Московский государственный технический университет
имени Н. Э. Баумана

Факультет: Информатика и системы управления
Кафедра: Программное обеспечение ЭВМ и информационные технологии
Дисциплина: Функциональное и логическое программирование

Лабораторная работа №5

Выполнили: Никичкин А. С., Фокеев А. И.
Группа: ИУ7–61

Москва, 2015 г.

1 Функция, которая по своему списку-аргументу `1st` определяет является ли он палиндромом (то есть равны ли `1st` и `'(reverse 1st)`)

```
(defun palindromep (lst)
  (equal lst
         (reverse lst)))

(defun problem-1 (lst)
  (mapcar #'palindromep lst))
```

2 Предикат `set-equal`, который возвращает `T`, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения

```
(defun pos-element-in-list (element lst)
  "Return mask where the element in the set"
  (mapcar #'(lambda (x)
              (if (equal element x) 1 0))
          lst))

(defun c-count (element lst)
  (let ((mask (pos-element-in-list element lst)))
    (reduce #'+ mask)))

(defun in-list (element lst)
  "Return T if the element in the list"
  (let* ((bits (pos-element-in-list element lst))
         (i (c-count 1 bits)))
    (declare (fixnum i))
    (< 0 i)))

(defun not-in-list (element lst)
  "Return T if the element not in the list"
  (not (in-list element lst)))

(defun _normalize-set (set)
  "Return new normalized set"
  (let ((result-set `(,(first set))))
    (mapcar #'(lambda (_element)
                (if (not-in-list _element result-set)
                    (rplacd (last result-set) `(,_element))))
            set)
    result-set))

(defun normalize-set (set)
  "Some condition before run normalize-set"
  (cond
    ((null set) nil)
    (t (_normalize-set set))))
```

```
36
37 (defun is-elements-in-set (set1 set2)
38   "Return mask of positions"
39   (mapcar #'(lambda (_element)
40                       (if (in-list _element set2) 1 0))
41           set1))
42
43 (defun _set-equalp (set1 set2)
44   "Internal for set-equalp without validation"
45   (let ((mask (mapcar #'(lambda (_element)
46                           (if (in-list _element set2) 1 0))
47                       set1)))
48     (format t "~a, ~a" (c-count 1 mask) (length set2))
49     (eql (c-count 1 mask) (length set2))))
50
51 (defun set-equalp (set1 set2)
52   "Test on equal of sets"
53   (let* ((normal-set1 (normalize-set set1))
54          (normal-set2 (normalize-set set2))
55          (length-set1 (length normal-set1))
56          (length-set2 (length normal-set2)))
57     (declare (fixnum length-set1 length-set2))
58     (cond
59       ((/= length-set1 length-set2) nil)
60       (t (_set-equalp normal-set1 normal-set2)))))
```

3 Функции, которые обрабатывают таблицу из точечных пар (страна . столица) и возвращают по стране — столицу, а по столице — страну

```
1 (setf test-table '((country1 . city1)
2                    (country2 . city2)
3                    (country3 . city3)
4                    (country4 . city4)
5                    (country5 . city5)))
6
7 (defun get-city (country table)
8   (rest (find-if #'(lambda (x) (equal (first x) country))
9                  table)))
10
11 (defun get-country (city table)
12   (first (find-if #'(lambda (x) (equal (rest x) city))
13                   table)))
```

4 Функция, которая переставляет в списке-аргументе первый и последний элемент

4.1 с использованием `rplaca` и `rplacd`

```
(defun swap-first-last-dl1 (dlst)
  "Swap for dotted-list"
  (let ((t-first (car dlst))
        (t-last (last dlst 0)))
    (cond ((consp dlst) (rplaca dlst t-last)
                        (rplacd (last dlst) t-first)
                        dlst)
          (T Nil))))

(defun swap-first-last-l1 (lst)
  "Swap for list"
  (let ((t-first (car lst))
        (t-last (car (last lst))))
    (cond ((consp lst) (rplaca lst t-last)
                       (rplacd (last lst 2) `(,t-first))
                       lst)
          (T Nil))))

(defun swap-first-last1 (lst)
  "Smart swap"
  (cond ((last lst 0) (swap-first-last-dl lst))
        (T (swap-first-last-l lst))))
```

4.2 с использованием butlast

```
(defun swap-first-last-dl2 (dlst)
  "Swap for dotted-list with copy"
  (let ((left (first dlst))
        (mid (butlast (rest dlst) 0))
        (right (last dlst 0)))
    (if (consp dlst)
      `(,right ,@mid . ,left)
      Nil)))

(defun swap-first-last-l2 (lst)
  "Swap for dotted-list with copy"
  (let ((left (first lst))
        (mid (butlast (rest lst)))
        (right (car (last lst))))
    (if (consp lst)
      `(,right ,@mid ,left)
      Nil)))

(defun swap-first-last2 (lst)
  "Smart swap"
  (cond ((not (consp (rest lst))) `(,(cdr lst) . ,(car lst)))
        ((last lst 0) (swap-first-last-dl2 lst))
        (T (swap-first-last-l2 lst))))
```

4.3 с использованием `remove-if`

```
(defun rm (fn lst &key from-end)
  (funcall fn #'(lambda (x) (or (equal x (car lst))
                                (equal x (car (last lst))))))
           lst
           :count 1
           :from-end from-end))

(defun swap-first-last3 (lst)
  "Swap but dotted-list"
  (cond ((= (length lst) 1) lst)
        (T `(,(car (last lst))
             ,@(t1 #'remove-if (t1 #'remove-if lst) :from-end T)
             ,(first lst)))))
```

5 Функция, которая переставляет в списке-аргументе два указанных своими порядковыми номерами элемента в этом списке

```
(defun swap (lst pos1 pos2)
  "Swap two elements by pos1 and pos2"
  (declare (integer pos1 pos2))
  (let ((x (nth pos1 lst))
        (y (nth pos2 lst))
        (l (length lst)))
    (cond ((and (> l 2)
                (< 0 pos1 l)
                (< 0 pos2 l))
           (rplaca (nthcdr pos1 lst) y)
           (rplaca (nthcdr pos2 lst) x)
           lst)
          (T (format T "Something wrong")))))
```

6 Функции, которые производят круговую перестановку в списке-аргументе влево и вправо

7 Функция, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента

7.1 все элементы списка — числа

7.2 элементы списка — любые числа

8 Функция, которая их списка-аргумента, содержащего только числа, выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка упорядоченного по возрастанию списка чисел