



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 №5 по курсу "Функциональное и логическое программирование"

Тема Использование управляющих структур, работа со списками

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

```
1 (defun my-reverse-rec (lst rev-lst)
2   (cond
3     ((null lst)
4      rev-lst)
5     (T
6      (my-reverse-rec (cdr lst) (cons (car lst) rev-lst))))
7   )
8 )
9
10 (defun my-reverse (lst)
11   (my-reverse-rec lst nil))
12
13 ;; (MY-REVERSE '(1 2 3 4 5)) -> (5 4 3 2 1)
```

Листинг 1.2 – my-length

```
1 (defun my-length-rec (set len)
2   (cond
3     ((null set)
4      len)
5     (T
6      (my-length-rec (cdr set) (+ len 1))))
7   )
8 )
9
10 (defun my-length (set)
11   (my-length-rec set 0))
12
13 ;; (MY-LENGTH '(1 2 3 4)) -> 4
```

Листинг 1.3 – my-last

```
1 (defun my-last (lst)
2   (cond
3     ((null (cdr lst))
4      (list (car lst)))
5     (T
6      (my-last (cdr lst))))
7   )
8 )
9
10 ;; (MY-LAST '(1 2 4 5)) -> (5)
```

Листинг 1.4 – my-append

```
1 (defun my-append (lst1 lst2)
2   (cond
3     ((null lst1)
4      lst2)
5     (T
6      (cons (car lst1) (my-append (cdr lst1) lst2))))
7   )
8 )
9
10 ;; (MY-APPEND '(1 2 3) '(4 5 6)) -> (1 2 3 4 5 6)
```

Листинг 1.5 – my-sort

```

1 (defun find-min-elem-rec (lst cur_min)
2   (cond
3     ((null lst)
4      cur_min)
5     (T
6      (find-min-elem-rec (cdr lst)
7                          (cond
8                            ((< (car lst) cur_min)
9                             (setf cur_min (car lst)))
10                           (T
11                            cur_min))))))
12 )
13 )
14
15 (defun find-min-elem (lst)
16   (find-min-elem-rec lst (car lst)))
17
18 (defun insert (lst elem elem_instead)
19   (cond
20     ((null lst)
21      elem_instead)
22     ((eql (car lst) elem_instead)
23      (setf (car lst) elem))
24     (T
25      (insert (cdr lst) elem elem_instead)))
26 )
27 )
28
29 (defun my-sort-rec (res_lst lst)
30   (let* ((min_elem (find-min-elem lst)))
31     (cond
32       ((null lst)
33        res_lst)
34       (T
35        (insert lst (car lst) min_elem)
36        (my-sort-rec
37          (my-append res_lst (list min_elem))
38          (cdr lst))))
39     )
40   )
41 )
42
43 (defun my-sort (lst)
44   (let* ((res_lst NIL))
45     (my-sort-rec res_lst lst)))
46
47 ;; (MY-SORT '(4 7 2 2 8 1 3)) -> (1 2 2 3 4 7 8)

```

1.2 Задание 1

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 1

```
1 (defun compare (lst1 lst2)
2   (cond
3     ((null lst1)
4      T)
5     ((eql (car lst1) (car lst2))
6      (compare (cdr lst1) (cdr lst2)))
7   )
8 )
9
10 (defun palindrome (lst)
11   (compare lst (my-reverse lst)))
12
13 ;; (palindrome '(1 2 3 2 1)) -> T
14 ;; (palindrome '(1 2 3 2)) -> NIL
```

1.3 Задание 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 1.7 – Решение задания 2

```
1 (defun elem-in-set (elem set)
2   (cond
3     ((null set)
4      NIL)
5     ((eql elem (car set))
6      T)
7     (T
8      (elem-in-set elem (cdr set))))
9   )
10 )
11
12 (defun check-set-equal (set1 set2)
13   (cond
14     ((null set1)
15      T)
16     ((elem-in-set (car set1) set2)
17      (check-set-equal (cdr set1) set2))
18   )
19 )
20
21
22 (defun set-equal (set1 set2)
23   (if (eql (my-length set1) (my-length set2))
24       (check-set-equal set1 set2)
25   )
26 )
27
28 ;; (SET-EQUAL '(1 2 3 4 5) '(4 2 5 1 3)) -> T
29 ;; (SET-EQUAL '(1 2 3 4 5) '(4 2 5 1 7)) -> NIL
```

1.4 Задание 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 1.8 – Решение задания 3

```
1 (defun find-capital (country table)
2   (cond
3     ((null table)
4      Nil)
5     ((eql country (caar table))
6      (cdar table))
7     (T
8      (find-capital country (cdr table))))
9   )
10 )
11
12 (defun find-country (capital table)
13   (cond
14     ((null table)
15      Nil)
16     ((eql capital (cdar table))
17      (caar table))
18     (T
19      (find-country capital (cdr table))))
20   )
21 )
22
23 ;; (find-capital
24 ;;   'Russia
25 ;;   '((USA . Washington)
26 ;;     (Russia . Moscow)
27 ;;     (Germany . Berlin)))
28 ;; -> MOSCOW
29
30 ;; (find-country
31 ;;   'Moscow
32 ;;   '((USA . Washington)
33 ;;     (Russia . Moscow)
34 ;;     (Germany . Berlin)))
35 ;; -> RUSSIA
```

1.5 Задание4

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

Листинг 1.9 – Решение задания 4

```
1 (defun swap-first-last (lst)
2   (my-append
3     (my-append (my-last lst)
4                 (my-reverse (cdr (my-reverse (cdr lst)))))
5   )
6   (list (car lst))
7 )
8 )
9
10 ;; (SWAP-FIRST-LAST '(1 2 3 4 5)) -> (5 2 3 4 1)
```

1.6 Задание 5

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

Листинг 1.10 – Решение задания 5

```
1 (defun get-elem (lst i)
2   (cond
3     ((null lst)
4      NIL)
5     ((eq i 0)
6      (car lst))
7     (T
8      (get-elem (cdr lst) (- i 1)))
9   )
10 )
11
12 (defun set-elem (lst elem i)
13   (cond
14     ((null lst)
15      NIL)
16     ((eq i 0)
```



```

17         (setf (car lst) elem))
18     (T
19         (set-elem (cdr lst) elem (- i 1)))
20 )
21 )
22
23 (defun swap-two-ellement (lst i j)
24     (let* ((tmp_elem (get-elem lst i)))
25         (set-elem lst (get-elem lst j) i)
26         (set-elem lst tmp_elem j)
27         lst
28     )
29 )
30
31 ;; (SWAP-TWO-ELLEMENT '(1 2 3 4 5) 1 4) -> (1 5 3 4 2)

```

1.7 Задание 6

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

Листинг 1.11 – Решение задания 6

```

1 (defun swap-to-left (lst)
2     (my-append (cdr lst) (list (car lst))))
3
4 ;; (SWAP-TO-LEFT '(1 2 3 4)) -> (2 3 4 1)
5
6 (defun swap-to-right (lst)
7     (my-append (my-last lst) (my-reverse (cdr (my-reverse lst)))))
8
9 ;; (SWAP-TO-RIGHT '(1 2 3 4)) -> (4 1 2 3)

```

1.8 Задание 7

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

Листинг 1.12 – Решение задания 7

```
1 (defun check-set-equal (set1 set2)
2   (cond
3     ((null set1)
4      T)
5     ((eql (car set1) (car set2))
6      (check-set-equal (cdr set1) (cdr set2)))
7     (T
8      NIL)
9   )
10 )
11
12 (defun set-equal (set1 set2)
13   (if (eql (my-length set1) (my-length set2))
14       (check-set-equal set1 set2)
15   )
16 )
17
18 (defun is-sublist (lst sublist)
19   (cond
20     ((null lst)
21      NIL)
22     ((set-equal (car lst) sublist)
23      T)
24     (T
25      (is-sublist (cdr lst) sublist))
26   )
27 )
28
29 (defun add-list (lst new_lst)
30   (if (is-sublist lst new_lst)
31       lst
32       (my-append lst (list new_lst)))
33   )
34 )
35
36 ;; (ADD-LIST '((1 2) (3 4) (1 1)) '(5 5)) -> ((1 2) (3 4) (1 1) (5 5))
37 ;; (ADD-LIST '((1 2) (3 4) (1 1)) '(1 1)) -> ((1 2) (3 4) (1 1))
```

1.9 Задание 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 1.13 – Решение задания 8

```
1 ;; а) все элементы списка -- числа
2 (defun mult-first-num-arg (lst num)
3   (setf (car lst) (* (car lst) num))
4   lst
5 )
6
7 ;; (MULT-FIRST-NUM-ARG '(1 2 3 4) 5) -> (5 2 3 4)
8
9 ;; б) элементы списка -- любые объекты
10 (defun mult-first-num-arg (lst num)
11   (cond
12     ((null lst)
13      NIL)
14     ((numberp (car lst))
15      (setf (car lst) (* (car lst) num)))
16     (T
17      (mult-first-num-arg (cdr lst) num))
18   )
19   lst
20 )
21
22 ;; (MULT-FIRST-NUM-ARG '(a (1 2) 3 4) 5) -> (A (1 2) 15 4)
23 ;; (MULT-FIRST-NUM-ARG '(a (1 2)) 5) -> (A (1 2))
```

1.10 Задание 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 чисел списка (+ 2 балла)).

Листинг 1.14 – Решение задания 9

```
1 (defun find-right-numbers (res_lst lst b1 b2)
2   (let* ((cur_elem (car lst)))
3     (cond
4       ((null lst)
5        res_lst)
6       ((and (numberp cur_elem)
7              (> cur_elem b1)
8              (< cur_elem b2))
9        (find-right-numbers
10         (my-append res_lst (list cur_elem))
11         (cdr lst) b1 b2))
12       (T
13        (find-right-numbers res_lst (cdr lst) b1 b2))
14     )
15   )
16 )
17
18 (defun select-between (lst b1 b2)
19   (let* ((res_lst NIL))
20     (my-sort (find-right-numbers res_lst lst b1 b2))
21   )
22 )
23
24 ;; (SELECT-BETWEEN '(1 5 4 2 3) 2 5) -> (3 4)
25 ;; (SELECT-BETWEEN '(1 5 4 2 3) 0 6) -> (1 2 3 4 5)
26 ;; (SELECT-BETWEEN '(1 5 4 2 3) 7 9) -> NIL
```

2 Ответы на теоретические вопросы к лабораторной работе

2.1 Структурноразрушающие и не разрушающие структуру списка функции

2.1.1 Не разрушающие структуру функции

Данные функции не меняют сам объект-аргумент, а создают копию. Если после использования функции сохраняется возможность работать с исходным списком, то значит функция не разрушает структуру.

- **append** – объединяет списки. Можно передать больше двух аргументов. Создает копию для всех аргументов, кроме последнего;
- **reverse** – возвращает копию исходного списка, элементы которого переставлены в обратном порядке. В целях эффективности работает только на верхнем уровне;
- **remove** – данная функция удаляет элемент по значению. По умолчанию используется *eql* для сравнения на равенство, но можно передать другую функцию через ключевой параметр *test*. Модифицирует список, но работает с копией, поэтому не разрушает структуру;
- **rplace** – переставляет *car*-указатель на второй элемент-аргумент (S-выражение);
- **rplacd** – переставляет *cdr*-указатель на второй элемент-аргумент (S-выражение);
- **subst** – заменяет все элементы списка, которые равны второму переданному элементу-аргументу на первый элемент аргумент. По умолчанию для сравнения используется функция *eql*.

2.1.2 Структуроразрушающие функции

Данные функции меняют сам объект-аргумент, невозможно вернуться к исходному списку. Чаще всего такие функции начинаются с префикса *n*. Если после использования функции теряется возможность работы с тем списком, который был изначально, то значит его структура разрушилась.

- **nconc** – работает аналогично функции *append*, только копирует не свои аргументы, а разрушает структуру;
- **nreverse** – работает аналогично функции *reverse*, только не создает копии списка;
- **nsubst** – работает аналогично функции *subst*, только не создает копии списка.

2.2 Отличие в работе функций *cons*, *list*, *append*, *ncons* и их результаты

- **cons** – имеет фиксированное количество аргументов (два). В случае, когда аргументами являются атомы создает точечную пару. В случае, когда первый аргумент атом а второй список, атом становится головой, а второй аргумент (список) становится хвостом;
- **list** – не имеет фиксированное количество аргументов. Создает список, у которого голова – это первый аргумент, хвост – все остальные аргументы. Результат всегда список;
- **append** – принимает на вход произвольное количество аргументов и для всех аргументов, кроме последнего создает копию, ссылая при этом последний элемент каждого списка-аргумента на первый элемент следующего по порядку списка-аргумента (так как модифицируются все списки-аргументы, кроме последнего, копирование для последнего не делается в целях эффективности);

- **nconc** – работает как функция *append*, но строит такой результат следующим образом: для каждого непустого аргумента-списка, *nconc* устанавливает в *cdr* его последней *cons*-ячейки ссылку на первую *cons*-ячейку следующего непустого аргумента-списка. После этого она возвращает первый список, который теперь является головой результата. Главное отличие от функции *append* заключается в том, что функция *nconc* разрушает структуру списка.