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 выполнятся 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, ...). conc является структуроразрушающим аналогом append, delete – структуроразрушающим аналогом remove.

2. Отличие в работе функций cons, list, append, nconc 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.

nconc – это структуроразрушающая версия append. Как и append, nconc возвращает соединение своих аргументов, 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-соединения.

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

а) При работе только с верхним уровнем, то есть список '((1 2) 3 (1 2))) считается палиндромом, а список '((1 2) 3 (2 1))) – нет.

```
; с использованием стандартных функций
(defun is_palindrome (lst) (equalp lst (reverse lst)))

; с использованиемм самостоятельно реализованных функций
(defun my_length (lst &optional (len 0))
  (cond
    ((null lst) len)
    ((my_length (cdr lst) (+ len 1)))
  )
)

(defun my_equalp (x1 x2)
  (cond
    ((numberp x1) (and (numberp x2) (= x1 x2)))
    ((atom x1) (and (atom x2) (eq x1 x2)))
    ((consp x1) (and (consp x2) (my_equalp (car x1)  (car x2))
       (my_equalp (cdr x1) (cdr x2))))
    ((listp x1) (and (listp x2) (= (my_length x1) (my_length x2))
       (my_equalp (car x1)  (car x2)) (my_equalp (cdr x1) (cdr x2))))
  )
)


(defun my_rev_upper (lst)
  (if (null lst)
    Nil
    (append
      (my_rev_upper (cdr lst))
      (cons (car lst) Nil))
  )
)

(defun is_palindrome (lst) (my_equalp lst (my_rev_upper lst)))
```

```
;тесты
(is_palindrome '(1 2 3)) => NIL
(is_palindrome '(1 2 1)) => T
(is_palindrome '((1) 2 1)) => NIL

(is_palindrome '((1 2) 3 (1 2))) => T
(is_palindrome '((1 2) 3 (2 1))) => NIL
```

б) При работе по всем уровням, то есть список '((1 2) 3 (1 2))) не считается палиндромом, а список '((1 2) 3 (2 1))) – да.

```
; с использованиемм самостоятельно реализованных функций
(defun my_rev_deeper (lst)
  (if (null lst)
    Nil
    (append
      (my_rev_deeper (cdr lst))
      (if (atom (car lst))
        (cons (car lst) Nil)
        (list (my_rev_deeper (car lst)))
      )
    )
  )
)

(defun is_palindrome (lst) (my_equalp lst (my_rev_deeper lst)))

;тесты
(is_palindrome '(1 2 3)) => NIL
(is_palindrome '(1 2 1)) => T
(is_palindrome '((1) 2 1)) => NIL

(is_palindrome '((1 2) 3 (1 2))) => Nil
(is_palindrome '((1 2) 3 (2 1))) => T
(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

а) Если элементами множеств являются только атомы.

```
; с использованием встроенных функций

(defun set-equal (set1 set2)
  (if (null set1)
    (null set2)
    (if (member (car set1) set2)
      (set-equal (cdr set1) (remove (car set1) set2 :count 1))
      Nil
    )
  )
)
```

Комментарии к решению:

- дополнительный параметр :count 1 в функции remove указан в учебных целях. Так как множество – это неупорядоченная совокупность неповторяющихся элементов, то этот параметр 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.

б) Если элементами множеств могут быть любые S-выражения.

```
; с использованиемм самостоятельно реализованных функций

(defun my_member (elem lst)
  (if (null lst)
    Nil
    (if (my_equalp elem (car lst))
      lst
      (my_member elem (cdr lst)))
  )
)
(defun my_list_without_last (lst)
  (if (null (cdr lst))
    Nil
    (cons (car lst) (my_list_without_last (cdr lst)))
  )
)
```

```
17
18 (defun my_member_before (elem lst)
19   (if (my_member elem lst)
20     (my_member_before elem (my_list_without_last lst))
21     lst
22   )
23 )
24
25 (defun my_remove_first (elem lst)
26   (append
27     (my_member_before elem lst)
28     (cdr (my_member elem lst)))
29 )
30
31 (defun set-equal (set1 set2)
32   (if (null set1)
33     (null set2)
34     (if (my_member (car set1) set2)
35       (set-equal (cdr set1) (my_remove_first (car set1) set2))
36       Nil
37     )
38   )
39 )
40
41 (set-equal '(1 2 3) '(1 2 3)) => T
42 (set-equal '() '()) => T
43 (set-equal '(1 2) '(1 2 3)) => NIL
44 (set-equal '(1) '(1 2)) => NIL
45 (set-equal '(1 2) '(1 (2))) => NIL
46 (set-equal '(1 (2)) '(1 (2))) => T
47 (set-equal '(1 (2)) '(1 (2 3))) => NIL
```

Комментарий к решению: данная функция корректно работает для сравнения как множеств, так и просто неупорядоченных совокупностей ((set-equal '(1 2 2 3) '(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
2  (defun find_capital_by_country (table country)
3    (if (null table)
4      (and (print '(No such country)) Nil)
5      (if (eq (caar table) country)
6        (cdar table)
7        (find_capital_by_country (cdr table) country)
8      )
9    )
10 )
11
12 (defun find_country_by_capital (table capital)
13   (if (null table)
14     (and (print '(No such capital)) Nil)
15     (if (eq (cdar table) capital)
16       (caar table)
17       (find_country_by_capital (cdr table) capital)
18     )
19   )
20 )
21
22 (defvar table)
23 (setq table '((Russia . Moscow) (GreatBritain . London)
                 (France . Paris) (Italy . Roma)))
24
25 (find_capital_by_country table 'Russia) => MOSCOW
26 (find_capital_by_country table 'Italy) => ROMA
27 (find_capital_by_country table 'USA) =>"(NO SUCH COUNTRY)" NIL
28
29 (find_country_by_capital table 'Moscow) => Russia
30 (find_country_by_capital table 'Paris) => FRANCE
31 (find_country_by_capital table 'Washington) =>"(NO SUCH CAPITAL)" NIL
32
33 ; вместо (caar table) можно использовать (row_country (car table))
34 (defun row_country (row_table) (car row_table))
35 ; вместо (cdar table) можно использовать (row_capital (car table))
36 (defun row_capital (row_table) (cdr row_table))
37 ; это сделает программу более гибкой при изменениях в формате таблицы, но медленней
```

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

```
; с использованиемм самостоятельно реализованных функций
(defun my_last (lst)
  (if (null (cdr lst))
    (car lst)
    (my_last (cdr lst))
  )
)

(defun my_list_without_last (lst)
  (if (null (cdr lst))
    Nil
    (cons (car lst) (my_list_without_last (cdr lst)))
  )
)

(defun swap-first-last (lst)
  (if (cdr lst)
    (append
      (cons (my_last lst) Nil)
      (my_list_without_last (cdr lst))
      (cons (car lst) Nil)
    )
    lst
  )
)
(swap-first-last '(1 2 3)) => (3 2 1)
; для 2 приведенных ниже примеров и нужна проверка if (cdr lst) в swap-first-last
(swap-first-last '()) => Nil
(swap-first-last '(1)) => (1)
(swap-first-last '(1 2)) => (2 1)
(swap-first-last '((1 2) 3 4 5)) => (5 3 4 (1 2))

; с использованием встроенных функций (и только если последний элемент – атом)
(defun swap-first-last (lst)
  (if (cdr lst)
    (let ((last_elem (car (last lst))))
      (append
        (cons last_elem (remove last_elem (cdr lst)))
        (cons (car lst) Nil))
    )
    lst
  )
)
```

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

```
; с использованиемм самостоятельно реализованных функций

; индексация начинается с 0
(defun my_nth (lst n)
  (if (or (< n 0) (< (my_length lst) (- n 1)))
    (print '(Error no such index))
    (if (= n 0)
      (car lst)
      (my_nth (cdr lst) (- n 1))
    )
  )
)

; from-элемента в результате нет
(defun list_slice_from (lst from)
  (if (= from -1)
    lst
    (list_slice_from (cdr lst) (- from 1))
  )
)

(defun list_without_last_n (lst n)
  (if (= n 0)
    lst
    (list_without_last_n (my_list_without_last lst) (- n 1))
  )
)

; to-элемента в результате нет
(defun list_slice_to (lst to)
  (list_without_last_n lst (- (my_length lst) to))
)

; элементы to и from не входят в результат
(defun list_slice_from_to (lst from to)
  (list_slice_from (list_slice_to lst to) from)
)
```

```
(defun swap-two-element-inner (lst index1 index2)
  (if (<= 0 index1 index2 (- (my_length lst) 1))
    (append
      (list_slice_to lst index1)
      (cons (my_nth lst index2) Nil)
      (list_slice_from_to lst index1 index2)
      (if (= index1 index2) ;защита от дублирования
        Nil
        (cons (my_nth lst index1) Nil)
      )
      (list_slice_from lst index2)
    )
    (print '(Error in indexes))
  )
)

(defun swap-two-element (lst index1 index2)
  (if (< index1 index2)
    (swap-two-element-inner lst index1 index2)
    (swap-two-element-inner lst index2 index1)
  )
)

(swap-two-element '(0 1 2 3 4 5 6 7) 2 4) => (0 1 4 3 2 5 6 7)
(swap-two-element '(0 1 2 3 4 5 6 7) 0 7) => (7 1 2 3 4 5 6 0)
(swap-two-element '(0 1 2 3 4 5 6 7) -1 2) => (ERROR IN INDEXES)
(swap-two-element '(0 1 2 3 4 5 6 7) 1 8) => (ERROR IN INDEXES)
(swap-two-element '(0 1 2 3 4 5 6 7) 5 3) => (0 1 2 5 4 3 6 7)
(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

```
(defun my_list_without_last (lst)
  (if (null (cdr lst))
    Nil
    (cons (car lst) (my_list_without_last (cdr lst)))
  )
)

(defun my_last (lst)
  (if (null (cdr lst))
    (car lst)
    (my_last (cdr lst))
  )
)

(defun swap-to-right (lst)
  (if (null (cdr lst))
    lst
    (cons (my_last lst) (my_list_without_last lst))
  )
)

(defun swap-to-left (lst)
  (if (null (cdr lst))
    lst
    (cons (cdr lst) (cons (car lst) Nil))
  )
)

(swap-to-left ()) => Nil
(swap-to-right ()) => Nil
(swap-to-left '(1)) => (1)
(swap-to-right '(1)) => (1)
(swap-to-left '(1 2 (3 4))) => ((2 (3 4)) 1)
(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.

```
(defun my_equalp (x1 x2)
  (cond
    ((numberp x1) (and (numberp x2) (= x1 x2)))
    ((atom x1) (and (atom x2) (eq x1 x2)))
    ((consp x1) (and (consp x2)  (my_equalp (car x1)  (car x2)) (
       my_equalp (cdr x1) (cdr x2))))
    ((listp x1) (and (listp x2) (= (my_length x1) (my_length x2)) (
       my_equalp (car x1)  (car x2)) (my_equalp (cdr x1) (cdr x2))))
  )
)

(defun my_member (elem lst)
  (if (null lst)
    Nil
    (if (my_equalp elem (car lst))
      lst
      (my_member elem (cdr lst)))
  )
)

(defun update (elem set)
  (if (my_member elem set)
    set
    (cons elem set)
  )
)

(update '(0 1) '((1 2) (2 3) (3 4))) => ((0 1) (1 2) (2 3) (3 4))
(update '(1 2) '((1 2) (2 3) (3 4))) => ((1 2) (2 3) (3 4))
(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,

```
(defun my_mult (num lst)
  (cons
    (* num (car lst))
    (cdr lst)
  )
)

(my_mult 2 '(1 2 3)) => (2 2 3)
```

б) элементы списка – любые объекты.

```

(defun my_mult (num smth)
  (cond
    ((null smth) Nil)
    ((numberp smth) (* num smth))
    ((atom smth) smth)
    ; cons
    (T (let ((res_car (my_mult num (car smth))))
      (if (my_equalp (car smth) res_car)
        (cons (car smth) (my_mult num (cdr smth)))
        (cons res_car (cdr smth))
      )
       )
    )
  )
)


(my_mult 2 '(1 2 3)) => (2 2 3)
(my_mult 2 '(a b c)) => (A B C)
(my_mult 2 '(a 2 3)) => (A 4 3)
(my_mult 2 '(a (2 . 3) (4 5 6))) => (A (4 . 3) (4 5 6))
(my_mult 2 '(a (b . 3) (4 5 6))) => (A (B . 6) (4 5 6))
(my_mult 2 '(a (b . c) (d 5 6))) => (A (B . C) (D 10 6))
(my_mult 2 '(a (b . c) (d e 6))) => (A (B . C) (D E 12))
(my_mult 2 '(a (b . c) (d f (g 5) 10))) => (A (B . C) (D F (G 10) 10))
(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))

```
; без сортировки
(defun select-between (lst a b)
  (if (null lst)
    Nil
    (if (< a (car lst) b)
      (cons (car lst) (select-between (cdr lst) a b))
      (select-between (cdr lst) a b)
    )
  )
)

(select-between '(0 3 7 5 4) 1 6) => (3 5 4)

; с сортировкой
; блочная (карманная, корзинная) сортировка
(defun my_sort (lst)
  (if (null lst)
    Nil
    (append
      (my_sort (remove-if-not (lambda (x) (> (car lst) x)) (cdr lst)))
      (remove-if-not (lambda (x) (= (car lst) x)) lst)
      (my_sort (remove-if-not (lambda (x) (< (car lst) x)) (cdr lst)))
    )
  )
)

; в функциях select-greater и select-lower предполагается, что переданный список
;отсортирован по возрастанию
(defun select-greater (lst a)
  (if (null lst)
    Nil
    (if (< a (car lst))
      lst
      (select-greater (cdr lst) a)
    )
  )
)


```

```
41 (defun select-lower (lst b)
42   (if (null lst)
43     Nil
44     (if (> b (car lst))
45       (cons (car lst) (select-lower (cdr lst) b))
46       Nil
47     )
48   )
49 )
50
51 (defun select-between-sorted (lst a b)
52   (select-greater (select-lower (my_sort lst) b) a)
53 )
54
55 (select-between-sorted '(0 3 7 5 4) 1 6) => (3 4 5)
```

Литература