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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Название: Использование управляющих структур, работа со списками

Дисциплина: Функциональное и логическое программирование

Студент	ИУ7-66Б		Т. А. Казаева
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Н. Б. Толпинская
		Подпись, дата	И. О. Фамилия

Москва, 2022 г.

1. Теоретические вопросы

1. *Структуроразрушающие и не разрушающие структуру списка функции.*

Не разрушающие структуру функции не меняют сам объект-аргумент, а создают его копию. Например, reverse, append.

Разрушающие структуру функции меняют объект-аргумент, и получить исходный уже невозможно. Такие функции начинаются с префикса n: nconc, nreverse.

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

a) cons конструирует точечную пару или список, в зависимости от второго аргумента. Является чистой функцией, принимает два аргумента;

b) list является формой, принимает произвольное количество аргументов и составляет из них список. Результатом работы всегда является список.

c) append является формой, принимает произвольное количество аргументов, создает копию для всех, кроме последнего, при этом последний элемент каждого списка-аргумента ссылается на первый элемент следующего по порядку списка-аргумента (или его копию, в зависимости от расположения).

d) nconc возвращает список с элементами из всех списков-аргументов (по порядку). Принцип работы: устанавливает cdr последней ячейки каждого списка в начало следующего списка. Последний аргумент может быть объектом любого типа. Вызванная без аргументов, возвращает nil.

2. Практические задания

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

```
(defun polyp(lst)
       (equal lst (reverse lst)))
```

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 set-equal (set1 set2)
  (and (= (length set1) (length set2))
       (every #'(lambda (elem) (member elem set2 :test #'equal))
        set1)
       (every #'(lambda (elem) (member elem set1 :test #'equal))
        set2)))
```

3. *Напишите свои необходимые функции, которые обрабатывают таблицу из 4-х точечных пар:*
(страна . столица), и возвращают по стране - столицу, а по столице — страну .

```
(defun search-country (captl table)
  (cond ((null table) nil)
    ((eq captl (cdar table)) (caar table))
    (t (search-country captl (cdr table)))))

(defun search-capital (cntry table)
  (cond ((null table) nil)
        ((eq cntry (caar table)) (cdar table))
        (t (search-capital cntry (cdr table)))))

(defun search-country-rassoc (captl table)
  (car (rassoc captl table)))

(defun search-capital-assoc (cntry table)
  (cdr (assoc cntry table)))
```

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

```
(defun swap-first-last (lst)
  (let* ((tmp (reverse (cdr lst)))
                 (mid (reverse (cdr tmp))))
                (cons (car tmp) (append mid (list (car lst))))))
```

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

```
(defun swap-two-elements(lst i j)
  (if (or (> i (length lst)) (> j (length lst)))
    lst
    (let ((fst (nth i lst))
          (snd (nth j lst)))
          (progn (setf (nth i lst) snd) (setf (nth j lst) fst) lst))))
```

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

```
(defun shift-to-left(lst)
  (let* ((tmp (cdr lst))
                 (fnl (car lst)))
                (append tmp (list fnl))))

(defun shift-to-right(lst)
  (let* ((tmp (reverse (cdr (reverse lst))))
         (fnl (car (reverse lst))))
            (cons fnl tmp)))
```

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

```
(defun add-list(src dest)
  (if (some #'(lambda (pair) (equal dest pair)) src)
     src
     (append src (list dest))))
```

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,*

b) *элементы списка – любые объекты.*

```
;; (a)
(defun multiply-lst-num (lst mul)
  (mapcar #'(lambda (elem) (* elem mul)) lst))

;; (b)
(defun multiply-lst (lst mul)
  (mapcar #'(lambda (elem) (cond ((listp elem) (multiply-lst elem mul))
                                 ((numberp elem) (* elem mul))
                                 (t elem)))
    lst))
```

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)).*

```
(defun select-between (lst left right)
  (sort (reduce #'(lambda (res el)
                    (if (and (> el left) (< el right))
                      (cons el res)
                      res)) lst :initial-value ()) #'<))
```