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 №7

По курсу: "Функциональное и Логическое программирование"

Тема Использование управляющих структур, модификация списков.

Группа ИУ7-63Б

Студент Сукочева А.

Преподаватель Толпинская Н.Б.

Преподаватель Строганов Ю. В.

Москва — 2021 г.

Практическая часть

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

```
(defun same-rec (lst1 lst2)
(cond ((null lst1))
    ((= (car lst1) (car lst2)) (same-rec (cdr lst1) (cdr lst2)))
    (T Nil)) )

(defun same (lst1 lst2)
        (if (= (length lst1) (length lst2) )
            (same-rec lst1 lst2)) )

(defun is-palindrome (lst)
    (same lst (reverse lst)))
```

Пример использования:

```
(is-palindrome '(1 2 3 3 2 1))  ;; T
(is-palindrome '(1 2 3 2 1))    ;; T
(is-palindrome '(1 2 3 4))      ;; Nil
```

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

```
(defun find-elem-in-set (set1 elem)
    (cond ((null set1) Nil)
        ((= (car set1) elem) T)
        (T (find-elem-in-set (cdr set1) elem)) ) )

(defun set-equal-rec (set1 set2)
    (cond ((null set1))
        ((find-elem-in-set set2 (car set1)) (set-equal-rec (cdr set1) set2))
        (T Nil)) )

(defun set-equal (set1 set2)
    (if (= (length set1) (length set2))
        (set-equal-rec set1 set2) Nil) )
```

Пример использования:

```
(set-equal '(1 2 3) '(4 5 6)) ;; Nil
(set-equal '(1 2 3) '(3 1 2)) ;; T
```

Задание 3. Напишите необходимые функции, которые обрабатывают таблицу из точечных пар:

(страна.столица), и возвращает по стране - столицу, а по столице - страну.

```
(defun find-capital (table country)
    (cond ((null table) Nil)
        ((eq (caar table) country) (cdar table))
        (T (find-capital (cdr table) country))) )

(defun find-country (table capital)
    (cond ((null table) Nil)
        ((eq (cdar table) capital) (caar table))
        (T (find-country (cdr table) capital))) )
```

Пример использования:

```
(find-capital
    '((Russia . Moscow)
    (Spain . Madrid)
    (France . Paris)) 'Russia) ;; MOSCOW

(find-capital
    '((Russia . Moscow)
    (Spain . Madrid)
    (France . Paris)) 'Hungary) ;; Nil


(find-country
    '((Russia . Moscow)
    (Spain . Madrid)
    (France . Paris)) 'Moscow) ;; RUSSIA

(find-country
    '((Russia . Moscow)
    (Spain . Madrid)
    (France . Paris)) 'Budapest) ;; Nil
```

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

Функция f1 возвращает все, кроме последнего

```
(defun f1 (lst)
    (reverse (cdr (reverse lst))) )
```

Функция swap-first-last добавляет к последнему элементу весь список без последнего элемента, а к нему первый элемент.

```
(defun swap-first-last (lst)
    (append (append (last lst) (cdr (f1 lst))) (cons (car lst) Nil)))
```

Пример использования:

```
(swap-first-last '(1 2 3 4 5)) ;; (5 2 3 4 1)
(swap-first-last '(1 2)) ;; (2 1)
```

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

```
(defun my-length-rec (lst n)
    (cond
    ((null lst) n)
    (T (my-length-rec (cdr lst) (+ n 1)))) )

(defun my-length (lst)
    (my-length-rec lst 0) )
```

Возвращает элемент по индексу.

```
(defun find-by-index-rec (lst index curr-index)
    (cond ((null lst) Nil)
    ((= index curr-index ) (car lst))
    (T (find-by-index-rec (cdr lst) index (+ curr-index 1)))) )

(defun find-by-index (lst index)
    (find-by-index-rec lst index 0))
```

```
(defun swap-two-elements-rec (lst index1 index2 curr-index source-list)
    (cond ((null lst) nil)
        ((= curr-index index1) (cons (find-by-index source-list index2)
           (swap-two-elements-rec (cdr lst) index1 index2 (+ curr-index 1)
           source-list)))
        ((= curr-index index2) (cons (find-by-index source-list index1)
           (swap-two-elements-rec (cdr lst) index1 index2 (+ curr-index 1)
           source-list)))
        (T (cons (car lst) (swap-two-elements-rec (cdr lst) index1 index2 (+
           curr-index 1) source-list) )) ) )
```

```
(defun swap-two-elements (lst i1 i2)
    (cond
    ((>= i1 (my-length lst))
    "The␣first␣index␣is␣larger␣than␣the␣size␣of␣the␣list")
    ((>= i2 (my-length lst))
    "The␣second␣index␣is␣larger␣than␣the␣size␣of␣the␣list")
    ((< i1 0)
    "The␣first␣index␣is␣less␣than␣zero")
    ((< i2 0)
    "The␣second␣index␣is␣less␣than␣zero")
    ((= i1 i2) lst)
    (T (swap-two-elements-rec lst i1 i2 0 lst))) )
```

Пример использования:

```
(swap-two-elements '(11 12 13 14 15) 0 4) ;; (15 12 13 14 11)
(swap-two-elements '(11 12 13 14 15) 4 0) ;; (15 12 13 14 11)
(swap-two-elements '(11 12 13 14 15) 0 0) ;; (11 12 13 14 15)
```

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

Возвращает список без последнего элемента.

```
(defun f1 (lst)
    (reverse (cdr (reverse lst))) )
```

```
(defun f-last (lst)
        (car (reverse lst)) )
```

```
(defun swap-to-right (lst)
    (cons (f-last lst) (f1 lst)))

(defun swap-to-left (lst)
        (append (cdr lst) (cons (car lst) nil)) )
```

Пример использования:

```
(swap-to-right '(1 2 3 4 5 6)) ;; (6 1 2 3 4 5)
(swap-to-left '(1 2 3 4 5 6))  ;;(2 3 4 5 6 1)
```

Теоретическая часть

Способы определения функции

1. С помощью `lambda`.
Базовое определение лямбда-выражения:
(lambda лямбда-список тело_функции)
Пример:

```
(lambda (x y) (+ x y))
```

Применение лямбда-выражений:
(лямбда-выражение фактические_параметры)
Пример:

```
((lambda (x y) (+ x y)) 1 2) ;; 3
```

2. С помощью `defun`. Для неоднократного применения функции (а также для построения рекурсивной функции) используется встроенная функция defun.
Синтаксис:
(defun имя_функции лямбда-список тело_функции)
Пример:

```
(defun sum (x y) (+ x y))
```

Пример вызова:

```
(sum 1 2) ;; 3
```

Варианты и методы модификации элементов списков

Функции для работы со списками делятся на две группы:

1. Не разрушающие структуру. Если сохраняется возможность работать с исходными списками, значит функции не разрушают структуру. (Пример: append, reverse, length, subst ...)
2. Разрушающие структуру. Если после использования какой-то стандартной функции после ее работы теряется возможность работы с теми списками, которые изначально были, значит их структура разрушилась. Чаще всего такие функции начинаются в буквы 'n (Пример: nconc, nreverse, nsubst ...)