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 по лабораторной работе №8 по курсу «Функциональное и логическое программирование»

Тема: Использование функционалов

Студент: Сорокин А. П. ИУ7-66Б

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

2021 г.

5.7. Функция, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда

а) все элементы списка – числа;

```
(defun multiply-by-numb (lst k)
    (cond
        ((null lst) Nil)
        (T (cons (* (car lst) k)
            (multiply-by-numb (cdr lst) k))
        )
    )
)
(defun multiply-by-numb-f (lst k) (mapcar #'(lambda (x) (* x k)) lst))
```

б) элементы списка - любые объекты.

```
(defun multiply-by (lst k)
    (if (null lst) Nil
        (cons
            (cond
                ((listp (car lst)) (multiply-by (car lst) k))
                ((numberp (car lst)) (* (car lst) k))
                (T (car lst))
            )
            (multiply-by (cdr lst) k)
        )
    )
)
(defun multiply-by-f (lst k)
    (mapcar #'(lambda (x)
            (cond
                ((listp x) (multiply-by x k))
                ((numberp x) (* x k))
                (T x)
            )
        )
        lst
    )
)
```

6.1. Что будет результатом (mapcar 'вектор '(570-40-8))?

```
(mapcar 'вектор '(570-40-8)) -> UNDEFINED-FUNCTION ВЕКТОР
(mapcar 'vector '(570-40-8)) -> (#(|570-40-8|))
```

5.8. Написать 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 add-to-sorted-asc (lst x)
    (cond
        ((null lst) (cons x Nil))
        ((< x (car lst)) (cons x lst))
        (T (cons (car lst) (add-to-sorted-asc (cdr lst) x)))
    )
)

(defun is-between (x a b) (and (<= a x) (<= x b)))

(defun sel-between-r (lst a b res)
    (if (null lst) res
        (sel-between-r (cdr lst) a b
            (if (is-between (car lst) a b)
                (add-to-sorted-asc res (car lst))
                res
            )
        )
    )
)

(defun select-between (lst a b)
    (cond
        ((null lst) Nil)
        ((> a b) (sel-between-r lst b a Nil))
        (T (sel-between-r lst a b Nil))
    )
)
```

6.2. Функция, которая уменьшает на 10 все числа из списка аргумента этой функции.

```
(defun decrease-by-10 (lst)
    (if (null lst) Nil
        (cons
            (cond
                ((listp (car lst)) (decrease-by-10 (car lst)))
                ((numberp (car lst)) (- (car lst) 10))
                (T (car lst))
            )
            (decrease-by-10 (cdr lst))
        )
    ))
```

```
(defun decrease-by-10-f (lst)
    (mapcar #'(lambda (x) (cond
            ((listp x) (decrease-by-10-f x))
            ((numberp x) (- x 10))
            (T x)
            )
        ) lst
    )
)
```

6.3. Функция, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком.

```
(defun get-first-arg (lst)
    (cond
        ((null lst) Nil)
        ((and (listp (car lst)) (not (null (car lst)))) (car lst))
        (T (get-first-arg (cdr lst)))
    )
)
```

Поиск суммы числовых элементов смешанного структурированного списка

```
(defun sum-all-numbers (lst)
    (reduce #'(lambda (sum x)
            (+ sum
                (cond
                    ((listp x) (sum-all-numbers x))
                    ((numberp x) x)
                    (T 0)
                )
            )
        )
        lst
        :initial-value 0
    )
)

(defun sum-all-numbers-r (lst)
    (if (null lst) 0
        (+ (sum-all-numbers-r (cdr lst))
            (cond
                ((listp (car lst)) (sum-all-numbers-r (car lst)))
                ((numberp (car lst)) (car lst))
                (T 0)
            )
        )
    )
)
```

Ответы на вопросы

1. Порядок работы и варианты использование функционалов.

Виды функционалов и их порядок работы:

- **применяющие**: применяют передаваемую функцию к остальным аргументам
- (apply #'func arg_lst) – применяет функцию к аргументам из списка
- (funcall #'func arg1 … argN) – применяет функцию к аргументам arg1…argN
- **отображающие**: применяют несколько раз передаваемую функцию к аргументам
- (mapcar #'func '(x1 x2 … xn)) -> ((func x1)(func x2)…(func xn)) – применяют несколько раз передаваемую функцию к аргументам x1…xn
- (mapcar #'func lst1 lst2 … lstn) – для функций нескольких аргументов: lst1…lstn – списки 1ых, 2ых… Nых аргументов
- (maplist #'func lst) – применяет несколько раз функцию к аргументам из сначала полного списка, затем из списка без первого элемента, затем без второго и так далее до исчерпания списка.

mapcar ~ mapcan, maplist ~ mapcon: mapcan и *mapcon* используют *nconc* при сборке

- (find-if #' predicat lst) – поиск первого элемента, удовлетворяющего предикату
- (remove-if #'predicat lst) – удаляет все элементы, удовлетворяющие предикату
- (remove-if-not #'predicat lst) – удаляет все элементы, не удовлетворяющие предикату
- (reduce #'func lst) – каскадно применяет функцию к аргументам списка

Пример: (reduce #'+ '(1 2 3 4)) □ (+ (+ (+ 1 2) 3) 4)

- (every #'predicat lst) – все ли элементы удовлетворяют предикату?
- (some #'predicat lst) – есть хотя бы один элемент, удовлетворяющий предикату?

Варианты использования функционалов (примеры из лекций):

```
(defun consist-of (lst)
     (if (member (car lst) (cdr lst) 10)))

(defun all-last-elements (lst)
     (if (eql (consist-of lst) 0) (list (car lst))))

(defun collection-to-set (lst)
     (maplist #'all-last-elements lst))

(collection-to-set '(i t I g t k s i f k)) -> (g t s i f k)

(defun decart (lstX lstY)
     (mapcan #'(lambda(x)
          (mapcar #'(lambda (y)
               (list x y)) lstY)) lstX))
```