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»

Отчёт
к лабораторной работе № 8
По курсу: «Функциональное и логическое программирование»
Тема: «Работа интерпретатора Lisp.»

Студент Прохорова Л. А.

Группа ИУ7-63Б

Оценка (баллы) _____

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

Москва.
2021 г.

Цель работы: приобрести навыки работы с управляющими структурами Lisp.
Задачи работы: изучить работу применяющих и отображающих функционалов.

Задание 7
Напишите функцию, которая умножает на заданное число-аргумент все числа
из заданного списка-аргумента, когда
а) все элементы списка --- числа
б) все элементы списка --- любые объекты

а) все элементы списка --- числа

С использованием функционалов

```
(defun mullall(k lst)
	(mapcar #'(lambda (x) (* k x)) lst)
)
```

Тестирование
(mullall 3 '(1 2 3)) -> (3 6 9)

С использование рекурсии

```
;cons

(defun mullall-r(k lst)
	(cond   ((null lst) nil)
		(t (cons (* k (car lst)) (mullall-r k (cdr lst))))
	)
)
```

Тестирование
(mullall-r 3 '(1 2 3)) -> (3 6 9)

Второй вариант рекурсии

```
;append

(defun mullall-r2(k lst rlst)
	(cond   ((null lst) rlst)
		(t (mullall-r2 k (cdr lst) (append rlst (cons (* k (car lst)) nil))))
	)
)
```

Тестирование
(mullall-r2 3 '(1 2 3) nil) -> (3 6 9)

(mullall-r2 3 nil nil) -> NIL

б) все элементы списка --- любые объекты

**numberp - возвращает истину, если объект относится к типу number ; в противном случае возвращает false*
**listp -принимает один аргумент и возвращает t, если аргумент оценивается как список, в противном случае он возвращает nil*

С использованием функционалов

```
(defun mullall-d(k lst)
	(mapcar
		#'(lambda (x)
			(cond ((numberp x)(* k x))
				((listp x) (mullall-d k x))
				(t x)
			)
		 ) lst)
)
```

Тестирование
(mullall-d 3 '(1 2 a b (d f 2 3))) -> (3 6 A B (D F 6 9))
(mullall-d 3 '(1 2 3)) -> (3 6 9)
(mullall-d 3 '(1 (1 2) (2 (3 (4))))) -> (3 (3 6) (6 (9 (12))))

С использованием рекурсии

```
(defun mullall-dr(k lst)
	(cond ((null lst) nil)
		((numberp (car lst)) (cons (* k (car lst)) (mullall-dr k (cdr lst))))
		((listp (car lst)) (cons (mullall-dr k (car lst)) (mullall-dr k (cdr lst))))
		(t (cons (car lst) (mullall-dr k (cdr lst))))
	)
)
```

Тестирование
(mullall-dr 3 '(1 2 a)) -> (3 6 A)
(mullall-dr 3 '(1 2 (3 e (r) (4 (5)))))->(3 6 (9 E (R) (12 (15))))

Задание 8.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)).

```
(defun is_between (left right a)
   (if (> a left) (< a right) (> a right))
)

(defun make_select_between (lst l r)
   (reduce
      #'(lambda (res el) (if (is_between l r el) (append res (cons el Nil)) res))
      lst :initial-value nil
   )
)

(defun insert_elem (elem list)
   (cond
      ((null list) (cons elem nil))
      ((< elem (car list)) (cons elem list))
      (t  (cons (car list) (insert_elem elem (cdr list))))
   )
)

(defun my_sort (list)
   (reduce
      #'(lambda (sorted tmp)
         (insert_elem tmp sorted)
      ) list :initial-value nil
   )
)

(defun sorted_select_between(left right list)
   (my_sort (make_select_between list left right))
)
```

Тестирование
(is_between 2 4 3)->T
(is_between 4 2 3)->T
(is_between 4 2 5)->NIL

(make_select_between '(2 3 1 5 4 0 3) 2 6)->(3 5 4 3)

(insert_elem 3 '(1 2 3))->(1 2 3 3)
(insert_elem 3 '(4 5 6))->(3 4 5 6)
(insert_elem 3 '(1 2 4 5))->(1 2 3 4 5)

(my_sort '(1 2 3 4))->(1 2 3 4)
(my_sort '(4 3 2 1))->(1 2 3 4)
(my_sort '(3 4 1 2))->(1 2 3 4)

(sorted_select_between 0 4 '(1 2 4 2 8 5 4 9 0 3 1 2 4 3 5))->(1 1 2 2 2 3 3)
(sorted_select_between 0 0 '(1 2 3))->NIL

Задание 1. Что будет результатом (mapcar 'вектор '(570-40-8))?
The function COMMON-LISP-USER::ВЕКТОР is undefined.
Вектор воспринимается как функция, переданная в параметры mapcar, и применяется последовательно к каждому из значений по car-указателям списковых ячеек. Но такая функция не была определена.

Задание 2. Напишите функцию, которая уменьшает на 10 все числа из списка аргумента этой функции.
С использованием функционалов

```
(defun minus_ten(lst)
   (mapcar
      #'(lambda (x)
         (cond
            ((numberp x) (- x 10))
            ((listp x) (minus_ten x))
            (t x)
         )
      ) lst
   )
)
```

Тестирование
(minus_ten '(10 (1 2 (11 12) 13 3) 10))->(0 (-9 -8 (1 2) 3 -7) 0)

С использованием рекурсии

```
(defun minus_ten_r (lst)
   (cond
      ((null lst) nil)
      ((numberp (car lst)) (cons (- (car lst) 10) (minus_ten_r (cdr lst)) ))
      ((listp (car lst)) (cons (minus_ten_r (car lst)) (minus_ten_r (cdr lst)) ))
      (t (cons (car lst) (minus_ten_r (cdr lst)) ))
   )
)
```

Тестирование
(minus_ten_r '(10 (1 2 (11 12) 13 3) 10))->(0 (-9 -8 (1 2) 3 -7) 0)

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

С использованием рекурсии

```
(defun first_listarg_el (lst)
   (cond
      ((null lst) nil)
      ((and (listp (car lst)) (not (null (car lst)))) (caar lst))
      (t (first_listarg_el (cdr lst)))
   )
)
```

Тестирование
(first_listarg_el '(1 () 3 (2 3 4) 4))->2

С использованием функционалов

```
(defun first_listarg (lst)
   (car ( find-if (lambda (x) (and (listp x) (not (null x)))) lst
   ))
)
```

Тестирование
(first_listarg '(1 () 3 (2 3 4) 4))->2

Сумма элементов смешанного структурированного списка

С использованием рекурсии

```
(defun my_sum_r(lst res)
   (cond
      ((null lst) res)
      ((numberp (car lst))  (my_sum_r (cdr lst)(+ res (car lst))))
      ((listp (car lst)) (my_sum_r (cdr lst) (my_sum_r (car lst) res)))
      (t (my_sum_r (cdr lst) res))
   )
)
```

Тестирование
(my_sum_r '(1 q 2) 0)->3
(my_sum_r '(1 2 3) 0)->6
(my_sum_r '(1 2 (3 (4 3) (2)) (2) (1 (2))) 0)->20

С использованием функционалов

```
(defun my_sum (lst)
   (reduce #'(lambda (res x)
      (cond
         ((numberp x) (+ res x))
         ((listp x) (+ (my_sum x) res))
         (t res)
      )) lst :initial-value 0)
)
```

Тестирование

(my_sum '(1 q 2 (3)))->6

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

(my_sum '(1 a (2 3)))->5

Ответы на теоретические вопросы:
1.Порядок работы и варианты

Функционалы – функции, принимающие в качестве аргумента другую функцию (имя или лямбда-выражение).
Функционалы классифицируются на:
1. Применяющие.
a. (apply #’func (arg_lst)) – вызывает функцию func и передает список arg_lst в качестве списка фактических параметрова функции func
b. (funcall #’func arg1…argn) – применяет функцию func к аргументам arg1…argn
2. Отображающие.
a. mapcar принимает на вход имя функции или lambda-выражение и переменное количество списков-аргументов. Если передан только один список-аргумент: функция, переданная в параметры mapcar, применяется последовательно к каждому из значений по car-указателям списковых ячеек. Из вычисленных значений формируется список с помощью list. Если передано несколько списков-аргументов: функция, переданная в аргументы mapcar, должна иметь столько же формальных параметров, сколько было передано списков-аргументов. Функция последовательно применяется к первым элементам всех списков-аргументов, затем ко вторым и т.д. Если списки-аргументы имеют разную длину, mapcar вычисляет элементы результирующего списка, пока не закончатся элементы самого короткого из списков-аргументов.
b. mapcan работает аналогичным образом, однако результирующий список формируется не с помощью list, а с помощью nconc.

c. maplist принимает имя функции или лямбда-выражение и ровно один список-аргумент. Применяет переданную функцию ко всему списку-аргументу, а затем - последовательно к каждому последующему хвосту, переходя по cdr-указателям. Формирует результирующий список из вычисленных значений с помощью cons.
d. Функционал mapcon работает аналогично, но формирует результирующий список с помощью nconc.
e. some проверяет удовлетворение условию хотя бы одного элемента, вычисляет до первого удовлетворяющего. Пример: (some (lambda (x) (> x 0)) '(3 -3 var -2 2)) => T
var - переменная, значение которой не было ранее определено, тем не менее ошибки не будет, тк не будет попытки вычислить ее значение
f. every проверяет удовлетворение условию всех элементов списка, вычисляет значения предиката до первого не удовлетворяющего элемента списка. Пример: (every (lambda (x) (> x 0)) '(3 -3 var -2 2))=> Nil
g. find-if последовательно вычисляет предикат от каждого из значений до первого удовлетворяющего условию элемента, возвращает значение первого удв элемента (если он найден) и Nil (если удв элементов нет)
Примеры
* (find-if (lambda (x) (> x 0)) '(-3 3 -2 2)) => 3
* (find-if (lambda (x) (> x 10)) '(-3 3 -2 2)) => NIL
h. remove-if из исходного списка формирует новый список, из которого исключены *все* элемент удв предикату. Исходный список не изменяется при этом. Пример: (remove-if (lambda (x) (> x 5)) ‘(1 5 6 7)) => (1 5)
i. remove-if-not работает аналогичным образом, но исключает элементы не удовлетворяющие предикату. Пример: (remove-if (lambda (x) (> x 5)) ‘(1 5 6 7)) => (6 7)
j. reduce принимает функцию с двумя формальными параметрами и ровно один список-аргумент. Применяет функцию к первым двум элементам списка, затем - последовательно применяет функцию к получаемому на каждом этапе результату и следующему элементу списка-аргумента. Пример: (reduce (lambda (x y) (cons y x)) ‘(1 2 3) :initial-value Nil) => (3 2 1)