|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 № 8**

|  |  |
| --- | --- |
| **Студент: Зейналов З. Г.**  **Группа: ИУ7-61Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель:** Толпинская Н. Б. |  |

Москва.

2020 г.

**Цель работы**: приобрести навыки работы с управляющими структурами Lisp.

**Задачи работы**: изучить работу функций с произвольным количеством аргументов, функций, разрушающих и не разрушающих структуру исходных аргументов.

**Задание 1**

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

(defun is\_pall(lst)

(cond

(

(null lst)

)

(

(and

(equal

(car lst)

(mapcon #'(**lambda** (el)

(cond

(

(null (cdr el))

(car el)

)

)

) lst)

)

(is\_pall

(mapcon #'(**lambda** (el)

(cond

(

(cdr el)

(cons (car el) nil)

)

)

) (cdr lst))

)

)

)

)

)

**Обозначение переменных**:

* lst – входной список.

Осуществляется проверка на равенство первого и последнего элементов списка. Если они равны, то осуществляется рекурсивный вызов текущей функции для списка-аргумента без первого и последнего аргументов. Иначе возвращается nil.

Условием выхода из рекурсии является достижение середины списка, то есть аргумента nil. В таком случае вернётся T.

(defun is\_pall (lst)

(cond

((null lst))

((reduce #'(**lambda** (x y) (and x y))

(mapcar #'equal lst (reverse lst)))

)

)

)

**Обозначение переменных**:

* lst – входной список.

Осуществляется полный перебор списка-аргумента и его перевёрнутой с помощью reverse копии. С помощью mapcar сравниваются их элементы, а затем с помощью reduce. проверяется, есть ли несовпадающие элементы.

**Тесты:**

1. (is\_pall '(1 2 3 2 1))

T

1. (is\_pall '(1))

T

1. (is\_pall '(1 2))

NIL

1. (is\_pall '((1 2 3) 2 3 2 (1 2 3)))

T

1. (is\_pall '(a c b b c a))

T

**Задание 4**

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

(defun swap-first-last(lst)

(cond

((null (cdr lst)) lst)

((nconc

(mapcon #'(**lambda** (el)

(cond ((null (cdr el)) el))) lst)

(mapcon #'(**lambda** (el)

(cond ((cdr el)(cons (car el) nil)))) (cdr lst))

(cons (car lst) nil)

)

)

)

)

**Обозначение переменных**:

* lst – входной список.

С помощью первого mapcon получается список из последнего элемента списка аргумента. С помощью второго mapcon получается список из первого элемента списка-аргумента. Затем эти списки объединяются с помощью nconc.

**Тесты:**

1. (swap-first-last '(a b c))

(C B A)

1. (swap-first-last '(a b))

(B A)

1. (swap-first-last '((a b c) d r e g (f b) d))

(D D R E G (F B) (A B C))

**Задание 6**

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

(defun swap-to-left (lst k)

(cond

((<= k 0) lst)

((swap-to-left

(nconc

(cdr lst)

(cons (car lst) nil)

)

(- k 1)

)

)

)

)

**Обозначение переменных**:

* lst – входной список;
* k – количество позиций, на которое необходимо выполнить перестановку.

С помощью nconc объединяется хвост списка-аргумента со списком, указатель на голову которого указывает на голову списка-аргумента, а указатель на хвост – на nil. Затем осуществляется рекурсивный вызов текущей функции для созданного списка и второго аргумента, уменьшенного на 1.

**Тесты**:

1. (swap-to-left '(1 2 3 4) 2)

(3 4 1 2)

1. (swap-to-left '(1 2 3 4 5 6) 1)

(2 3 4 5 6 1)

1. (swap-to-left '(1 2 3 (1 5) 3) 2)

(3 (1 5) 3 1 2)

1. (swap-to-left '(1 2 (2 1 5) 3 3) 3)

(3 3 1 2 (2 1 5))

**Обозначение переменных**:

(defun swap-to-right (lst k)

(cond

((<= k 0) lst)

((swap-to-right

(nconc

(mapcon #'(**lambda** (el)

(cond

((null (cdr el)) el))) lst

)

(mapcon #'(**lambda** (el)

(cond

((cdr el)(cons (car el) nil)))) lst

)

)

(- k 1)

)

)

)

)

* lst – входной список;
* k – количество позиций, на которое необходимо выполнить перестановку.

С помощью первого mapcon осуществляется получение списка, указатель на голову которого указывает на последний элемент списка-аргумента, а указатель на хвост – на nil. С помощью второго mapcon осуществляется получение списка-аргумента без последнего элемента. Затем с помощью nconc происходит объединение этих двух списков. Далее осуществляется рекурсивный вызов текущей функции для созданного списка и второго аргумента, уменьшенного на 1. Условием выхода из рекурсии является окончание перестановки элементов, то есть, когда второй аргумент становится равен 0.

**Тесты**:

1. (swap-to-right '(1 2 3 4) 2)

(3 4 1 2)

1. (swap-to-right '(1 2 3 4 5 6) 1)

(6 1 2 3 4 5)

1. (swap-to-right '(1 2 3 4 (1 4) 2) 4)

(3 4 (1 4) 2 1 2)

1. (swap-to-right '(1 2 3 (1 2) 2 4 1) 5)

(3 (1 2) 2 4 1 1 2)

**Задание 7**

Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда:

1. все элементы списка – числа;
2. элементы списка – любые объекты.

(defun mult\_num (lst k)

(mapcar #'(**lambda** (el) (\* el k)) lst)

)

**Обозначение переменных**:

* lst – входной список;
* k – число-множитель.

C помощью mapcar осуществляется формирование списка, состоящего из элементов списка-аргумента, умноженных на число-аргумент.

**Тесты:**

1. (mult\_num '(1 2 3 4) 5)

(5 10 15 20)

1. (mult\_num '(2 1 3) -3)

(-6 -3 -9)

(defun mult\_all (lst k)

(mapcar #'(**lambda** (el)

(cond ((numberp el) (\* el k))

((atom el) el)

((mult\_all el k)))

) lst

)

)

**Обозначение переменных**:

* lst – входной список;
* k – число-множитель.

С помощью mapcar осуществляется проход по всему списку и проверка типа каждого элемента. Если он является числом, то производится умножение на число-аргумент функции и возвращается результат. Если он является атомом, то он возвращается. Если он является списком, то к нему рекурсивно применяется текущая функция.

**Тесты**:

1. (mult\_all '(1 2 3 (1 2 3) 4 3) 3)

(3 6 9 (3 6 9) 12 9)

1. (mult\_all '(1 2 3 (1 2 3) 4 3) -2)

(-2 -4 -6 (-2 -4 -6) -8 -6)

1. (mult\_all '(1 d 3 (1 2 3) f 3) -2)

(-2 D -6 (-2 -4 -6) F -6)

**Задание 8**

Напишите функцию, select-between, которая из списка-аргумента, содержащего только числа, выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка.

(defun select-between(lst a b)

(remove-if #'(**lambda** (x)

(and (or (< x a)(> x b)) (or (> x a)(< x b)))

) lst

)

)

**Обозначение переменных**:

* lst – входной список;
* k – число-множитель.

С помощью функционала remove-if осуществляется проход по всему списку-аргументу и удаление тех элементов списка, которые не входят в заданные границы-аргументы.

**Тесты**:

1. (select-between '(1 2 3 4 5 2) 2 4)

(2 3 4 2)

1. (select-between '(1 2 3 4 5) 1 6)

(1 2 3 4 5)

1. (select-between '(1 2 3 4 5) 6 7)

NIL