Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №7 по дисциплине «Функциональное и логическое программирование»

Тема Рекурсивные функции

Студент Зайцева А. А.

Группа ИУ7-62Б

Оценка (баллы) _____

Преподаватели Толпинская Н.Б., Строганов Ю. В.

Москва — 2022 г.

Практические задания

Используя рекурсию:

1. Написать хвостовую рекурсивную функцию my-reverse, которая развернет верхний уровень своего списка-аргумента lst.

```
(defun move-to (lst result)
  (cond
    ((null lst) result)
    (t (move-to (cdr lst) (cons (car lst) result)))
  )
)

(defun my-reverse (lst)
  (move-to lst ())
)

(my-reverse '(1 a ((2 . 3) 5 6) 7 8)) => (8 7 ((2 . 3) 5 6) A 1)
```

2. Написать функцию, которая возвращает первый элемент списка-аргумента, который сам является непустым списком.

1. Хвостовая рекурсия

```
; дополнительная проверка (listp (cdr x)) нужна из-за следующего результата:
;(listp '(1 . 2)) => T

(defun f (lst)
  (cond
    ((null lst) Nil)
    (
      ((lambda (x) (and (listp x) (listp (cdr x)) (not (null x))))
        (car lst))
      (car lst)
    )
    (t (f (cdr lst)))
  )
)
(f '(0 () (1 . 2) (2 3) (3) a)) => (2 3)
(f '(1)) => Nil
```

2. С помощью функционала find-if

```
(defun f (lst)
  (find-if
   #'(lambda (x)
    (and (listp x) (listp (cdr x)) (not (null x)))
    )
   lst
   )
)
(f '(0 () (1 . 2) (2 3) (3) a)) => (2 3)
(f '(1)) => Nil
```

3. Написать функцию, которая выбирает из заданного списка только числа между двумя заданными границами.

1. Хвостовая рекурсия

```
; а) один cond, но рекурсивный вызов встречается в теле дважды
(defun move_to (lst res a b)
  (cond
    ((null lst) (reverse res))
    ((and (numberp (car lst)) (< a (car lst) b))
      (move_to (cdr lst) (cons (car lst) res) a b))
    ((move_to (cdr lst) res a b))
  )
)
; б) два cond, но рекурсивный вызов встречается в теле единожды
(defun move_to (lst res a b)
  (cond
    ((null lst) (reverse res))
    (t (move_to (cdr lst)
      (cond
        ((and (numberp (car lst)) (< a (car lst) b))
          (cons (car lst) res))
        (res)
      ) a b)
     )
   )
)
(defun select_between (lst a b)
  (cond
    ((< a b) (move_to lst () a b))
    (t (move_to lst () b a))
  )
)
```

2. Рекурсия, которая собирает результат на выходе.

```
(defun select_between (lst a b)
  (cond
    ((null lst) Nil)
    ((and (numberp (car lst)) (< a (car lst) b)) (cons (car lst) (
       select_between (cdr lst) a b)))
    (t (select_between (cdr lst) a b))
  )
)
(select_between '(0 3 7 a 5 (4 3) 1 6) 1 6) => (3 5)
```

4. Напишите рекурсивную функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда:

1. Все элементы списка – числа.

```
; а) с помощью рекурсии, собирающей результат на выходе
(defun mult_all (lst num)
  (cond
    ((null lst) Nil)
    (t (cons (* (car lst) num) (mult_all (cdr lst) num)))
  )
)

; б) через хвостовую рекурсию
(defun move_to (lst res num)
  (cond
    ((null lst) (reverse res))
    (t (move_to (cdr lst) (cons (* (car lst) num) res) num))
  )
)

(defun mult_all (lst num)
  (move_to lst () num)
)

(mult_all '(0 10 -10 5.5 2/3) 2) => (0 20 -20 11.0 4/3)
```

2. Элементы списка – любые объекты. С помощью рекурсии, собирающей результат на выходе.

а) Работа только по верхнему уровню.

```
; а) с помощью рекурсии, собирающей результат на выходе
(defun mult_all (lst num)
  (cond
    ((null lst) Nil)
    ((numberp (car lst)) (cons (* (car lst) num) (mult_all (cdr lst)
       num)))
    (t (cons (car lst) (mult_all (cdr lst) num)))
  )
)
(mult_all '(0 a "abc" (1 k) 2/3 ((1 . 2) . 3)) 2) => (0 A "abc" (1 K)
   4/3 ((1 . 2) . 3))
```

б) Работа по всем уровням структурированного списка.

```
; Для определения того, является ли x точечной парой, используется проверка (atom (cdr x))
; вместо (consp x) из-за следующего результата:
; (listp '(k . 10))=(consp '(k . 10)) => T

; вспомогательная функция, для точечных пар
(defun mult_all_cons (cns num)
  (cond
    ((and (numberp (car cns)) (numberp (cdr cns))) (cons (* (car cns)
       num) (* (cdr cns) num)))
    ((numberp (car cns)) (cons (* (car cns) num) (cdr cns)))
    ((and (numberp (cdr cns)) (atom (car cns))) (cons (car cns) (* (cdr
        cns) num)))
    ((and (numberp (cdr cns)) (consp (car cns))) (cons (mult_all_cons (
       car cns) num) (* (cdr cns) num)))
    ((consp (car cns)) (cons (mult_all_cons (car cns) num) (cdr cns)))
    (t (cons (car cns) (cdr cns)))
  )
)
(defun mult_all (lst num)
  (cond
    ((null lst) Nil)
    ((numberp (car lst)) (cons (* (car lst) num) (mult_all (cdr lst)
       num)))
    ((atom (car lst)) (cons (car lst) (mult_all (cdr lst) num)))
    ((atom (cdr (car lst))) (cons (mult_all_cons (car lst) num) (
       mult_all (cdr lst) num)))
    (t (cons (mult_all (car lst) num) (mult_all (cdr lst) num)))
  )
)
(mult_all '(0 a "abc" (1 k) 2/3 ((1 . 2) . 3)) 2) => (0 A "abc" (2 K)
   4/3 ((2 . 4) . 6))
```

5. 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 (упорядоченного по возрастанию списка чисел (+ 2 балла)).

(Границы не включительно)
Сортировка по невозрастанию:

```
; блочная (карманная, корзинная) сортировка
(defun my_sort (lst)
  (cond ((null lst) Nil)
    (t (nconc
      (my_sort (remove-if-not (lambda (x) (< (car lst) x)) (cdr lst)))
      (remove-if-not (lambda (x) (= (car lst) x)) lst)
      (my_sort (remove-if-not (lambda (x) (> (car lst) x)) (cdr lst)))
      )
    )
  )
)
; сортировка выбором

; в эту функцию не должен попадать пустой список
; (лямбда-функция принимает строго 2 аргумента)
(defun list_min (lst)
  (reduce
    #'(lambda (a b) (cond ((< a b) a) (b)))
    lst
  )
)
(defun my_sort_inner (lst res)
  (cond
    ((null lst) res)
    (t (let*
        (
          (cur_min (list_min lst))
          (lst_rest (remove cur_min lst :count 1))
        )
        (my_sort_inner lst_rest (cons cur_min res))
      )
    )
  )
)

(defun my_sort (lst)
  (my_sort_inner lst ())
)
```

Сначала исходный список сортируется по невозрастанию. Затем рекурсивно отсекается его голова, пока голова не станет меньше верхней границы. Наконец, отсекается его конец, начиная с элемента, который меньше или равен нижней границе, в процессе чего список инвертируется и становится отсортированным по неубыванию. Такой подход позволяет избежать рекурсии, в которой результат собирается на выходе.

```
; в функциях cut_upper и cut_lower предполагается, что переданный список
; отсортирован по невозрастанию
(defun cut_upper (lst b)
  (cond
    ((null lst) Nil)
    ((< (car lst) b) lst)
    (t (cut_upper (cdr lst) b))
  )
)

(defun cut_lower (lst a res)
  (cond
    ((null lst) res)
    ((<= (car lst) a) res)
    (t (cut_lower (cdr lst) a (cons (car lst) res)))
  )
)

(defun select-between-sorted (lst a b)
  (cut_lower (cut_upper (my_sort lst) b) a ())
)

(select-between-sorted '(0 3 7 6 5 4 1) 1 6) => (3 4 5)
```

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

а) одноуровневого смешанного

```
(defun rec-add-inner (lst sum)
  (cond
    ((null lst) sum)
    ((numberp (car lst)) (rec-add-inner (cdr lst) (+ sum (car lst))))
    (t (rec-add-inner (cdr lst) sum))
  )
)

```

```
(defun rec-add (lst)
  (rec-add-inner lst 0)
)

(rec-add '(1 a 2 -1/2 "abc")) => 5/2
```

б) структурированного (то есть элементами могут быть списки)

```
(defun rec-add-inner (lst sum)
  (cond
    ((null lst) sum)
    ((numberp (car lst)) (rec-add-inner (cdr lst) (+ sum (car lst))))
    ((atom (car lst)) (rec-add-inner (cdr lst) sum))
    (t (rec-add-inner (cdr lst) (+ sum (rec-add (car lst)))))
  )
)

(defun rec-add (lst)
  (rec-add-inner lst 0)
)

(rec-add '(1 (1 (1 1)) a 1 ((1 "abc") 1))) => 7
```

(взаимная рекурсия, но по сути – рекурсия более высокого порядка)

7. Написать рекурсивную версию с именем recnth функции nth.

```
(defun recnth (n lst)
  (cond
    ((null lst) Nil)
    ((= n 0) (car lst))
    (t (recnth (- n 1) (cdr lst)))
  )
)

(recnth 0 '()) => Nil
(recnth 2 '(0 1)) => Nil
(recnth 0 '(0 1 2)) => 0
(recnth 2 '(0 (1 1) (2))) => (2)
```

8. Написать рекурсивную функцию allodd, которая возвращает t когда все элементы списка нечетные.

```
; если гарантируется, что все элементы списка – целые числа
(defun allodd (lst)
  (cond
    ((null lst) t)
    ((oddp (car lst)) (allodd (cdr lst)))
  )
)

(allodd '(1 3 5)) => T
(allodd '()) => T
(allodd '(2 3)) => Nil
(allodd '(1 3 4)) => Nil

; если такой гарантии нет (то есть если встречается элемент списка,
; который не является целым нечетным числом, возвращается Nil)
(defun allodd (lst)
  (cond
    ((null lst) t)
    ((and (integerp (car lst)) (oddp (car lst))) (allodd (cdr lst)))
  )
)

(allodd '(1 3 5)) => T
(allodd '(1 3.0)) => Nil
(allodd '(1 a)) => Nil
```

9. Написать рекурсивную функцию, которая возвращает первое нечетное число из списка (структурированного), возможно создавая некоторые вспомогательные функции.

```
; x может быть как атомом, так и структурой
(defun first_odd (x)
  (cond
    ((null x) Nil)
    ((and (integerp x) (oddp x)) x)
    ((atom x) Nil)
    ((or (first_odd (car x)) (first_odd (cdr x))))
  )
)

(first_odd '()) => NIL
```

```
(first_odd '(() 2 1)) => 1
(first_odd '(1.0 1)) => 1
(first_odd '(a 1)) => 1
(first_odd '("abc" 1)) => 1
(first_odd '((2 4) 1)) => 1
(first_odd '((2 . 4) 1)) => 1
(first_odd '(((2 . 1) 4) 3)) => 1
(first_odd '((2 4 (2 1)) 5)) => 1
(first_odd '((2 4 (2)) 6 (1 (3 5)))) => 1
```

10. 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

```
; а) с помощью рекурсии, собирающей результат на выходе
(defun squares (lst)
  (cond
    ((null lst) Nil)
    (t (cons (* (car lst) (car lst)) (squares (cdr lst))))
  )
)

; б) через "хвостовую рекурсию"
(defun squares_inner (lst result)
  (cond
    ((null lst) (reverse result))
    (t (squares_inner (cdr lst) (cons (* (car lst) (car lst)) result)))
  )
)

(defun squares (lst)
  (squares_inner lst ())
)

(squares '(0 10 -10 5.5 2/3)) => (0 100 100 30.25 4/9)
```