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 № _9_

Дисциплина Функциональные и логические языки программирования

Студент Козырев Марк

Группа ИУ7-63

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.

2. Написать предикат set-equal, который возвращает t, если два его множество-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

```
(defun set-equal (set1 set2)
        (and (subsetp set1 set2)  (subsetp set2 set1)))
```

set-equal '(1 2 3) '(3 2 1))
T

3. Напишите необходимые функции, которые обрабатывают таблицу из точечных пар: (страна. столица), и возвращают по стране - столицу, а по столице - страну.

Функционалы:

```
(setf country_city ( list ( cons 'russia 'moscow) (cons 'england 'london) (cons 'germany 'berlin)))

(defun look_in_table (table value)
        (remove nil
                (mapcar
                        #'(lambda (table)
                                (cond
                                        ((equalp (cdr table) value)(car table))
                                        ((equalp (car table) value)(cdr table))
                                )
                        )
                        table
                )
        )
)
```

(search_cc country_city 'moscow) -> (RUSSIA)
(search_cc country_city 'london) -> (ENGLAND)

Рекурсия:

```
(defun look_in_table_rec (table value)
        (cond
                ((null table) nil)
                ((equalp (cdar table) value) (caar table) )
                ((equalp (caar table) value) (cdar table) )
                (t (look_in_table_rekurs (cdr table) value) )
        )
)
```

(search_cc country_city 'moscow) -> MOSCOW

Входные данные - список точечных пар

7. Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда все элементы списка --- числа

```
(defun mul(lst val)
        (mapcar #'(lambda (x) (* x val))
                        lst
        )
)
```
(mul '(1 2 3) 4) - >(4 8 12)

Входные данные - список и заданное число

2. Напишите функцию, которая уменьшает на 10 все числа из списка аргумента этой функции.
Функционал:

```
(defun minus_ten (lst)
        (mapcar
                #'(lambda (value)
                        (- value 10)
                )
                lst
        )
)
```

(minus_ten '(1 2 3)) -> (-9 -8 -7)
Входные данные - список

Рекурсия:

```
(defun minus_ten_rec (lst)
       (cond
              ((null lst) nil)
              (t (append (list(- (car lst) 10)) (minus_ten_rekurs (cdr lst))))
       )
)
```

3. Написать функцию, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком.

```
(defun first_list(lst)
        (cond
                        ((null lst) nil)
                        ((listp (car lst)) (car lst))
                        (t (first_list (cdr lst)))
        )
)
```
(first_list '(1 (2 3) 4)) -> (2 3)
Входные данные - список с вложенным списком

4. Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10.

Функционал:

```
(defun more_less (lst value1 value2)
	(remove nil (mapcar #'(lambda (value)
		(cond
			((and
				(> value value1)
				(< value value2)
			(eval value)))))
			lst
		)
	)
)
```

(more_less '(1 2 3 32 -2) 0 3) -> (1 2)

Входные данные - список, меньшее значение, большее значение

Рекурсия:

```
(defun more_less_rec (lst value1 value2)
	(remove nil
		(cond
			((null lst)
				nil)
			(t
				(append
					(list
						(and
							(> (car lst) value1)
							(< (car lst) value2)
						(eval (car lst))))
			(more_less_rekurs (cdr lst) value1 value2)))
		)
	)
)
```

(more_less_rec '(1 2 3 32 -2) 0 3) -> (1 2)

5. Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов. (Напомним, что A x B это множество всевозможных пар (a b), где a принадлежит A, принадлежит B.)

```
(defun decart(X Y)
	(mapcan #'
		(lambda (x)
			(mapcar #'
				(lambda (y) (list x y))
				Y
			)
		)
		X
```

```
          )
)
```
(decart '(1 2 3) '(4 5 6)) -> ((1 4) (1 5) (1 6) (2 4) (2 5) (2 6) (3 4) (3 5) (3 6))

```
(defun dekart(lst1 lst2)
 (cond
  ((null lst2)
   nil)
  ((not (null lst2))
   (append
    (list (car lst1))
    (list (car lst2))
    (dekart lst1 (cdr lst2))))
  )
)

(defun start (lst1 lst2)
 (cond
  ((null lst1)
   nil)

  (t (append
   (dekart lst1 lst2)
   (start (cdr lst1) lst2)))

 )
)
 (start '(1 2 3) '(4 5 6)) ->  (1 4 1 5 1 6 2 4 2 5 2 6 3 4 3 5 3 6)
```

6. Почему так реализовано reduce, в чем причина?
(reduce #*+0) -> 0
(reduce #*+ ()) -> 0

reduce использует результат предыдущего, поэтому необходимо начальное значение (initial_value). Очевидно, что начально значение для сложения 0.