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);

(defun mult-els-deep (lst num)
    (mapcar #'(lambda (arg)
                      (cond
                            ((listp arg) (mult-els-deep arg num))
                            ((numberp arg) (* arg num))
                            (t arg))) lst))
```

```
(mult-els-deep '(1 333) 2) -> (2 666);
(mult-els-deep '(1 (((5))) 333 (3 2)) 2) -> (2 (((10))) 666 (6 4));
(mult-els-deep '(1 (((5))) 333 () (3 2)) 2) -> (2 (((10))) 666 NIL (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:

```
(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))))

; 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)))
```

```
(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);
```

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:

```
(defun reduce-numbers-by-10 (lst)
    (mapcar #'(lambda (element)
              (cond ((numberp element) (- element 10))
                    (t element)))
            lst))
```

```
(reduce-numbers-by-10 '(1 2 3 (10 33) -2)) -> (-9 -8 -7 (10 33) -12);
(reduce-numbers-by-10 '(1 help 2 me 3 (10 33) -2)) -> (-9 HELP -8 ME -7 (10 33) -
12);
(reduce-numbers-by-10 '(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.

```
(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)))))
```

```
(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.

```
(defun sum-all-nums (cur)
    (cond ((numberp cur) cur)
          ((symbolp cur) 0)
          (t (+ (sum-all-nums (car cur)) (sum-all-nums (cdr cur))))))

(defun sum-all-nums-dop (lst)
    (reduce (lambda (accum cur-element)
            (cond ((numberp cur-element) (+ accum cur-element))
            ((listp cur-element) (+ accum (sum-all-nums-tail cur-element)))
                            (t accum)))
            (cons 0 lst)))
```

```
(sum-all-nums-dop '(1 2 -9 4 7)) -> 5;
(sum-all-nums-dop '(1 2 -9 4 (3 2) 7)) -> 10;
(sum-all-nums-dop '(1 (((300))) 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.