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 ИУ7-65Б _____ Бугаенко А.П.
(подпись, дата) (Фамилия И.О.)

Преподователь _____ Толпинская Н.Б.
(подпись, дата) (Фамилия И.О.)

Москва, 2022

1 Цели и задачи работы

Цель работы — приобрести навыки работы с управляющими структурами Lisp. Задачи работы —изучить работу функций с произвольным количеством аргументов, функций разрушающих и неразрушающих структуру исходных аргументов.

2 Теоретические вопросы

2.1 Структуроразрушающие и не разрушающие структуру списка функции

Функции можно разбить на две группы - разрушающие структуру и не разрушающие структуру. Для работы со списком его необходимо создать, получить доступ и модифицировать. Функции, разрушающие структуру - изменяют структуру списка. Функции не разрушающую структуру - производят какие-то операции без изменения поданного на вход списка.

В функции, разрушающие структуру списка входят - nconc, nreverse В функции, не разрущающие структуру списка входят - append, reverse, list, cons

2.2 Отличие в работе и результат работы функций cons, list, append, nconc

При работе функции cons создаётся бинарный узел, car которого указывает на первый аргумент, а cdr на второй. В результате работы создаётся новый объект и при этом не разрушаются списки, поданные на вход.

При работе функции list создаётся новый список из поданных на вход элементом, при этом сами элементы не меняются.

При работе функции append создаётся копия списка, в конец которой добавляется новый элемент. При этом старый список остаётся без изменений.

При работе функции nconc новый список создаётся путём присвоения cdr указателям концов списков начал следуюещго списка. В результате разрушается структура списка, поскольку при попытке получить доступ к старому списку будет выводится новый список начиная с аргумента, с которым был связан символ.

3 Практические задания

3.1 Задание 1

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

```
(defun my_reverse_rec (lst new_lst)
  (if (not (eql (cdr lst) nil))
      (my_reverse_rec (cdr lst) (cons (car lst) new_lst))
      (cons (car lst) new_lst)))

(defun my_reverse (lst)
  (my_reverse_rec lst nil))

(defun cmp_list_rec (lst_1 lst_2)
  (if (eql (car lst_1) (car lst_2))
      (if (not (or (eql (cdr lst_1) nil) (eql (cdr lst_2) nil)))
          (cmp_list_rec (cdr lst_1) (cdr lst_2))
          (if (and (eql (cdr lst_1) nil) (eql (cdr lst_2) nil))
              t
              nil))
      nil))

(defun check_palindrom (lst)
  (cmp_list_rec lst (my_reverse lst)))
```

3.2 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.

```
(defun in-list (lst elem)
  (if (eql (car lst) elem)
      t
      (if (eql (cdr lst) nil)
          nil
          (in-list (cdr lst) elem)
          )
      )
  )

(defun set-equal-rec (set_1 set_2)
  (if (in-list set_1 (car set_2))
      (if (not (eql (cdr set_2) nil))
          (set-equal-rec set_1 (cdr set_2))
          t
          )
```

```
17        nil
18        )
19    )
20
21  (defun set-equal (set_1 set_2)
22    (and (set-equal-rec set_1 set_2) (set-equal-rec set_2 set_1))
23    )
```

3.3 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.

```
1  (setf  table (list (cons 'russia 'moscow) (cons 'germany 'berlin) (cons 'france
       'paris) (cons 'spain 'madrid)))
2
3  (defun find-cntry-rec (table capital)
4    (if (eql (cdar table) capital)
5        (caar table)
6        (if (not (eql (cdr table) nil))
7            (find-cntry-rec (cdr table) capital)
8            nil
9            )
10       )
11   )
12
13 (defun find-capital-rec (table country)
14   (if (eql (caar table) country)
15       (cdar table)
16       (if (not (eql (cdr table) nil))
17           (find-cntry-rec (cdr table) country)
18           nil
19           )
20       )
21   )
```

3.4 Задание 4

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

```
1  (defun swap-first-last-rec (lst first)
2    (if (eql (cddr lst) nil)
3        (if (defvar new-first (cdr lst))
4            (if (and
5                 (or (setf new-first (cons (cadr lst) (cdr first))) t)
6                 (or (setf (cdr first) nil) t)
```

```
7                (or (setf (cdr lst) first) t)
8                )
9               new-first
10               nil
11            )
12           nil)
13       (swap-first-last (cdr lst) first)
14       )
15   )
16
17 (defun swap-first-last (lst)
18   (swap-first-last-rec lst lst))
```

3.5 Задание 5

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

```
1  (defun get-by-index (lst index cntr)
2    (if (not (eql lst nil))
3        (if (eql cntr index)
4            lst
5            (get-by-index (cdr lst) index (+ cntr 1))
6            )
7        nil
8     )
9    )
10
11 (defun swap-two-ellement (lst first second)
12   (let ((first_elem (get-by-index lst first 0)) (second_elem (get-by-index lst
       second 0)))
13     (let ((temp (car first_elem)))
14       (setf (car first_elem) (car second_elem))
15       (setf (car second_elem) temp)
16       lst
17       )
18     )
19   )
```

3.6 Задание 6

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

```
1  (defun my-last (lst)
2    (if (eql (cdr lst) nil)
3        lst
```

```
4       (my-last (cdr lst))
5       )
6   )
7
8 (defun swap-to-left-rec (lst temp)
9   (if (not (eql lst nil))
10      (let ((temp-local (car lst)))
11        (setf (car lst) temp)
12        (swap-to-left-rec (cdr lst) temp-local)
13        )
14      )
15  )
16
17 (defun swap-to-left (lst)
18   (let ((temp (car lst)))
19     (setf (car lst) (car (my-last lst)))
20     (swap-to-left-rec (cdr lst) temp)
21     lst
22     )
23   )
24
25 (defun my_reverse_rec (lst new_lst)
26   (if (not (eql (cdr lst) nil))
27       (my_reverse_rec (cdr lst) (cons (car lst) new_lst))
28       (cons (car lst) new_lst)))
29
30 (defun my_reverse (lst)
31   (my_reverse_rec lst nil))
32
33 (defun swap-to-right (lst)
34   (let ((temp nil) (reverse-lst (my_reverse lst)))
35     (setf temp (car reverse-lst))
36     (setf (car reverse-lst) (car (my-last reverse-lst)))
37     (swap-to-left-rec (cdr reverse-lst) temp)
38     (my_reverse reverse-lst)
39     )
40   )
```

3.7 Задание 7

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

```
1 (defun cmp-lst (lst_1 lst_2)
2   (if (and
3        (eql lst_1 nil)
4        (eql lst_2 nil)
```

```
        )
       t
       (if
        (and
         (not (eql lst_1 nil))
         (not (eql lst_2 nil))
         )
        (if (eql (car lst_1) (car lst_2))
            (cmp-lst (cdr lst_1) (cdr lst_2))
            )
        )
       )
  )

(defun check-duosublist (lst sublst)
  (if (eql lst nil)
      nil
      (if (cmp-lst (car lst) sublst)
          t
          (check-duosublist (cdr lst) sublst)
          )
      )
  )

(defun append-if-not-in-lst (lst sublst)
  (if (not (check-duosublist lst sublst))
      (append lst (list sublst))
      )
  )
```

3.8 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, 6) элементы списка — любые объекты.

```
(defun mult-first-num (lst num)
  (if (not (eql lst nil))
      (and
       (setf (car lst)  (car lst) num))lst)))(defun mult-first-maybe-num-rec (lst num first)(if (not (eql lst
           nil))(if (numberp (car lst))(and(setf (car lst) ( (car lst) num))first)(mult-first-maybe-num-rec (cdr lst)
           num first))))(defun mult-first-maybe-num (lst num)(mult-first-maybe-num-rec lst num lst))
```

3.9 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 списка чисел (+ 2 балла)).

```
1  (defun count-size-rec (lst cnt)
2    (if (eql (cdr lst) nil)
3        cnt
4        (count-size-rec (cdr lst) (+ cnt 1)))
5    )
6
7  (defun count-size (lst)
8    (count-size-rec lst 1)
9    )
10
11 (defun check-nums (lst)
12   (if (eql (cdr lst) nil)
13       t
14       (if (numberp (car lst))
15           (check-nums (cdr lst))
16           nil
17        )
18    )
19   )
20
21 ;; 0 -- start -- num -- end -->
22
23 (defun select-between-rec (lst new-lst start end)
24   (if (not (eql lst nil))
25       (if (and
26            (>= (car lst) start)
27            (<= (car lst) end)
28            )
29           (if (eql new-lst nil)
30               (select-between-rec (cdr lst) (list (car lst)) start end)
31               (select-between-rec (cdr lst) (append new-lst (list (car lst))) start
                     end)
32               )
33           (select-between-rec (cdr lst) new-lst start end)
34           )
35       new-lst
36       )
37   )
38
39
40 (defun select-between (lst start end)
```

```
41  (if (and
42       (eql (count-size lst) 5)
43       (check-nums lst)
44       )
45      (select-between-rec lst nil start end)
46      )
47  )
```