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

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 10

Дисциплина	Функциональное и логическое программирование.
Тема	Рекурсивные функции
Студент	Степанов А. О.
Группа	ИУ7-63Б
Оценка (баллы)	
Преподаватель	Толпинская Н.Б.

Москва, 2020 г.

ЗАДАНИЕ 1

Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list.

```
(defun sum_length (lst)
    (cond
        ((null lst) 0)
        (T (+ (length (car lst)) (sum_length (cdr lst))))
    )
)

(sum_length '((1 2 3) (1 2) (1) (1 2 3 4) (1 2))) ;;; 12
```

ЗАДАНИЕ 2

Написать рекурсивную версию (с именем reg-add) вычисления суммы чисел заданного списка.

```
(defun reg-add (lst)
    (cond
        ((null lst) 0)
        (T (+ (car lst) (reg-add (cdr lst))))
    )
)

(reg-add '(2 4 6)) ;;; 12
```

ЗАДАНИЕ 3

Написать рекурсивную версию с именем recnth функции nth.

```
(defun recnth (n lst)
    (cond
        ((null lst) Nil)
        ((eql n 0) (car lst))
        (T (recnth (- n 1) (cdr lst)))
    )
)

(recnth 0 '(1 2 3)) ;;; 1
(recnth 1 '(1 2 3)) ;;; 2
(recnth 2 '(1 2 3)) ;;; 3
(recnth 3 '(1 2 3)) ;;; Nil
```

ЗАДАНИЕ 4

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

```
(defun alloddr (lst)
    (cond
        ((null lst) T)
        ((eql (mod (car lst) 2) 0) Nil)
        (T (alloddr (cdr lst)))
    )
)

(alloddr '(1 2 3 4)) ;;; Nil
(alloddr '(1 3 5 7 9 10)) ;;; Nil
(alloddr '(1 3 5 7 9)) ;;; T
```

ЗАДАНИЕ 5

Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка-аргумента.

```
(defun last_rec (lst)
    (cond
        ((null lst) Nil)
        ((eql (length lst) 1) (car lst))
        (T (last_rec (cdr lst)))
    )
)

(last_rec ()) ;;; Nil
(last_rec '(1 2 3 4)) ;;; 4
```

ЗАДАНИЕ 6

Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-ого аргумента функции.

Вариант:

1. от n-аргумента функции до последнего >= 0,
2. от n-аргумента функции до m-аргумента с шагом d.

```
(defun sum_recursive (a b d n lst)
    (cond
        ((null lst) 0)
        ((< b 0) 0)
        ((and (<= a 0) (eql n 0))
            (+
                (car lst)
                (sum_recursive a (- b 1) d (- d 1) (cdr lst)))
        )
        ((<= a 0) (sum_recursive a (- b 1) d (- n 1) (cdr lst)))
        (T (sum_recursive (- a 1) (- b 1) d 0 (cdr lst)))
    )
)

(defun sum_elements (a b d lst)
    (cond
        ((or (> a b) (<= d 0)) 0)
        (T (sum_recursive a b d 0 lst))
    )
)

(sum_elements 5 3 1 '(1 2 3 4 5 6 7 8 9 10)) ;;; 0
(sum_elements 0 9 1 '(1 2 3 4 5 6 7 8 9 10)) ;;; 1+2+3+4+5+6+7+8+9+10 = 55
(sum_elements 2 8 1 '(1 2 3 4 5 6 7 8 9 10)) ;;; 3+4+5+6+7+8+9 = 42
(sum_elements 2 5 2 '(1 2 3 4 5 6 7 8 9 10)) ;;; 3+5 = 8
(sum_elements 2 8 3 '(1 2 3 4 5 6 7 8 9 10)) ;;; 3+6+9 = 18
```

ЗАДАНИЕ 7

Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

```
(defun last_odd (lst)
    (cond
        ((null lst) Nil)
        (T
            (let
                ((next_iteration (last_odd (cdr lst))))
                (cond
                    (
                        (and
                            (eql (mod (car lst) 2) 1)
                            (not next_iteration)
```

```
12                          ) (car lst)
13                      )
14                      (T next_iteration)
15                  )
16              )
17          )
18      )
19  )
20
21  (last_odd '(1 2 3 4 5 6)) ;;; 5
22  (last_odd '(1 2 3 4 5)) ;;; 5
23  (last_odd '(2 4 6 8)) ;;; Nil
```

ЗАДАНИЕ 8

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.

```
1  (defun square_list (lst)
2      (cond
3          ((null lst) Nil)
4          (T (cons (* (car lst) (car lst)) (square_list (cdr lst))))
5      )
6  )
7
8  (square_list '(1 2 3 4 5)) ;;; (1 4 9 16 25)
```

ЗАДАНИЕ 9

Написать функцию с именем select-odd, которая из заданного списка выбирает все нечетные числа. (Вариант 1: select-even, вариант 2: вычисляет сумму всех нечетных чисел(sum-all-odd) или сумму всех четных чисел (sum-all-even) из заданного списка.)

```
1  (defun select_odd_or_even (lst n)
2      (cond
3          ((null lst) Nil)
4          ((eql (mod (car lst) 2) n)
5              (append (list (car lst)) (select_odd_or_even (cdr lst) n))
6          )
7          (T (select_odd_or_even (cdr lst) n))
8      )
```

```
)

(defun select-odd (lst) (select_odd_or_even lst 1))
(defun select-even (lst) (select_odd_or_even lst 0))

(select-odd '(1 2 3 4 5 6 7 8 9 10)) ;;; (1 3 5 7 9)
(select-even '(1 2 3 4 5 6 7 8 9 10)) ;;; (2 4 6 8 10)

(defun sum_odd_or_even (lst n)
    (cond
        ((null lst) 0)
        ((eql (mod (car lst) 2) n)
            (+ (car lst) (sum_odd_or_even (cdr lst) n))
        )
        (T (sum_odd_or_even (cdr lst) n))
    )
)

(defun sum-odd (lst) (sum_odd_or_even lst 1))
(defun sum-even (lst) (sum_odd_or_even lst 0))

(sum-odd '(1 2 3 4 5 6 7 8 9 10)) ;;; 25
(sum-even '(1 2 3 4 5 6 7 8 9 10)) ;;; 30
```

ТЕОРЕТИЧЕСКИЕ ВОПРОСЫ

Способы организации повторных вычислений в Lisp

— использование функционалов

— использование рекурсии

Что такое рекурсия? Классификация рекурсивных функций в Lisp

Рекурсия – это ссылка на определяемый объект во время его определения. Т. к. в Lisp используются рекурсивно определенные структуры (списки), то рекурсия – это естественный принцип обработки таких структур.

— простая рекурсия - один рекурсивный вызов в теле

— рекурсия первого порядка - рекурсивный вызов встречается несколько раз

— взаимная рекурсия - используется несколько функций, рекурсивно вызывающих друг друга.

Различные способы организации рекурсивных функций и порядок их реализации

— Хвостовая рекурсия. В целях повышения эффективности рекурсивных функций рекомендуется формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии. Это и есть хвостовая рекурсия.

— Возможна рекурсия по нескольким параметрам

— Дополняемая рекурсия – при обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова , а вне его

— Выделяют группу функций множественной рекурсии. На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи.

Способы повышения эффективности реализации рекурсии

— Использование хвостовой рекурсии. Если условий выхода несколько, то надо думать о порядке их следования.

— Превращение не хвостовой рекурсии в хвостовую. Для превращения не хвостовой рекурсии в хвостовую и в целях формирования результата (результирующего списка) на входе в рекурсию, рекомендуется использовать дополнительные (рабочие) параметры. При этом становится необходимым создат фунецию – оболочку для реализации очевидного обращения к функции.