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 дисциплине «Функциональное и логическое программирование»

Тема Использование управляющих структур, работа со списками

Студент Романов А.В.

Группа ИУ7-63Б

Оценка (баллы) _____

Преподаватель Толпинская Н.Б., Строганов Ю. В.

Москва — 2021 г.

Задание 1

Постановка задачи

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

Решение

```
(defun my-reverse-internal (lst acc)
  (cond ((null lst) acc)
    (t (my-reverse-internal (cdr lst) (cons (car lst) acc)))))

(defun my-reverse (lst)
  (my-reverse-internal lst ()))

(defun my-equal-internal (lst1 lst2)
  (if (null lst1) T
  (and (= (car lst1) (car lst2)) (my-equal-internal (cdr lst1) (cdr lst2))))
     )

(defun my-equal (lst1 lst2)
  (if (= (length lst1) (length lst2))
    (my-equal-internal lst1 lst2)
    Nil))

(defun is-palindrome (lst)
  (my-equal lst (my-reverse lst)))
```

Задание №2

Постановка задачи

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 my-subsetp (set1 set2)
  (reduce
    #'(lambda (acc1 set1-el)
      (and acc (reduce
        #'(lambda (acc2 set2-el)
          (or acc2 (= set2-el set1-el))) set2 :initial-value Nil)))
  set1 :initial-value T))
```

```
8
9  (defun set-equal (set1 set2)
10   (if (= (length set1) (length set2))
11     (and (my-subsetp set1 set2) (my-subsetp set2 set1))
12     Nil))
```

Задание №3

Постановка задачи

Напишите необходимые функции, которые обрабатывают таблицу из точечных пар: `(страна . столица)`, и возвращают по стране столицу, а по столице — страну

Решение

```
1  (defun get-capital (table country)
2    (cdr (assoc country table)))
3
4  (defun get-country (table capital)
5    (car (rassoc capital table)))
```

Задание №4

Постановка задачи

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

Решение

```
1  (defun rec-add-to-end (lst elem)
2    (cond ((cdr lst)
3      (rec-add-to-end (cdr lst) elem))
4        ((listp elem)
5      (setf (cdr lst) elem))
6        (t (setf (cdr lst) (cons elem Nil))))
7    lst)
8
9  (defun add-to-end (lst elem)
10   (if (null lst)
11     (cons elem Nil)
12     (rec-add-to-end lst elem)))
13
```

```
(defun swap-first-last (lst)
  (add-to-end
    (cons
      (car (reverse lst)) ;; last
      (reverse (cdr (reverse lst)))) ;; tail without last
    (car lst))) ;; head
```

Задание №5

Постановка задачи

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

Решение

```
(defun swap-two-elements-internal (lst i1 i2 v1 v2 acc pos)
  (cond ((null lst) acc)
    ((= pos i1) (swap-two-elements-internal (cdr lst) i1 i2 v1 v2 (add-to-
       end acc v2) (+ pos 1)))
    ((= pos i2) (swap-two-elements-internal (cdr lst) i1 i2 v1 v2 (add-to-
       end acc v1) (+ pos 1)))
    (t (swap-two-elements-internal (cdr lst) i1 i2 v1 v2 (add-to-end acc (
       car lst)) (+ pos 1))))
)

(defun swap-two-elements (lst i1 i2)
  (swap-two-elements-internal lst i1 i2 (nth i1 lst) (nth i2 lst) () 0))
```

Задание №6

Постановка задачи

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

Решение

```
(defun swap-to-left (lst)
  (add-to-end (cdr lst) (car lst)))

(defun swap-to-right (lst)
  (cons
```

```
6     (car (reverse lst))
7     (reverse (cdr (reverse lst)))))
```

Контрольные вопросы

Вопрос 1. Способы определения функций
Ответ. Существует два способа определений функций:

- через `defun`;
- через `lambda`.

Пример `defun`:

```
1 (defun func-name (args-list) function-body)
2 (defun get-cube(y) (* y y y))
3 (get-cube y)
```

Пример `lambda`:

```
1 (lambda (args-list) function-body)
2 ((lambda (x) (* x x)) 2)
```

Вопрос 2. Варианты и методы модификации списков
Ответ. Не разрушающие структуру списка функции. Данные функции не меняют сам объект-аргумент, а создают копию. К таким функциям относятся: `append, reverse, last, nth, nthcdr, length, remove, subst` и прочие.

Структуроразрушающие функции. Данные функции меняют сам объект-аргумент, невозможно вернуться к исходному списку. Чаще всего такие функции начинаются с префикса `n-`. К такми функция относятся: `nreverse, nconc, nsubst` и прочие.

Вопрос 3. Отличие в работе функций `cons`, `list`, `append` и в их результате
Ответ. Функция `cons` — чисто математическая, конструирует списковую ячейку, которая может вовсе и не быть списком. Является списком только в том случае, если вторым аргументом передан список.

Функция `list` — форма, принимает произвольное количество аргументов и конструирует из них список. Результат — всегда список. При нуле аргументов возвращает пустой список.

Функция `append` — форма, принимает на вход произвольное количество аргументов и для всех аргументов, кроме последнего, создает копию, ссылая при этом последний элемент каждого списка-аргумента на первый элемент следующего по порядку списка-аргумента. Копирование для последнего не делается в целях эффективности.