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 №5

Название: Использование управляющих структур, работа со списками.

Дисциплина: Функциональное и логическое программирование

Студент ИУ7-64Б _____ Л.Е.Тартыков
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Н.Б.Толпинская
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Ю.В.Строганов
(Подпись, дата) (И.О. Фамилия)

Москва, 2022

1 Теоретические вопросы

1.1 Структуроразрушающие и не разрушающие структуру списка функции.

Не разрушающие структуру списка функции – функции, которые не изменяют исходный список. Примеры таких функций:

1. **append** – имеет переменной число параметров; выполняет объединение списков в один; возвращает список в качестве значения; выполняет копирование всех элементов списка, кроме последнего.

2. **remove** – имеет два аргумента; удаляет все вхождения элемента el из списка lst (формула 1.1).

$$(remove\ el\ lst) \tag{1.1}$$

3. **reverse** – имеет один аргумент и меняет порядок элементов аргумента на противоположный.

4. **substitute** – имеет три аргумента; заменяет все элементы списка, которые равны второму аргументу на значение первого.

5. **member** – имеет два аргумента; возвращает хвост списка, начиная со списковой ячейки, удовлетворяющей первому аргументу; иначе возвращает nil

6. **nthcdr** – имеет два аргумента (номер элемента (n) и список) ; возвращает хвост списка, начиная с n-ого элемента.

7. **nth** – имеет два аргумента (номер элемента (n), список); возвращает car-указатель на n-ый элемент списка.

8. **length** – имеет один аргумент (последовательность – строки или список); возвращает число элементов второго аргумента.

Структуроразрушающие функции – функции, которые изменяют указатели списка. Таким образом, не происходит копирования элементов списка. Такие функции обычно начинаются с символа n (nreverse, nconc, ...). Примеры таких функций:

1. **nconc** – аналогично функции append, но копирования элементов не происходит.

2. **delete** – аналогично функции remove; не создает копии исходного списка.

3. **nreverse** – аналогично функции reverse; не создает копии исходного списка.

4. **nsubstitute** – аналогично функции substitute; не создает копии исходного списка.

1.2 Отличие в работе функций cons, list, append, nconc и в их результате.

cons – имеет два аргумента и возвращает бинарный узел. Если вторым аргументом является атом, то возвращается точечная пара; если список – список. Создает в памяти списочную ячейку для аргумента.

list – имеет произвольное число аргументов и возвращает список. Создаются столько списочных ячеек, сколько переданных аргументов.

append – имеет произвольное число аргументов; важным свойством является то, что создается копия всех аргументов (новые списочные ячейки), кроме последнего, и дальнейшая работа ведется с ней; при этом сохраняется возможность работать с исходным списком. В результате функции возвращается список.

nconc – функция, аналогичная append за исключением того, что исходный список не копируется (структуроразрушающая функция). Создается столько списочных ячеек, сколько переданных аргументов.

2 Практические задания

2.1 Написать функцию, которая по своему списку-аргументу lst определяет, является ли он палиндромом.

Листинг 2.1 – Задание 1

```
1     (defun is_palindrom (lst)
2         (equal lst (reverse lst)))
```

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

Листинг 2.2 – Задание 2

```
1  (defun is_elem_in_set (elem set)
2      (cond ((null set) nil)
3          ((= elem (car set)) T)
4          (T (is_elem_in_set elem (cdr set)))
5      )
6  )
7
8  (defun is_equal_set (set_1 set_2)
9      (cond ((null set_1))
10           ((is_elem_in_set (car set_1) set_2)
11            (is_equal_set (cdr set_1) set_2)
12           )
13           (T nil)
14     )
15 )
16
17
18
```

```
(defun set_equal (set_1 set_2)
    (if (= (length set_1) (length set_2))
        (is_equal_set set_1 set_2)
        'Ошибка
    )
)
```

2.3 Напишите свои необходимые функции, которые обрабатывают таблицу из 4-х точечных пар: (страна . столица), и возвращают по стране - столицу, а по столице — страну.

Листинг 2.3 – Задание 3

```
(defvar table)
(setq table '((Russia . Moscow) (Ireland . Dublin) (Japan . Tokyo)
              (China . Pekin)))

(defun get_country (capital)
    (let ((country 0))
         (cond ((mapcar #'(lambda (x) (cond ((eq (cdr x) capital)
                                             (setq country (car x)))))
                                             table)
                country)
         )
    )
)

(defun get_capital (country)
    (let ((capital 0))
         (cond ((mapcar #'(lambda (x) (cond ((eq (car x) country)
                                             (setq capital (cdr x)))))
                                             table)
                capital)
         )
    )
)
```

2.4 Напишите функцию swap-first-last, которая переставляет в списке-аргументе первый и последний элементы.

Листинг 2.4 – Задание 4

```
(defun swap_first_last (lst)
    (cons (car (last lst))
          (nconc (cdr (nreverse (cdr (nreverse lst)))) (cons (car lst) nil)))
)
```

2.5 Напишите функцию swap-two-element, которая переставляет в списке- аргументе два указанных своими порядковыми номерами элемента в этом списке.

Листинг 2.5 – Задание 5

```
(defvar first_elem)
(defvar second_elem)

(defun append_elem (lst index index_1 index_2)
    (cond ((null lst) nil)
        ((= index index_1)
         (cons second_elem (append_elem (cdr lst) (+ index 1) index_1 index_2)))
        ((= index index_2)
         (cons first_elem (append_elem (cdr lst) (+ index 1) index_1 index_2)))
        (T (cons (car lst)
           (append_elem (cdr lst) (+ index 1) index_1 index_2)))
    )
)
```

Листинг 2.6 – Задание 5 (продолжение)

```
(defun swap_two_elements (lst index_1 index_2)
    (setq first_elem (nth index_1 lst))
    (setq second_elem (nth index_2 lst))
    (let ((len_lst (length lst))
          (index 0))
         (cond ((or (< index_1 0) (> index_1 len_lst)) "Некорректный размер")
               ((or (< index_2 0) (> index_2 len_lst)) "Некорректный размер")
               ((>= index_1 index_2) "Некорректный размер")
               (T (append_elem lst 0 index_1 index_2))
         )
    )
)
```

2.6 Напишите две функции, swap-to-left и swap-to-right, которые производят одну круговую перестановку в списке-аргументе влево и вправо, соответственно.

Листинг 2.7 – Задание 6

```
(defun without_last (lst)
    (nreverse (cdr (nreverse lst)))
)

(defun swap_to_right (lst)
    (cons (car (last lst)) (without_last lst))
)

(defun swap_to_left (lst)
    (nconc (cdr lst) (cons (car lst) nil))
)
```

2.7 Напишите функцию, которая добавляет к множеству двухэлементных списков новый двухэлементный список, если его там нет.

Листинг 2.8 – Задание 7

```
(defun check_lists_in_set (set_lists)
    (cond ((null set_lists) T)
        ((not (listp (car set_lists))) nil)
        ((if (= 2 (length (car set_lists)))
            (check_lists_in_set (cdr set_lists)) nil))
    )
)

(defun is_in_set (new_list set_lists)
    (cond ((null set_lists) T)
          ((not (equal new_list (car set_lists)))
           (is_in_set new_list (cdr set_lists)))
    )
)

(defun append_new_elem (new_list set_lists)
    (cond ((not (listp new_list)) "Ошибка")
          ((not (listp set_lists)) "Ошибка")
          ((not (check_lists_in_set set_lists)) "Ошибка")
          ((is_in_set new_list set_lists) (nconc set_lists `(,new_list)))
          (T set_lists)
    )
)
```

2.8 Напишите функцию, которая умножает на заданное число-аргумент первый числовой элемент списка из заданного 3-х элементного списка- аргумента, когда а) все элементы списка — числа, б) элементы списка – любые объекты.

Листинг 2.9 – Задание 8

```
(defun multiply_var_1 (lst number)
    (cond ((null lst) nil)
          (T (cons (* number (car lst)) (cdr lst)))
    )
)

(defun multiply_var_2 (lst number)
    (cond ((null lst) nil)
          (T (if (numberp (car lst))
                 (cons (* number (car lst)) (cdr lst))
                 (cons (car lst) (multiply_var_2 (cdr lst) number)))
          )
    )
)

(defun check_lst (lst number)
    (cond ((not (numberp number)) nil)
          ((not (listp lst)) nil)
          ((not (= 3 (length lst))) nil)
          (T T)
    )
)

(defun multiply_list_by_number_1 (lst number)
    (cond ((check_lst lst number) (multiply_var_1 lst number)))
)
(defun multiply_list_by_number_2 (lst number)
    (if (check_lst lst number) (multiply_var_2 lst number) "Ошибка")
)
```

2.9 Напишите функцию, select-between, которая из списка-аргумента из 5 чисел выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка (упорядоченного по возрастанию списка чисел).

Листинг 2.10 – Задание 9

```
(defun select_between (lst board_left board_right)
    (let ((result_lst nil))
        (setq result_lst (if (check_input_data lst board_left board_right)
        (find_left_board lst board_left board_right 0)
        nil
                        )
        )
        (if (not (null result_lst)) (bubble_sort_asc result_lst) "Ошибка")
    )
)

(defun check_input_data (lst board_left board_right)
    (cond ((not (= (length lst) 5)) nil)
          ((< board_left 0) nil)
          ((>= board_right 5) nil)
          ((<= board_right board_left) nil)
          (T T)
    )
)

(defun find_left_board (lst board_left board_right index_board)
    (if (= index_board board_left)
        (find_right_board (cdr lst) board_right (+ index_board 1))
        (find_left_board (cdr lst) board_left board_right (+ index_board 1))
    )
)




(defun find_right_board (lst board_right index_board)
    (if (= index_board board_right)
        (cons (car lst) nil)
        (cons (car lst) (find_right_board (cdr lst) board_right (+ index_board
           1)))
    )
)

(defun bubble (lst)
    (cond ((atom (cdr lst)) lst)
          ((> (car lst) (cadr lst))
            (cons (cadr lst) (bubble (cons (car lst) (cddr lst)) ))
          )
          (T lst)
    )
)


```

```
47 (defun bubble_sort_asc (lst)
48     (cond ((atom (cdr lst)) lst)
49           (T (bubble (cons (car lst) (bubble_sort_asc (cdr lst)))))
50     )
51 )
```