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»

Отчёт по лабораторной работе №7 по курсу «Функциональное и логическое программирование»

Тема: Использование управляющих структур, работа со списками

Студент: Сорокин А. П. ИУ7-66Б

Преподаватели: Толпинская Н. Б.
Строганов Ю. В.

2021 г.

1. Функция, которая по своему списку-аргументу lst определяет, является ли он палиндромом (т. е. равны ли lst и ‘(reverse lst)).

```
(defun my-reverse (lst)
    (funcall
        (defun revers (src res)
            (cond
                ((null src) res)
                (T (revers (cdr src) (cons (car src) res)))
            )
        )
        lst Nil
    )
)

(defun pal (lst rev len)
    (cond
        ((<= len 1) T)
        ((equal (car lst) (car rev))
            (pal (cdr lst) (cdr rev) (- len 2)))
    )
)

(defun palindrom (lst)
    (apply #'pal (list lst (my-reverse lst) (length lst)))
)
```

2. Предикат set-equal, которые возвращает T, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

```
(defun search-in-set (x set)
    (cond
        ((null set) Nil)
        ((equal x (car set)) T)
        (T (search-in-set x (cdr set)))
    )
)

(defun my-subsetp (set1 set2)
    (cond
        ((null set1) T)
        ((search-in-set (car set1) set2) (my-subsetp (cdr set1) set2))
        (T Nil)
    )
)
```

```
(defun set-equal (set1 set2)
    (cond
        ((and (null set1) (null set2)) T)
        ((or (null set1) (null set2)) Nil)
        (T (and (my-subsetp set1 set2) (my-subsetp set2 set1)))
    )
)
```

3. Необходимые функции, которые обрабатывают таблицу из точечных пар: (страна . столица) и возвращают по стране – столицу, а по столице – страну.

```
(defun get-capital (state lst)
    (cond
        ((null lst) Nil)
        ((equal (car (car lst)) state) (cdr (car lst)))
        (T (get-capital state (cdr lst)))
    )
)

(defun get-state (capital lst)
    (cond
        ((null lst) Nil)
        ((equal (cdr (car lst)) capital) (car (car lst)))
        (T (get-state capital (cdr lst)))
    )
)
```

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

```
(defun all-but-last-r (lst res)
    (if (null (cdr lst))
        res
        (all-but-last-r (cdr lst) (append res (cons (car lst) Nil)))
    )
)

(defun all-but-last (lst) (all-but-last-r lst Nil))

(defun swap-first-last (lst)
    (if (null (cdr lst))
        lst
        (append (last lst) (cdr (all-but-last lst)) (cons (car lst)
Nil))
    )
)
```

5. Функция swap-two-element, которая переставляет в списке-аргументе два указателя своими порядковыми номерами элемента в этом списке.

```
(defun my-nth (lst i)
    (cond
        ((null lst) Nil)
        ((= i 0) (car lst))
        (T (my-nth (cdr lst) (- i 1)))
    )
)

(defun cons-left (lst)
    (if (null lst)
        Nil
        (cons (car lst) (cons-left (cdr lst)))
    )
)

(defun swap-two-element-r (lst i j elem-to-place)
    (cond
        ((null lst) Nil)
        ((= i 0)
            (cons elem-to-place
                (if (< j 0)
                    (cons-left (cdr lst))
                    (swap-two-element-r (cdr lst) (- j 1) i (car lst))
                )
            )
        )
        (T (cons (car lst)
            (swap-two-element-r (cdr lst) (- i 1) (- j 1) elem-to-
place))
        )
    )
)

(defun swap-two-element (lst i j)
    (cond
        ((null lst) lst)
        ((= i j) lst)
        ((> i j) (swap-two-element-r lst j i (my-nth lst i)))
        (T (swap-two-element-r lst i j (my-nth lst j)))
    )
)
```

6. Функции swap-to-left и swap-to-right, которые производят круговую перестановку в списке-аргументе влево и вправо, соответственно.

```
(defun swap-to-left (lst)
    (if (null lst)
        lst
        (append (cdr lst) (cons (car lst) Nil))
    )
)

(defun my-last (lst)
    (if (null (cdr lst))
        (car lst)
        (my-last (cdr lst))
    )
)

(defun swap-to-right (lst)
    (if (null lst)
        lst
        (append (cons (my-last lst) Nil) (all-but-last lst))
    )
)
```

<u>Ответы на вопросы</u>

1. Способы определения функций.

- без имени: `(lambda <лямбда-список> (<тело_функции>))`
- с именем: `(defun <имя> <лямбда-список> <тело_функции>)`

<лямбда-список> - список формальных параметров

2. Варианты и методы модификации элементов списка.

При работе со списками есть два варианта их модификации: с разрушением их структур и без их разрушения.

Разрушающие структуру функции не сохраняют возможность работа со старыми структурами, т. к. эти функции изменяют их.

Неразрушающие структуру списков функции сохраняют такую возможность. Если требуется вернуть модифицированный вариант списка-аргумента, то возвращается его изменённая копия.

Неразрушающие: append, reverse, remove, subst; last, nth, nthcdr, length.

Разрушающие: nconc, nreverse, delete, nsubst; rplaca, rplacd.