> 15

; (format t "~a~%" ( f8 '(15))) ; --> 15

; (format t "~a~%" ( f8 '(a b c))) ; --> Nil

; (format t "~a~%" ( f8 '((a b c) () (2 0 4) (2 1) (11)))) ; --> 1

; 9. Используя cons-дополняемую рекурсию с одним тестом завершения,

; написать функцию, которая получает как аргумент список чисел, а возвращает

; список квадратов этих чисел в том же порядке.

(defun f9\_ (lst restlst)

(cond ((null lst) restlst)

(T (cons (\* (car lst) (car lst))

(f9\_ (cdr lst) restlst)))

)

)

(defun f9 (lst)

(f9\_ lst nil)

)

; (format t "~a~%" ( f9 '(1 2 3 4))) ; --> (1 4 9 16)

; (format t "~a~%" ( f9 '())) ; --> ()

; 10. Преобразовать структурированный список в одноуровневый

(defun into\_one (lst res)

(cond ((null lst) res)

((atom lst) (cons lst res))

(T (into\_one (car lst)

(into\_one (cdr lst)

res)))

)

)

(defun f10 (lst)

(into\_one lst nil)

)

; (format t "~a~%" ( f10 '((a b c) () (1 2 3) (4 5) ((((6))))))) ; --> (a b c 1 2 3 4 5 6)

; (format t "~a~%" ( f10 '(((((((10))))))))) ; --> (10)

; (format t "~a~%" ( f10 ())) ; --> ()