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»

Лабораторная работа № 9

По дисциплине «Функциональное и логическое программирование»

Студент: Тимонин А. С.

Группа ИУ7-62б

Преподаватель Толпинская Н. Б.

Москва.
2020 г.

Практическая часть

Задание 2.

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

```
(defun eqLst (lst1 lst2)
    (and (subsetp lst1 lst2) (subsetp lst2 lst1))
)
; lst1, lst2 – списки
; (eqLst (list 1 2) (list 2 1)) -> T

(defun checkEl (elm lst)
    (cond
        ((equal elm (car lst)) t)
        ((equal nil (car lst)) nil))
        (t (checkEl elm (cdr lst))
    )
)
; elm – елемент из списка, lst – список
; (checkEl 5 (list 5 1)) -> T


(defun checkLst (lst1 lst2)
    (if (equal nil (car lst1))
        t
        (if (not (checkEl (car lst1) lst2))
            nil
            (checkLst (cdr lst1) lst2)
        )
    )
)
```

```
; lst1, lst2 – списки
; (checkLst (list 1 2) (list 3 5 1)) -> T

(defun set-equal (lst1 lst2)
(
    and
        (checkLst lst1 lst2)
        (checkLst lst2 lst1))
)
; lst1, lst2 – списки
; (set-equal (list 1 2) (list 2 1)) -> T
```

Задание 3.

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

```
(defun createPairs(lst1 lst2)
    (list (cons (car lst1) (car lst2))
          (cons (cadr lst1) (cadr lst2))
          (cons (caddr lst1) (caddr lst2))
          (cons (cadddr lst1) (cadddr lst2))
    )
)
; lst1, lst2 – списки

(setq lst1 '(Rus Ukr Blr Kaz))
(setq lst2 '(Moscow Kiev Minsk Astana))
```

```
(setq pair1 (createPairs lst1 lst2)) -> список из точечных пар страна-столица -> ((RUS . MOSCOW)(UKR . KIEV)…)

(defun searchEl (pairs element)
    (cond
        ((null pairs) Nil)
        ( (equal (caar pairs) element) (cdar pairs) )
        ( (equal (cdar pairs) element) (caar pairs) )
        (t (searchEl (cdr pairs) element))
    )
)
; searchEl – ищет либо страну, либо столицу в списке, одно найдет, другое выдаст. Нашло столицу, ответ будет страна
; pairs – список из точечных пар, element – искомый в списке элемент
; (searcEl pair1 'RUS) -> MOSCOW
```

Задание 7.

Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда

A. все элементы списка --- числа,

B. элементы списка -- любые объекты.

```
; a) все элементы списка --- числа,

(defun multNum1 (lst num)
    (mapcar #'(lambda (x) (* x num))
        lst
    )
)
```

```
; lst - список, num - множитель
; (multNum1 (list 1 2 3) 10) -> (10 20 30)

(defun multNum2 (lst number)
    (cons
        (if (not (null lst))
            (* (car lst) number)
        )

        (cond
            ((< (length (cdr lst)) 1) Nil)
            (t (multNum (cdr lst) number))
        )
    )
)
; lst - список, number - множитель
; (multNum2 (list 1 2 3) 10) -> (10 20 30)

; 6) элементы списка -- любые объекты.

(defun multAll (lst num)
    (mapcar
        #'(lambda (x)
                (cond
                    ((numberp x) (* x num))
                    ((listp x) (multAll x num))
                    (t x)
                )
          ) lst

    )
)
```

```
; lst – список, num – множитель
; (multAll (list 1 2 'a (list 5 'b 6)) 10) -> (10 20 A (50
B 60))
```

Задание 2.

Напишите функцию, которая уменьшает на 10 все числа из списка аргумента этой функции.

```
(defun allMinus (lst)
    (mapcar
        #'(lambda (x)
                (cond
                    ((numberp x) (- x 10))
                    ((listp x) (allMinus x))
                    (t x)
                )
          ) lst
    )
)
; lst – список
; (allMinus (list 1 2 3)) -> (-9 -8 -7)
```

Задание 3.

Написать функцию, которая возвращает первый аргумент списка -аргумента. который сам является непустым списком.

```
(defun firstList (lst)
    (cond
        ((null (cdr lst)) nil)
        ((and (listp (car lst)) (> (length (car lst)) 0))
(car lst))
```

```
        (t (firstList (cdr lst)))
    )
)
; lst – список
; (firstList (list 1 2 5 () (list 'k) (list 'ek))) -> (K)

(defun firstList2 (lst)
    (mapcar
        #'(lambda (x)
                (cond
                    ( (and (listp x) (> (length x) 0)) x )
                    ; (t x)
                )
          ) lst
    )
)
; lst – список
; (firstList2 (list 1 2 5 () (list 'k) (list 'ek))) -> (K)
```

Задание 4.

Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10. (Вариант: между двумя заданными границами.)

```
(defun selectBetweenInner (lst left right result)
(mapcar #'(lambda (x)
    (cond   ((listp x) (selectBetweenInner x left
right result))
        ((and   (numberp x) (> x left) (< x right))
            (nconc result (cons x nil))
            )
```

```
        )
    )
    lst
)
(cdr result)
)
; lst – список, left, right – границы левая и правая,
result – результирующие списковые ячейки
; (selectBetweenInner (list 1 2 3 4 5 -5 -6 -7 -8) 0 3
(cons nil nil)) -> (1 2)

(defun selectBetween (lst left right);
    (selectBetweenInner lst left right (cons nil nil))
)
; lst – список, left, right – границы левая и правая
; (selectBetween (list 1 2 3 4 5 -5 -6 -7 -8) 0 3) ->
(1 2)
```

Задание 5.

Написать функцию, вычисляющую декартово произведение двух своих списков- аргументов. (Напомним, что A x B это множество всевозможных пар (a b), где a принадлежит A, принадлежит B.)

```
(defun decart (a b)
    (mapcan #'
        (lambda (x)
            (mapcar #'
                (lambda (y) (list x y))
                a
            )
```

```
        )
        b
    )
)
; a, b – списки
; (decart (list 1 2) (list 'a ‘b)) -> ((A 1) (A 2) (B 1) (B 2) (C 1) (C 2))
```

Задание 6.

Почему так реализовано reduce?

```
(reduce #*+0) -> 0 ; Ошибка
(reduce #*+ ()) -> 0 ; Вернет изначальное значение сложения 0
```

Теоретическая часть

Способы организации повторных вычислений в Lisp.

1. Использование функционалов;
2. Использование рекурсии.

Различные способы использования функционалов.

Функционалы:

1. Применяющие – однократное применение функции, являющейся аргументом, к остальным аргументам;
2. Отображающие – многократное применение функции, являющейся аргументом, к остальным аргументам по верхнему уровню.

Что такое рекурсия?

Рекурсия – это ссылка на определяемый объект во время его определения. Рекурсия в Lisp - естественный принцип обработки списков.

Способы организации рекурсивных функций.

1. Хвостовая рекурсия;
2. Рекурсия по нескольким параметрам;
3. Дополняемая рекурсия;
4. Множественная рекурсия.

Способы повышения эффективности реализации рекурсии.

В целях повышения эффективности рекурсивных функций рекомендуется
формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии. Это и есть хвостовая рекурсия.

Для превращения не хвостовой рекурсии в хвостовую и в целях формирования результата (результирующего списка) на входе в рекурсию рекомендуется использовать дополнительные (рабочие) параметры. При этом становится необходимым создать функцию-оболочку (как в задании 4) для реализации очевидного обращения к функции.