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

к лабораторной работе №8

По курсу: «Функциональное и логическое программирование»

Тема: «Использование функционалов».

Студент: Якуба Д.В.
Группа: ИУ7-63Б
Преподаватели: Толпинская Н. Б.,
Строганов Ю. В.

Москва, 2021 г.

Практическая часть

Задание 1. Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда:

а) Все элементы списка – числа,

б) Элементы списка – любые объекты.

Решение:

```
(defun mult-els (lst num)
    (mapcar #'(lambda (arg) (* arg num)) lst))
```

```
(mult-els '(1 3 4) 3) -> (3 9 12);
(mult-els '(1 333) -2) -> (-2 -666);
(mult-els '(1 333) 0) -> (0 0);
```

```
; С использованием функционала для одномерного смешанного списка
(defun mult-els (lst num)
    (mapcar #'(lambda (arg) (cond
                                ((numberp arg) (* arg num))
                                (t arg)))
                                lst))

; С использованием функционала для структурированного смешанного списка
(defun mult-els-deep (lst num)
    (mapcar #'(lambda (arg)
                        (cond
                              ((listp arg) (mult-els-deep arg num))
                              ((numberp arg) (* arg num))
                              (t arg))) lst))

; Рекурсивно для одномерного смешанного списка
(defun mult-els-rec (lst num)
    (cond
        ((null lst) nil)
        ((numberp (car lst)) (cons (* (car lst) num) (mult-els-
rec (cdr lst) num)))
        (t (cons (car lst) (mult-els-rec (cdr lst) num)))))

; Рекурсивно для структурированного смешанного  списка
(defun mult-els-rec-deep (lst num)
    (cond
        ((null lst) nil)
        ((listp (car lst)) (cons (mult-els-rec-deep (car lst) num) (mult-els-
rec-deep (cdr lst) num)))
        ((numberp (car lst)) (cons (* (car lst) num) (mult-els-rec-
deep (cdr lst) num)))
        (t (cons (car lst) (mult-els-rec-deep (cdr lst) num)))))
```

```
(mult-els '(a 1 b 3 c 4 d) 3) -> (A 3 B 9 C 12 D);
(mult-els '(1 a 333) -2) -> (-2 A -666);
(mult-els '(1 333) 0) -> (0 0);
(mult-els-deep '(1 a 333) 2) -> (2 A 666);
```

```
(mult-els-deep '(1 (((5))) 333 (3 2)) 2) -> (2 (((10))) 666 (6 4));
(mult-els-deep '(1 (a 5) (((5) a (7))) b 333 (3 2)) 2) -> (2 (A 10) (((10) A (14)))
B 666 (6 4));
(mult-els-rec '(1 3 4) 3) -> (3 9 12);
(mult-els-rec '(1 333) -2) -> (-2 -666);
(mult-els-rec '(1 333) 0) -> (0 0);
(mult-els-rec-deep '(1 a 333) 2) -> (2 A 666);
(mult-els-rec-deep '(1 (((5))) 333 (3 2)) 2) -> (2 (((10))) 666 (6 4));
(mult-els-rec-deep '(1 (a 5) (((5) a (7))) b 333 (3 2)) 2) -> (2 (A 10) (((10) A
(14))) B 666 (6 4));
```

Задание 2. 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)).

Решение:

```
; С использованием рекурсии для списка, содержащего только числа
(defun select-rec (cur-lst down-limit up-limit)
    (cond
        ((null cur-lst) nil)
        ((and
            (<= (car cur-lst) up-limit)
            (>= (car cur-lst) down-limit))
                (cons (car cur-lst) (select-rec (cdr cur-lst) down-limit up-
limit)))
        (t (select-rec (cdr cur-lst) down-limit up-limit))))

; Для структурированных смешанных списков
(defun select-rec (cur-lst down-limit up-limit)
    (cond
        ((null cur-lst) nil)
        ((listp (car cur-lst)) (cons (select-rec (car cur-lst) down-limit up-
limit) (select-rec (cdr cur-lst) down-limit up-limit)))
        ((and
            (numberp (car cur-lst))
            (<= (car cur-lst) up-limit)
            (>= (car cur-lst) down-limit))
                (cons (car cur-lst)
                                (select-rec (cdr cur-lst) down-limit up-
limit)))
        (t (select-rec (cdr cur-lst) down-limit up-limit))))

; обёрточная функция
(defun select-between (lst fNum sNum)
    (let ((down-limit (cond ((< fNum sNum) fNum) (t sNum)))
          (up-limit (cond ((>= fNum sNum) fNum) (t sNum))))

          (select-rec lst down-limit up-limit)))

; С использованием функционалов для списка, содержащего только числа
(defun select-between-fun (lst fNum sNum)
    (cond
        ((< fNum sNum) (remove-if-
not (lambda (el) (and (>= el fNum) (<= el sNum))) lst))
        (t (remove-if-not (lambda (el) (and (<= el fNum) (>= el sNum))) lst))))
```

```
; С использованием функционалов для структурированного смешанного списка
(defun select-between-fun-deep (lst fNum sNum)
    (mapcan #'(lambda (el)
                        (cond
                           ((listp el) (cons (select-between-fun-
deep el fNum sNum) nil))
                           (t (and (numberp el) (>= el fNum) (<= el sNum) (cons
 el nil))))) lst))
```

```
(select-between '(1 2 3 2 4 5 1 2) 2 4) -> (2 3 2 4 2);
(select-between '(1 2 3 2 4 5 1 2) -3 0) -> nil;
(select-between '(-2 2 7 1.5 4 5.2 1 2) -3 5) -> (-2 2 1.5 4 1 2);
(select-between-fun-deep '((1 2 3) 4 5 a 6 4 ((3)) a) 2 4) -> ((2 3) 4 4 ((3)));
(select-between-fun-deep '(1 3 a 2) 2 4) -> (3 2);
(select-between-fun-deep '(1 1 a 1) 2 4) -> NIL;
```

Задание 3. Что будет результатом (mapcar ‘вектор ‘(570-40-8))?

Ответ:

Результатом выполнения будет ошибка: «name ВЕКТОР is undefined». Для исправления ситуации потребуется заменить «’вектор» на «’vector».

(mapcar ‘vector ‘(570-40-8)) -> (#(|570-40-8|)).

(#(|570-40-8|)) – это список, включающий в себя вектор фиксированной длины 1, состоящий из элемента «570-40-8». Синтаксис #(...) – способ записи векторов. При этом | | - это способ записи строк, начинающихся с цифры.

Задание 4. Напишите функцию, которая уменьшает на 10 все числа из списка-аргумента этой функции.

Решение:

```
; Рекурсивно для структурированного смешанного списка
(defun reduce-numbers-by-10-rec-deep (lst)
    (cond
        ((null lst) nil)
        ((listp (car lst)) (cons (reduce-numbers-by-10-rec-
deep (car lst)) (reduce-numbers-by-10-rec-deep (cdr lst))))
        ((numberp (car lst)) (cons (- (car lst) 10) (reduce-numbers-by-10-rec-
deep (cdr lst))))
        (t (cons (car lst) (reduce-numbers-by-10-rec-deep (cdr lst))))))

; С использованием функционала для структурированного смешанного списка
(defun my-mc-function (element)
    (cond
        ((numberp element) (- element 10))
        ((listp element) (reduce-numbers-by-10-deep element))
        (t element)))

(defun reduce-numbers-by-10-deep (lst)
    (mapcar #'my-mc-function lst))
```

```
(reduce-numbers-by-10-rec-deep '(1 2 3 (10 33) -2)) -> (-9 -8 -7 (0 23) -12);
```

```
(reduce-numbers-by-10-rec-deep '(1 help 2 me 3 (10 33) -2)) -> (-9 HELP -8 ME -7
(0 23) -12);
(reduce-numbers-by-10-rec-deep '(когда-нибудь это закончится 7.5 333)) -> (КОГДА-
НИБУДЬ ЭТО ЗАКОНЧИТСЯ -2.5 323);
(reduce-numbers-by-10-deep '(1 2 3 (10 33) -2)) -> (-9 -8 -7 (0 23) -12);
(reduce-numbers-by-10-deep '(1 help 2 me 3 (10 33) -2)) -> (-9 HELP -8 ME -7 (0
23) -12);
(reduce-numbers-by-10-deep '(когда-нибудь это закончится 7.5 333)) -> (КОГДА-
НИБУДЬ ЭТО ЗАКОНЧИТСЯ -2.5 323);
```

Задание 5. Написать функцию, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком.

```
; Рекурсивно
(defun ret-first-lst (lst)
    (cond ((null lst) lst)
          ((and (listp (car lst)) (not (null (car lst)))) (car lst))
          (t (ret-first-lst (cdr lst)))))

; С использованием функционала
(defun ret-first-lst-fun (lst)
    (find-if #'(lambda (x) (and (listp x) (not (null x)))) lst))
```

```
(ret-first-lst '(1 2 3)) -> nil;
(ret-first-lst '(1 ((())) 2 3)) -> ((NIL));
(ret-first-lst '(1 (3982 222) 2 (387631) 3)) -> (3982 222);
(ret-first-lst '(() 1 () (3982 222) 2 (387631) 3)) -> (3982 222);
```

Задание 6. Найти сумму числовых элементов смешанного структурированного списка.

```
; для работы со структурированным смешанным списком рекурсивно
(defun sum-all-nums-rec-deep (lst)
    (cond
        ((null lst) 0)
        ((numberp (car lst)) (+ (car lst) (sum-all-nums-rec-deep (cdr lst))))
        ((listp (car lst)) (+ (sum-all-nums-rec-deep (car lst)) (sum-all-nums-
rec-deep (cdr lst))))
        (t (sum-all-nums-rec-deep (cdr lst)))))

;для работы со смешанным структурированным списком с использованием функционала
(defun sum-all-nums-deep (lst)
    (reduce (lambda (accum cur-element)
                    (cond ((numberp cur-element) (+ accum cur-element))
                          ((listp cur-element) (+ accum (sum-all-nums-
deep cur-element)))
                          (t accum)))
            (cons 0 lst)))
```

```
(sum-all-nums-rec-deep '(1 2 (n) -9 4 7)) -> 5;
(sum-all-nums-rec-deep '(a 1 2 -9 4 (3 n 2) 7)) -> 10;
(sum-all-nums-rec-deep '(1 (((300 a))) 2 -9 4 (3 2) 7)) -> 310;
(sum-all-nums-deep '(1 2 (n) -9 4 7)) -> 5;
(sum-all-nums-deep '(a 1 2 -9 4 (3 n 2) 7)) -> 10;
(sum-all-nums-deep '(1 (((300 a))) 2 -9 4 (3 2) 7)) -> 310;
```

Теоретическая часть

1. Порядок работы и варианты использования функционалов.

Функции высших порядков – функционалы – используются для построения синтаксически-управляемых программ, в качестве одного из аргументов принимают описание функции.

Существует следующая классификация функционалов:

1) Применяющие: (apply #'fun arg-lst), (funcall #'fun arg1 arg2 … argN). Подобные функционалы позволяют применить переданную функцию к списку аргументов.

2) Отображащие: mapcar, mapcan, maplist, mapcon, find-if, remove-if, remove-if-not, reduce, every, some. Данные функции позволяют организовывать повторяющиеся вычисления. Функции find-if, remove-if, remove-if-not, every, some в качестве функции-аргумента принимают некоторый предикат. Функции mapcar, mapcan, maplist, mapcon и reduce принимают функции для работы с элементами обрабатываемых списков.

mapcar применяется ко всем первым элементам списков-аргументов до тех пор, пока не будет окончена работа с самым коротким переданным списком: (mapcar #'fun arg1 arg2 … argN).

maplist на каждой итерации для работы берёт хвост переданного ей списка (хвост хвоста списка и так далее, пока список не станет пустым): (maplist #'fun lst).

mapcan работает так же, как и mapcar, с той лишь разницей, что для формирования результата работы функции используется nconc.

mapcon работает так же, как и maplist, с той лишь разницей, что для формирования результата работы функции используется nconc.

reduce позволяет аккумулировать результат вычислений при обработке каждого элемента списка. Подразумевается, что передаваемая функция принимает два аргумента: аккумулятор и обрабатываемый элемент списка. Начальное значение аккумулятора – первое числовое значение обрабатываемого списка.