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»

Лабораторная работа № __10__

Дисциплина Функциональные и логические языки программирования

Студент Фирсова Дарья

Группа ИУ7-62

Оценка (баллы) _____

Преподаватель _____

Москва.
2020 г.

1. Способы организации повторных вычислений в Lisp
   Рекурсия и использование функционалов

2. Различные способы использования функционалов
   1. применяющие
   2. отображающие функционалы
   3. функционалы, являющиеся предикатами
   4. функционалы, использующие предикаты в качестве функционального объекта.

3. Что такое рекурсия? Способы организации рекурсивных функций

Рекурсия — это ссылка на определяемый объект во время его определения.
простая рекурсия - один рекурсивный вызов в теле
рекурсия первого порядка - рекурсивный вызов встречается несколько раз
взаимная рекурсия - используется несколько функций, рекурсивно вызывающих друг друга.

4. Способы повышения эффективности реализации рекурсии.
Использование хвостовой рекурсии, причем условие выхода в условном выражении должно стоять первым.

7. Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list, т.е. например для аргумента ((1 2) (3 4)) -> 4.

```
(defun all_length(lst);
        (reduce #'+
                        (mapcar         (lambda         (x)
                                                (if (listp x) (all_length x) 1)
                                        )
                                        lst
                        )
        )
)
```

Входные данные - список

8. Написать рекурсивную версию (с именем reg-add) вычисления суммы чисел заданного списка.
Например: (reg-add (2 4 6)) -> 12

```
(defun reg_add (lst)
        (cond
                ((null lst) 0)
                (t (+ (car lst) (reg_add (cdr lst) )))
        )
)
```

(reg_add '(1 2 3)) -> 6

Входные данные - список

9. Написать рекурсивную версию с именем recnth функции nth.

```
(defun rec_nth (n lst rec)
        (cond
                ((null lst) nil)
                ((equal rec n) (car lst))
                (t (rec_nth n (cdr lst) (+ rec 1)))
        )
)
(defun recnth (n lst)
        (rec_nth n lst 0)
)
```

(rec_nth '(1 2 3) 2) -> 3

Входные данные - список и номер искомого аргумента

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

```
(defun alloddr (lst)
        (cond
                ((null lst) t)
                (t (and (oddp (car lst)) (allodr (cdr lst))))
        )
)
```

(alloddr '(1 3)) -> T

Входные данные - список

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

```
(defun argum(lst)
 (cond
  ((null (cdr lst)) (car lst))
  (t (argum (cdr lst)))
 )
)
```

(argum '(1 2 3)) -> 3

Входные данные - список

12. 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 12_summ_n (n lst current)
 (cond
          ((equal n current) 0)
          (t (+ (car lst) (12_summ_n n (cdr lst) (+ current 1))))
 )
)

(defun start_12 (n lst)
 (cond
        ((>=(length lst) n) (12_summ_n n lst 0))
 )
)
```

(start_12 '3 '(1 2 3 4)) -> 6

Входные данные - номер элемента и список

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

```
(defun last_odd(curr val)
	(cond
		((eq curr Nil) val)
		((oddp (car curr)) (last_odd (cdr curr) (car curr)))
		(t (last_odd (cdr curr) val))
	)
)

(defun last_odd_start(lst)
	(last_odd lst Nil)
)
```

(last_odd_start '(1 2 3 4 5)) -> 5
Входные данные - список

14. 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 14_pow_2 (lst)
	(cond
		((null lst) nil)
		(t (cons
			(*
				(car lst)
				(car lst))
			(14_pow_2 (cdr lst)))
		)
	)
)
```

(pow_2 '(1 2 3)) -> (1 4 9)
Входные данные - список

15. 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.)

```
(defun 15_select_odd (lst)
	(remove nil (cond
		((null lst) nil)
		(t (cons(cond ((oddp(car lst)) (car lst))) (select_odd (cdr lst)) ))
		)
	)
)
```

(sum_select_odd '(1 2 3 5)) -> 9

Входные данные - список