



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»

Отчет по лабораторной работе №5  
по дисциплине «Функциональное и логическое  
программирование»

Тема Использование управляющих структур, работа со списками

Студент Зайцева А. А.

Группа ИУ7-62Б

Оценка (баллы) \_\_\_\_\_

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

Москва — 2022 г.

# Теоретические вопросы

## 1. Структуроразрушающие и не разрушающие структуру списка функции

Функции, работающие со списками, делятся на:

- функции, не разрушающие структуру списка (сохраняется возможность работать с исходным списком);
- функции, разрушающие структуру списка (не сохраняется возможность работы с исходным списком, зато функция выполняется быстрее по сравнению со своим не разрушающим аналогом, так как не создаются копии cons-ячеек).

Из-за такого разделения существует много дублирующих функций: функциям, не разрушающим структуру списка (`reverse`, `substitute`, ...) соответствуют структуроразрушающие функции, которые, как правило, начинаются с буквы «n», как признак того, что не создаются копии (`nreverse`, `nsubstitute`, ...). `cons` является структуроразрушающим аналогом `append`, `delete` – структуроразрушающим аналогом `remove`.

## 2. Отличие в работе функций `cons`, `list`, `append`, `ncons` и в их результате

`cons` принимает 2 указателя на любые S-выражения и возвращает новую cons-ячейку (списковую ячейку), содержащую 2 значения. Если второе значение не `NIL` и не другая cons-ячейка, то ячейка печатается как два значения в скобках, разделённые точкой (так называемая точечная пара). Иначе, по сути, эта функция включает значение первого аргумента в начало списка, являющегося значением второго аргумента.

Функция `list`, составляющая список из значений своих аргументов (у которого голова – это первый аргумент, хвост – все остальные аргументы), создает столько списковых ячеек, сколько аргументов ей было передано. Эта функция относится к особым, поскольку у неё может быть произвольное число аргументов, но при этом все аргументы вычисляются.

`append` принимает произвольное количество аргументов-списков и соединяет (сливает) элементы верхнего уровня всех списков в один список. Действие

append иногда называют конкатенацией списков. В результате должен быть построен новый список.

Например: (append (list 1 2) (list 3 4)) ==> (1 2 3 4).

С точки зрения функционального подхода, задача функции append - вернуть список (1 2 3 4) не изменяя ни одну из cons-ячеек в списках-аргументах (1 2) и (3 4). append на самом деле создаёт только две новые cons-ячейки, чтобы хранить значения 1 и 2, соединяя их вместе и делая ссылку из CDR второй ячейки на первый элемент последнего аргумента - списка (3 4). После этого функция возвращает cons-ячейку содержащую 1. Ни одна из входных cons-ячеек не была изменена, и результатом, как и требовалось, является список (1 2 3 4). Единственная хитрость в том, что результат, возвращаемый функцией append имеет общие cons-ячейки со списком (3 4). Таким образом, если последний переданный список будет модифицирован, то итоговый список будет также изменен.

ncons - это структуроразрушающая версия append. Как и append, ncons возвращает соединение своих аргументов, но строит такой результат следующим образом: для каждого непустого аргумента-списка, ncons 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-соединения.

Итак, отличия: cons является базисной, list, append, ncons - нет; list, append, ncons принимают произвольное количество аргументов (причем аргументами append и ncons могут быть только списки), cons - фиксированное (два); cons создает точечную пару или список (в зависимости от второго аргумента), list, append и ncons - список; cons и list создают новые списковые ячейки (все), append имеет общие списковые ячейки с последним списком, ncons не создает cons-ячеек; cons является структуроразрушающей, а cons, list и append - нет.

Пусть (setf lst1 '(a b)); (setf lst2 '(c d)).

```
1 (cons lst1 lst2) => ((A B) C D)
2 lst1 => (A B)
3 (list lst1 lst2) => ((A B) (C D))
4 lst1 => (A B)
5 (append lst1 lst2) => (A B C D)
6 lst1 => (A B)
7 (ncons lst1 lst2) => (A B C D)
8 lst1 => (A B C D)
```

# Практические задания

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

Работа только с верхним уровнем: список `'((1 2) 3 (1 2))` считается палиндромом, а список `'((1 2) 3 (2 1))` – нет.

```
1 ; с использованием стандартных функций
2 (defun is_palindrome (lst) (equalp lst (reverse lst)))
3
4 ; с использованием самостоятельно реализованных функций
5 (defun my_length (lst &optional (len 0))
6   (cond
7     ((null lst) len)
8     ((my_length (cdr lst) (+ len 1)))
9   )
10 )
11
12 (defun my_equalp (x1 x2)
13   (cond
14     ((numberp x1) (and (numberp x2) (= x1 x2)))
15     ((atom x1) (and (atom x2) (eq x1 x2)))
16     ((consp x1) (and (consp x2) (my_equalp (car x1) (car x2)) (
17       my_equalp (cdr x1) (cdr x2))))
18     ((listp x1) (and (listp x2) (= (my_length x1) (my_length x2)) (
19       my_equalp (car x1) (car x2)) (my_equalp (cdr x1) (cdr x2))))
20   )
21 )
22
23 (defun my_rev_upper (lst)
24   (if (null lst)
25     Nil
26     (append
27       (my_rev_upper (cdr lst))
28       (cons (car lst) Nil))
29   )
30 )
31
32 (defun is_palindrome (lst) (my_equalp lst (my_rev_upper lst)))
33
34 ; тесты
35 (is_palindrome '(1 2 3)) => NIL
36 (is_palindrome '(1 2 1)) => T
```

```
36 | (is_palindrome '((1) 2 1)) => NIL
37 |
38 | (is_palindrome '((1 2) 3 (1 2))) => T
39 | (is_palindrome '((1 2) 3 (2 1))) => NIL
```

Работа по всем уровням (и использование самостоятельно реализованных функций): список `'((1 2) 3 (1 2))` не считается палиндромом, а список `'((1 2) 3 (2 1))` – да.

```
1 (defun my_rev_deeper (lst)
2   (if (null lst)
3       Nil
4       (append
5         (my_rev_deeper (cdr lst))
6         (if (atom (car lst))
7             (cons (car lst) Nil)
8             (list (my_rev_deeper (car lst))))
9       )
10  )
11 )
12 )
13
14 (defun is_palindrome (lst) (my_equalp lst (my_rev_deeper lst)))
15
16 ; тесты
17 (is_palindrome '(1 2 3)) => NIL
18 (is_palindrome '(1 2 1)) => T
19 (is_palindrome '((1) 2 1)) => NIL
20
21 (is_palindrome '((1 2) 3 (1 2))) => Nil
22 (is_palindrome '((1 2) 3 (2 1))) => T
23 (is_palindrome '((1 (2 3)) 4 ((3 2) 1))) => T
```

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

Если элементами множеств являются только атомы (с использованием встроенных функций):

```
1
2 (defun set-equal (set1 set2)
3   (if (null set1)
4       (null set2)
5       (if (member (car set1) set2)
6           (set-equal (cdr set1) (remove (car set1) set2 :count 1))
7           Nil)
8   )
9 )
10 )
```

Комментарии к решению:

- если передаваемые аргументы – множества в том понимании, что элементы в них не повторяются, то дополнительный параметр :count 1 в функции remove можно не указывать;
- ветвь if (null set1) необходима, так как если множества совпадают, то при последнем вызове set-equal аргументами будут два пустых списка. Тогда далее будет произведена проверка, является ли (car set1)=Nil членом set2=Nil, результатом которой будет Nil, и задача будет решена неверно.

Если элементами множеств могут быть другие множества (и с использованием собственных функций):

```
1 (defun my_member (elem lst)
2   (if (null lst)
3       Nil
4       (if (my_equalp elem (car lst))
5           lst
6           (my_member elem (cdr lst))))
7 )
8 )
9
10 (defun my_list_without_last (lst)
11   (if (null (cdr lst))
12       Nil
13       (cons (car lst) (my_list_without_last (cdr lst))))
14 )
15 )
16
17 (defun my_member_before (elem lst)
18   (if (my_member elem lst)
19       (my_member_before elem (my_list_without_last lst))
20       lst
21   )
22 )
23
24 (defun my_remove_first (elem lst)
25   (append
26     (my_member_before elem lst)
27     (cdr (my_member elem lst)))
28 )
29
30 (defun set-equal (set1 set2)
31   (if (null set1)
32       (null set2)
33       (if (my_member (car set1) set2)
34           (set-equal (cdr set1) (my_remove_first (car set1) set2))
```

```

35         Nil
36     )
37 )
38 )
39
40 (set-equal '(1 2 3) '(1 2 3)) => T
41 (set-equal '() '()) => T
42 (set-equal '(1 2) '(1 2 3)) => NIL
43 (set-equal '(1) '(1 2)) => NIL
44 (set-equal '(1 2) '(1 (2))) => NIL
45 (set-equal '(1 (2)) '(1 (2))) => T
46 (set-equal '(1 (2)) '(1 (2 3))) => NIL

```

**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 (defun find_capital_by_country (table country)
2   (if (null table)
3       (and (print '(No such country)) Nil)
4       (if (eq (caar table) country)
5           (cdar table)
6           (find_capital_by_country (cdr table) country)
7       )
8   )
9 )
10
11 (defun find_country_by_capital (table capital)
12   (if (null table)
13       (and (print '(No such capital)) Nil)
14       (if (eq (cdar table) capital)
15           (caar table)
16           (find_country_by_capital (cdr table) capital)
17       )
18   )
19 )
20
21
22
23 (defvar table)
24 (setq table '((Russia . Moscow) (GreatBritain . London) (France . Paris)
25               (Italy . Roma)))
26
27 (find_capital_by_country table 'Russia) => MOSCOW
28 (find_capital_by_country table 'Italy) => ROMA

```



```

28 (find_capital_by_country table 'USA) => "(NO SUCH COUNTRY)" NIL
29
30 (find_country_by_capital table 'Moscow) => Russia
31 (find_country_by_capital table 'Paris) => FRANCE
32 (find_country_by_capital table 'Washington) => "(NO SUCH CAPITAL)" NIL

```

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

```

1 (defun my_last (lst)
2   (if (null (cdr lst))
3       (car lst)
4       (my_last (cdr lst))
5   )
6 )
7
8 (defun my_list_without_last (lst)
9   (if (null (cdr lst))
10      Nil
11      (cons (car lst) (my_list_without_last (cdr lst)))
12   )
13 )
14
15 (defun swap-first-last (lst)
16   (if (cdr lst)
17       (append
18         (cons (my_last lst) Nil)
19         (my_list_without_last (cdr lst))
20         (cons (car lst) Nil)
21       )
22       lst
23   )
24 )
25
26 (swap-first-last '(1 2 3)) => (3 2 1)
27
28 ; для 2 приведенных ниже примеров и нужна проверка if (cdr lst)
29 (swap-first-last '()) => Nil
30 (swap-first-last '(1)) => (1)
31
32 (swap-first-last '(1 2)) => (2 1)
33 (swap-first-last '((1 2) 3 4 5)) => (5 3 4 (1 2))

```

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

```
1 ; индексация начинается с 0
2 (defun my_nth (lst n)
3   (if (or (< n 0) (< (my_length lst) (- n 1)))
4     (print '(Error no such index))
5     (if (= n 0)
6         (car lst)
7         (my_nth (cdr lst) (- n 1)))
8   )
9 )
10 )
11
12 ; from-элемента в результате нет
13 (defun list_slice_from (lst from)
14   (if (= from -1)
15       lst
16       (list_slice_from (cdr lst) (- from 1))
17   )
18 )
19
20 (defun list_without_last_n (lst n)
21   (if (= n 0)
22       lst
23       (list_without_last_n (my_list_without_last lst) (- n 1))
24   )
25 )
26
27 ; to-элемента в результате нет
28 (defun list_slice_to (lst to)
29   (list_without_last_n lst (- (my_length lst) to))
30 )
31
32 ; элементы to и from не входят в результат
33 (defun list_slice_from_to (lst from to)
34   (list_slice_from (list_slice_to lst to) from)
35 )
36
37 (defun swap-two-element-inner (lst index1 index2)
38   (if (<= 0 index1 index2 (- (my_length lst) 1))
39       (append
40         (list_slice_to lst index1)
41         (cons (my_nth lst index2) Nil)
42         (list_slice_from_to lst index1 index2)
43         (if (= index1 index2) ;защита от дублирования
```

```

44         Nil
45         (cons (my_nth lst index1) Nil)
46     )
47     (list_slice_from lst index2)
48 )
49 (print '(Error in indexes))
50 )
51 )
52
53 (defun swap-two-element (lst index1 index2)
54   (if (< index1 index2)
55       (swap-two-element-inner lst index1 index2)
56       (swap-two-element-inner lst index2 index1)
57   )
58 )
59
60 (swap-two-element '(0 1 2 3 4 5 6 7) 2 4) => (0 1 4 3 2 5 6 7)
61 (swap-two-element '(0 1 2 3 4 5 6 7) 0 7) => (7 1 2 3 4 5 6 0)
62 (swap-two-element '(0 1 2 3 4 5 6 7) -1 2) => (ERROR IN INDEXES)
63 (swap-two-element '(0 1 2 3 4 5 6 7) 1 8) => (ERROR IN INDEXES)
64 (swap-two-element '(0 1 2 3 4 5 6 7) 5 3) => (0 1 2 5 4 3 6 7)
65 (swap-two-element '(0 1 2 3 4 5 6 7) 3 3) => (0 1 2 3 4 5 6 7)

```

**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_list_without_last (lst)
2   (if (null (cdr lst))
3       Nil
4       (cons (car lst) (my_list_without_last (cdr lst))))
5 )
6 )
7
8 (defun my_last (lst)
9   (if (null (cdr lst))
10      (car lst)
11      (my_last (cdr lst))
12   )
13 )
14
15 (defun swap-to-right (lst)
16   (if (null (cdr lst))
17       lst
18       (cons (my_last lst) (my_list_without_last lst))

```

```

19 )
20 )
21
22 (defun swap-to-left (lst)
23   (if (null (cdr lst))
24       lst
25       (cons (cdr lst) (cons (car lst) Nil)))
26 )
27 )
28
29 (swap-to-left ()) => Nil
30 (swap-to-right ()) => Nil
31 (swap-to-left '(1)) => (1)
32 (swap-to-right '(1)) => (1)
33 (swap-to-left '(1 2 (3 4))) => ((2 (3 4)) 1)
34 (swap-to-right '(1 2 (3 4))) => ((3 4) 1 2)

```

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

```

1 (defun my_equalp (x1 x2)
2   (cond
3     ((numberp x1) (and (numberp x2) (= x1 x2)))
4     ((atom x1) (and (atom x2) (eq x1 x2)))
5     ((consp x1) (and (consp x2) (my_equalp (car x1) (car x2)) (
6       my_equalp (cdr x1) (cdr x2))))
7     ((listp x1) (and (listp x2) (= (my_length x1) (my_length x2)) (
8       my_equalp (car x1) (car x2)) (my_equalp (cdr x1) (cdr x2))))
9   )
10 )
11
12 (defun my_member (elem lst)
13   (if (null lst)
14       Nil
15       (if (my_equalp elem (car lst))
16           lst
17           (my_member elem (cdr lst))))
18 )
19
20 (defun update (elem set)
21   (if (my_member elem set)
22       set
23       (cons elem set))
24 )

```

```

24 | )
25 |
26 | (update '(0 1) '((1 2) (2 3) (3 4))) => ((0 1) (1 2) (2 3) (3 4))
27 | (update '(1 2) '((1 2) (2 3) (3 4))) => ((1 2) (2 3) (3 4))
28 | (update '(1 2) Nil) => ((1 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,

```

1 | (defun my_mult (num lst)
2 |   (cons
3 |     (* num (car lst))
4 |     (cdr lst)
5 |   )
6 | )
7 |
8 | (my_mult 2 '(1 2 3)) => (2 2 3)

```

б) элементы списка – любые объекты.

```
1
2 (defun my_mult (num smth)
3   (cond
4     ((null smth) Nil)
5     ((numberp smth) (* num smth))
6     ((atom smth) smth)
7     ; cons
8     (T (let ((res_car (my_mult num (car smth))))
9         (if (my_equalp (car smth) res_car)
10            (cons (car smth) (my_mult num (cdr smth)))
11            (cons res_car (cdr smth)))
12        )
13    )
14  )
15 )
16 )
17
18
19 (my_mult 2 '(1 2 3)) => (2 2 3)
20 (my_mult 2 '(a b c)) => (A B C)
21 (my_mult 2 '(a 2 3)) => (A 4 3)
22 (my_mult 2 '(a (2 . 3) (4 5 6))) => (A (4 . 3) (4 5 6))
23 (my_mult 2 '(a (b . 3) (4 5 6))) => (A (B . 6) (4 5 6))
24 (my_mult 2 '(a (b . c) (d 5 6))) => (A (B . C) (D 10 6))
25 (my_mult 2 '(a (b . c) (d e 6))) => (A (B . C) (D E 12))
26 (my_mult 2 '(a (b . c) (d f (g 5) 10))) => (A (B . C) (D F (G 10) 10))
27 (my_mult 2 Nil) => Nil
```

**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
2 (defun select-between (lst a b)
3   (if (null lst)
4       Nil
5       (if (< a (car lst) b)
6           (cons (car lst) (select-between (cdr lst) a b))
7           (select-between (cdr lst) a b)
8       )
9   )
```

```

10 )
11
12 (select-between '(0 3 7 5 4) 1 6) => (3 5 4)
13
14 (defun my_sort (lst)
15   (if (null lst)
16       Nil
17       (append
18         (my_sort (remove-if (lambda (x) (< (car lst) x)) (cdr lst)))
19         (list (car lst))
20         (my_sort (remove-if (lambda (x) (> (car lst) x)) (cdr lst)))
21       )
22   )
23 )
24
25 (defun select-greater (lst a)
26   (if (null (cdr lst))
27       Nil
28       (if (< a (car lst))
29           lst
30           (select-greater (cdr lst) a)
31       )
32   )
33 )
34
35
36 (defun select-lower (lst b)
37   (if (null (cdr lst))
38       Nil
39       (if (> b (car lst))
40           (cons (car lst) (select-lower (cdr lst) b))
41           Nil
42       )
43   )
44 )
45
46 (defun select-between-sorted (lst a b)
47   (select-greater (select-lower (my_sort lst) b) a)
48 )
49
50 (select-between-sorted '(0 3 7 5 4) 1 6) => (3 4 5)

```

# Литература

- [1] Большакова Елена Игоревна Груздева Надежда Валерьевна. Основы программирования на языке Лисп. М.: Издательский отдел факультета ВМК МГУ имени М.В.Ломоносова (лицензия ИД № 05899 от 24.09.2001); МАКС Пресс, 2010. с. 112.