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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

О Т Ч Е Т

по лабораторной работе № 7

Дисциплина: Функциональное и логическое программирование

Студент	ИУ7-62Б		Е.В. Брянская
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Н.Б.Толпинская
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2021

Лабораторная работа №7

Лаб 5 : 1 - 6

№1 Написать функцию, которая по своему списку-аргументу lst определяет, является ли он палиндромом (то есть равен ли lst и (reverse lst)).

```
(defun my-append (lst1 lst2)
    (cond ((null lst1) lst2)
          ((cons (car lst1) (my-append (cdr lst1) lst2)))
    )
)

(defun my-reverse (lst)
    (cond ((null lst) nil)
          (t my-append (my-reverse (cdr lst)) (lst (car lst)))
    )
)

(defun my-equal (lst1 lst2)
    (cond ((and (atom lst1) (atom lst2)) (eql lst1 lst2))
          ((or (and (atom lst1) (not (atom lst2)))
               (and (not (atom lst1)) (atom lst2))) nil)
          (t (and (my-equal (car lst1) (car lst2))
                  (my-equal (cdr lst1) (cdr lst2)))))
    )
)

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

② (defun is-palindrome-2 (lst)
    (equalp 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.

① (defun compare (lst1 lst2)
    (mapcar #'lambda (x)
        (if (member x lst2 :test #'equal) t nil) lst1)
)

(defun set-equal (lst1 lst2)
    (and (every #'(lambda (x) (eql x t)) (compare lst1 lst2))
        (every #'(lambda (x) (eql x t)) (compare lst2 lst1)))
)

② (defun compare (lst1 lst2)
    (cond ((null lst1) t)
        ((member (car lst1) lst2 :test #'equal)
            (compare (cdr lst1) lst2))
)
)

(defun set-equal (lst1 lst2)
    (and (compare lst1 lst2)
        (compare lst2 lst1))
)

(3) Напишите необходимые ф-ции которые обрабатывают таблицу из точечных пар (страна . столица), и возвращающую по стране - столицу, а по столице - страну

```
(defun my-assoc (key table)
    (cond ((null table) nil)
          ((if (equal key (caar table))
               (cdar table)
               (my-assoc key (cdr table))))
    )
)
(defun my-rassoc (val table)
    (cond ((null table) nil)
          ((if (equal val (cdar table))
               (caar table)
               (my-rassoc val (cdr table))))
    )
)
```

(№4) Напишите ф-цию my-last - last которая представляет в списке - аргументе первый и последний элементы.

```
(defun my-last (lst)
    (cond ((null (cdr lst)) lst)
          (t (my-last (cdr lst))))
)

(defun without-last (lst)
    (cond ((null (cdr lst)) nil)
          ((null (cddr lst)) (cons (car lst) nil))
          (t (append (cons (car lst) nil)
                     (without-last (cdr lst))))
    )
)
```

```
(defun my-last (lst)
  (cond ((null (cdr lst)) lst)
        (t (append (my-last lst)
                   (without-last (cdr lst))
                   (cons (car lst) nil)))
  )
)
```

№5 Напишите ф-цию swap-two-elements которая переставляет в списке-аргументе два заданных своими порядковыми номерами элемента в этом списке.

```
(defun my-nth (n lst)
  (cond ((< n 0) nil)
        ((= n 0) (car lst))
        (t (my-nth (- n 1) (cdr lst)))
  )
)

(defun my-nthset (new n lst)
  (cond ((= n 0) (setf (car lst) new))
        (t (my-nthset new (- n 1) (cdr lst)))
  )
)

(defun swap-two-elements (lst n1 n2)
  (let ((el1 (my-nth n1 lst))
        (el2 (my-nth n2 lst)))
    (if (and el1 el2)
        (and (my-nthset el2 n1 lst)
             (my-nthset el1 n2 lst))
        (print "ERROR: wrong index"))
  )
)
```

(6) Напишите 2 ф-ции swap-to-left и swap-to-right, которые производят круговую перестановку в списке-аргументе влево и вправо соответственно на n позиций.

```lisp
(defun my-nthcdr (lst n)
    (cond ((= n 0) lst)
          (t (my-nthcdr (cdr lst) (- n 1)))
    )
)

(defun first-n-elements (lst n)
    (cond ((= n 0) nil)
          (t (append (cons (car lst) nil)
                     (first-n-elements (cdr lst) (- n 1))))
    )
)

(defun my-length (lst)
    (cond ((null lst) 0)
          (t (+ 1 (my-length (cdr lst))))
    )
)

(defun swap-to-left (lst n)
    (cond ((null lst) nil)
          ((< n 0) (print "ERROR: wrong number (<0)"))
          ((> n (my-length lst)) (swap-to-left lst
                                    (- n (my-length lst))))
          (t (append (my-nthcdr lst n)
                     (first-n-elements lst n)))
    )
)

(defun swap-to-right (lst n)
    (cond ((null lst) nil)
          ((< n 0) (print "ERROR: wrong number (<0)"))
          ((> n (my-length lst)) (swap-to-right lst
                                    (- n (my-length lst))))
```

```
(+ (append (my-nthcdr lst (- (my-length lst) ...
           (first-n-elements lst (- (my-length lst) n))))
  )
)
```

Вопросы

① Способы определения функций

В Lisp можно определять функции без имени с помощью λ-выражений:

(lambda (x1, x2... xn) f(n)) → тело функции

λ-список форм. аргументов

Можно определять и именованные ф-ции с помощью defun

(defun имя λ-список тело)

- имя → символьный атом
- λ-список → список форм. парам.
- тело → определение ф-ции

Применить ф-цию:

(funcall #'имя или λ-выражение факт. пар.)

(apply #'имя или λ-выражение (факт. пар.))

② Варианты и методы модификации эл-ов списка

Поставить значение в соответствие с символьным атомом можно с помощью:

(setf имя значение)

(setq имя значение)

Изменить car и cdr можно с помощью функций rplaca и rplacd соответственно. Эти функции переставляют указатели на аргумент, который подается вторым.

С помощью ф-ций (subst new old lst) и (nsubst new old lst) можно заменить соответствующие старые значения на новые в списке lst. Причем subst не разрушает структуру в отличии от nsubst.

Можно использовать функционалы:

а) применяющие apply, funcall

б) отображающие mapcar, maplist, mapcan и т.д.