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 к лабораторной работе №5

по курсу «Функциональное и логическое программирование»

по теме «Использование управляющих структур, работа со списками»

Студент: Костев Д.И.

Группа: ИУ7-61Б

Преподаватель: Толпинская Н.Б.

2022 г.

Теоретические вопросы

1. Структуроразрушающие и не разрушающие структуру списка функции

Функции, реализующие операции со списками, делятся на две группы:

1. не разрушающие структуру функции; данные функции не меняют переданный им объектаргумент, а создают копию, с которой в дальнейшем производят необходимые преобразования; к таким функциям относятся: append, reverse, last, nth, nthcdr, length, remove, subst и др.

2. структуроразрушающие функции; данные функции меняют сам объект-аргумент, из-за чего теряется возможность работать с исходным списком; чаще всего имя структуроразрушающих функций начинается с префикса -n: nreverse, nconc, nsubst и др.

Обычно в Lisp существуют функции-дубли, которые реализуют одно и то же преобразование, но по разному (с сохранением структуры и без): apppend/nconc, reverse/nreverse и т.д.

2. Отличие в работе функций cons, list, append, nconc и в их результате

Функция cons – чисто математическая, она принимает ровно 2 аргумента, создает бианрный узел и расставляет указатели (car - на первый аргумент, cdr - на второй). В результате работы функции может получится как точечная пара, так и список (зависит от второго аргумента).

Функция list – это форма, она принимает произвольное количество аргументов и создает из них список. В отличии от функции cons, list создает столько бинарных узлов, сколько передано ей аргументов, и связывает их вместе. Результатом работы данной функции всегда будет список.

Функция append также является формой. Она принимает на вход произвольное число аргументов. Для всех аргументов, кроме последнего, эта

функция создает копию, ссылая при этом последний элемент каждого списка аргумента на первый элемент следующего по порядку списка аргумента. В результате работы функции append может получится как список, так и точечная пара (зависит от последнего аргумента).

Таким образом, cons создает один бинарный узел, list создает столько бинарных узлов, сколько передано аргументов, append создает копии всех бинарных узлов для каждого из аргументов, исключая последний аргумент

Практические задания

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

Решение:

```
(defun is_same (lst1 lst2)
	(cond
		((null lst1))
		((eql (car lst1) (car lst2)) (is_same (cdr lst1) (cdr lst2)))
		(T NIL)))

(defun is_palyndrome (lst) (is_same lst (reverse lst)))
```

Результат работы:
(is_palyndrome '(1 2 3 2 1)) → T
(is_palyndrome '(1 2 3 4 5)) → NIL

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

Решение:

```
(defun find_element (el lst)
	(cond
		((null lst) NIL)
		((eql el (car lst)) T)
		(T (find_element el (cdr lst)))))

(defun compare_elements (lst1 lst2)
	(cond
		((null lst1) T)
		((find_element (car lst1) lst2) (compare_elements (cdr lst1) lst2))
		(T NIL)))

(defun set_equal (lst1 lst2) (if (= (length lst1) (length lst2))
	(compare_elements lst1 lst2)))
```

Результат работы:
(set_equal '(1 2 3 4) '(4 1 3 2)) → T
(set_equal '(1 2 3 4) '(4 1 5 2)) → NIL
(set_equal '(1 2 3 4) '(1 2 3 4 5)) → 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

Решение:

```
(defun get_capital (table country)
    (cond
        ((null table) NIL)
        ((eql country (caar table)) (cdar table))
        (T (get_capital (cdr table) country))))

(defun get_country (table capital)
    (cond
        ((null table) NIL)
        ((eql capital (cdar table)) (caar table))
        (T (get_country (cdr table) capital))))
```

Результат работы:

(get_capital '((Russia . Moscow) (France . Paris) (England . London)) 'England) → LONDON

(get_country '((Russia . Moscow) (France . Paris) (England . London)) 'Paris) → FRANCE

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

Решение:

```
(defun put_last_as_first (lst)
    (append (cons (car (reverse lst)) NIL) (reverse (cdr (reverse lst)))))

(defun swap_first_last (lst)
    (append (put_last_as_first (cdr lst)) (cons (car lst) NIL)))
```

Результат работы:

(swap_first_last '(1 2 3 4 5)) → (5 2 3 4 1)

(swap_first_last '(1 2)) → (2 1)

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

Решение:

```
(defun insert_el_end (el lst) (append lst (cons el NIL)))

(defun swap_elements_by_pos (lst pos1 pos2 el1 el2 res ind)
    (cond
        ((null lst) res)
        ((= ind pos1)
        (swap_elements_by_pos (cdr lst) pos1 pos2 el1 el2
        (insert_el_end el2 res) (+ ind 1)))
        ((= ind pos2)
        (swap_elements_by_pos (cdr lst) pos1 pos2 el1 el2
        (insert_el_end el1 res) (+ ind 1)))
        (T (swap_elements_by_pos (cdr lst) pos1 pos2 el1 el2
        (insert_el_end (car lst) res) (+ ind 1)))))

(defun swap_two_elements (lst pos1 pos2)
    (swap_elements_by_pos lst pos1 pos2 (nth pos1 lst) (nth pos2 lst) () 0))
```

Результат работы:
(swap_two_elements '(1 2 3 4 5) 0 3) → (4 2 3 1 5)

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

Решение:

```
(defun swap_to_right (lst)
    (append (cons (car (reverse lst)) NIL) (reverse (cdr (reverse lst)))))

(defun swap_to_left (lst)
    (append (cdr lst) (cons (car lst) NIL)))
```

Результат работы:
(swap_to_right '(1 2 3 4 5)) → (5 1 2 3 4)
(swap_to_left '(1 2 3 4 5)) → (2 3 4 5 1)

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

Решение:

```
(defun check_existance (el lst)
    (cond
        ((null lst) NIL)
        ((and (eql (car el) (caar lst)) (equal (cdr el) (cdar lst)) T))
        (T (check_existance el (cdr lst)))))

(defun add_element (el lst)
    (cond
        ((check_existance el lst) lst)
        (T (cons el lst))))
```

Результат работы:
(add_element '(1 2) '((1 2) (3 4) (5 6))) → ((1 2) (3 4) (5 6))
(add_element '(7 8) '((1 2) (3 4) (5 6))) → ((7 8) (1 2) (3 4) (5 6))

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,

б) элементы списка – любые объекты.

Решение:

а)
```
(defun mult_only_numbers (num lst) (cons (* num (car lst)) (cdr lst)))
```

Результат работы:
(mult_only_numbers 2 '(1 4 6)) → (2 4 6)
(mult_only_numbers 2 '(1)) → (2)

б)
```
(defun mult_various (num lst)
  (cond
    ((numberp (first lst)) (list (* num (first lst)) (second lst) (third lst)))
    ((numberp (second lst)) (list (first lst) (* num (second lst)) (third lst)))
    ((numberp (third lst)) (list (first lst) (second lst) (* num (third lst))))
    (T lst)))
```

Результат работы:
(mult_various 2 '(2 '3 (2 3 4))) → (4 '3 (2 3 4))
(mult_various 2 '((1 2) 3 'bmstu)) → ((1 2) 6 'BMSTU)
(mult_various 2 '(’2 '2 2)) → ('2 '2 4)
(mult_various 2 '((2 3 4) '2 '34)) → ((2 3 4) '2 '34)

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)).

Решение:

```
(defun check_borders (x1 x2 el) (and (> el x1) (< el x2)))

(defun select_between (lst x1 x2)
(cond
    ((null lst) NIL)
    ((check_borders x1 x2 (car lst)) (append (cons (car lst) NIL)
    (select_between (cdr lst) x1 x2)))
    (T (select_between (cdr lst) x1 x2))))
```

Результат работы:
(select_between '(1 2 3 4 5 6) 2 5) → (3 4)
(select_between '(1 2 3 4 5 6) 4 5) → NIL
(select_between '(1 2 3 4 5 6) 0 7) → (1 2 3 4 5 6)