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 №6 по курсу "Функциональное и логическое программирование"

Тема Использование функционалов

Студент Ковалец К. Э.

Группа ИУ7-63Б

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

Москва — 2022 г.

1 Практические задания

1.1 Мои функции

Листинг 1.1 – my-reverse

```
(defun my-reverse-rec (lst rev-lst)
    (cond
        ((null lst)
            rev-lst)
        (T
            (my-reverse-rec (cdr lst) (cons (car lst) rev-lst)))
    )
)

(defun my-reverse (lst)
    (my-reverse-rec lst nil))

;; (MY-REVERSE '(1 2 3 4 5)) -> (5 4 3 2 1)
```

Листинг 1.2 – my-append

```
(defun my-append (lst1 lst2)
    (cond
        ((null lst1)
            lst2)
        (T
            (cons (car lst1) (my-append (cdr lst1) lst2)))
    )
)

;; (MY-APPEND '(1 2 3) '(4 5 6)) -> (1 2 3 4 5 6)
```

Листинг 1.3 – my-sort

```
(defun find-min-elem-rec (lst cur_min)
    (cond
        ((null lst)
            cur_min)
        (T
            (find-min-elem-rec (cdr lst)
                (cond
                    ((< (car lst) cur_min)
                        (setf cur_min (car lst)))
                    (T
                        cur_min))))
    )
)

(defun find-min-elem (lst)
    (find-min-elem-rec lst (car lst)))

(defun insert (lst elem elem_instead)
    (cond
        ((null lst)
            elem_instead)
        ((eql (car lst) elem_instead)
            (setf (car lst) elem))
        (T
            (insert (cdr lst) elem elem_instead))
    )
)

(defun my-sort-rec (res_lst lst)
    (let* ((min_elem (find-min-elem lst)))
        (cond
            ((null lst)
                res_lst)
            (T
                (insert lst (car lst) min_elem)
                (my-sort-rec
                    (my-append res_lst (list min_elem))
                    (cdr lst)))
        )
    )
)

(defun my-sort (lst)
    (my-sort-rec NIL lst))

;; (MY-SORT '(4 7 2 2 8 1 3)) -> (1 2 2 3 4 7 8)
```

Используя функционалы:

1.2 Задание 1

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

Листинг 1.4 – Решение задания 1

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

;; (REDUCE-BY-TEN '(10 ((11) 12))) -> (0 1 2)
```

1.3 Задание 2

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

- все элементы списка – числа,
- элементы списка – любые объекты.

Листинг 1.5 – Решение задания 2

```
;; a) все элементы списка -- числа

(defun multiply-by (lst numb)
    (mapcar #'(lambda (x) (* x numb)) lst))

;; (MULTIPLY-BY '(1 2 3) 10) -> (10 20 30)


```

```
;; б) элементы списка -- любые объекты

(defun multiply-by (lst numb)
    (mapcar
        #'(lambda (x)
            (cond
                ((numberp x)
                    (* x numb))
                ((listp x)
                    (multiply-by x numb))
                (T x)
            )
        ) lst
    )
)

;; (MULTIPLY-BY '(((1 2) 3) (4 5) a) 10) -> (((10 20) 30) (40 50) A)
```

1.4 Задание 3

Написать функцию, которая по своему списку-аргументу lst определяет является ли он палиндромом (то есть равны ли lst и (reverse lst)).

Листинг 1.6 – Решение задания 3

```
(defun compare (lst1 lst2)
    (reduce #'(lambda (b1 b2) (and b1 b2))
        (mapcar #'eql lst1 lst2)))

(defun palindrome (lst)
    (apply #'(lambda (lst1 lst2) (compare lst1 lst2))
        (list lst (my-reverse lst))))

;; (PALINDROME '(1 2 3)) -> NIL
;; (PALINDROME '(1 2 2 1)) -> T
;; (PALINDROME '(1 2 3 2 1)) -> T
```

1.5 Задание 4

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

Листинг 1.7 – Решение задания 4

```
(defun and-lst (lst)
    (reduce #'(lambda (b1 b2) (and b1 b2)) lst))

(defun or-lst (lst)
    (reduce #'(lambda (b1 b2) (or b1 b2)) lst))

(defun subset (set1 set2)
    (and-lst (mapcar #'(lambda (elem1)
        (or-lst (mapcar #'(lambda (elem2)
            (eql elem1 elem2)) set2))) set1)))

(defun set-equal (set1 set2)
    (and (subset set1 set2)
         (subset set2 set1)))

;;(SET-EQUAL '(1 2 3 4 5) '(4 2 5 1 3)) -> T
;;(SET-EQUAL '(1 2 3 4 5) '(4 2 5 1 7)) -> NIL
```

1.6 Задание 5

Написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

Листинг 1.8 – Решение задания 5

```
(defun square (lst)
    (mapcar #'(lambda (x) (* x x)) lst))

;; (SQUARE '(1 2 3 4)) -> (1 4 9 16)
```

1.7 Задание 6

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)).

Листинг 1.9 – Решение задания 6

```
(defun select-between (lst bord1 bord2)
    (my-sort (remove-if #'(lambda (x)
        (not (or (> bord1 x bord2)
                 (< bord1 x bord2)))) lst)))

;; (SELECT-BETWEEN '(1 5 4 2 3) 2 5) -> (3 4)
;; (SELECT-BETWEEN '(1 5 4 2 3) 0 6) -> (1 2 3 4 5)
;; (SELECT-BETWEEN '(1 5 4 2 3) 7 9) -> NIL
```

1.8 Задание 7

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.)

Листинг 1.10 – Решение задания 7

```
(defun decart_mult (lst1 lst2)
    (mapcan #'(lambda (x1)
        (mapcar #'(lambda (x2) (list x1 x2))
            lst2)) lst1))

;; (DECART_MULT '(1 2 3) '(1 2 3)) ->
;; ((1 1) (1 2) (1 3) (2 1) (2 2) (2 3) (3 1) (3 2) (3 3))
```

1.9 Задание 8

Почему так реализовано reduce, в чем причина?

Листинг 1.11 – Решение задания 8

```
(reduce #'+ 0)  ;; -> Error
(reduce #'+ ()) ;; -> 0
```

1.10 Задание 9

Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list, т.е. например для аргумента ((1 2) (3 4)) -> 4.

Листинг 1.12 – Решение задания 9

```
(defun sum-len-list-of-list (lst)
    (reduce #'+ (mapcar
        #'(lambda (x)
            (cond ((listp x) (sum-len-list-of-list x))
                  (T 1)))
        lst)))

;; (SUM-LEN-LIST-OF-LIST '((1 2) (3 4))) -> 4
```