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
Использование функционалов и рекурсии

Студент Лучина Е.Д

Группа ИУ7-61Б

Преподаватель Толпинская Н.Б.

Москва.

2020 г.

5.2) Написать предикат set_equal, который возвращает t, если два его множество-

аргумента содержат одни и те же элементы, порядок которых не имеет значения.

Считается, что на вход функции set_equal передаются два списка, являющиеся по смыслу неупорядоченными множествами (то есть списки не содержат дубликаты элементов и порядок элементов не имеет значения).

При реализации используется функция my_unsorted_equal, которая принимает два аргумента, являющихся любой структурой, и возвращает t - если они равны, иначе nil. (используется в качестве предиката сравнения в функциях member и remove ниже)

```
(defun my_unsorted_equal (a b)
    (cond
        ((not (listp a)) (and (not (listp b)) (eq a b)))
        ((not (listp b)) nil)
        (t (set_equal a b))
    )
)
```

1. Элемент a не является списком
    a. Элемент b не является списком - результат сравнения
    b. Иначе не равны
2. Элемент b не является списком (элемент b является списком) - не равны
3. Сравнить как множества

```
(defun set_equal (set1 set2)
     (cond ((null set1) (null set2))
           ((null set2) nil)
           ((not (member (car set1) set2 :test #'my_unsorted_equal)) nil)
           (t (set_equal (cdr set1) (remove (car set1) set2
                                            :test #'my_unsorted_equal)))
     )
)
```

1. Первый список пуст
    a. Второй список пуст - равны
    b. Иначе - не равны
2. Второй список пуст (первый не пуст) - не равны
3. Первый элемент set1 не является членом set2 (оба списка не пусты) - не равны
4. (оба списка не пустые, первый элемент является членом второго) - рекурсивный вызов для хвоста set1 и set2 \ элемент.

Тесты:

- `(set_equal () ()) - T`
- `(set_equal () '(1)) - Nil`
- `(set_equal '(1) ()) - Nil`
- `(set_equal '(1 2 3) '(1 2 3 4)) - Nil`
- `(set_equal '(1 2 3 4) '(1 2 3)) - Nil`
- `(set_equal '(1 2 3) '(1 2 3)) - T`
- `(set_equal '(1 2 3) '(1 3 2)) - T`
- `(set_equal '(1 2 3 5) '(5 1 3 2)) - T`
- `(set_equal () '(())) - Nil`

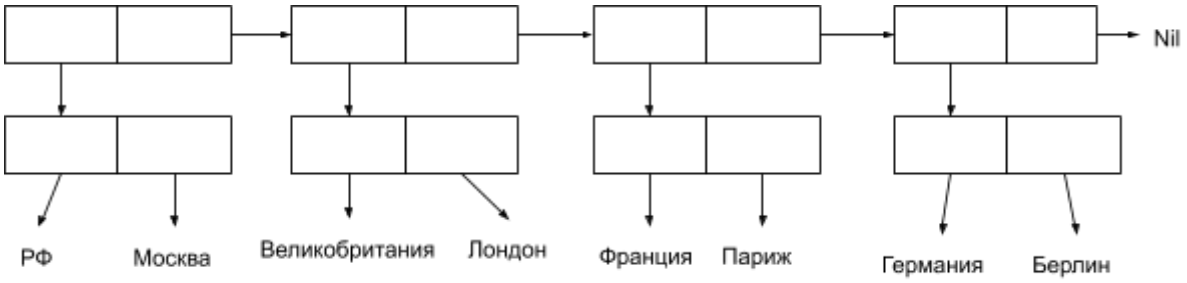
- **(set_equal '(()) ()) - Nil**
- **(set_equal '(1 (2 3) 5) '(5 1 (2 3))) - T**
- **(set_equal '(1 (2 3 (1)) 5) '(5 1 (2 3 (1)))) - T**
- **(set_equal '(1 (2 3 (1)) 5) '(5 1 (2 (1) 3))) - T**

*попытка применения функционалов (сравнивает только одноуровневые числовые множества - иначе не работает сортировка)

проверяет равенство длин; сортирует; соединяет соответствующие друг другу элементы в точечные пары - получается список точечных пар; проверяет каждую пару на равенство двух значений - формирует список из T и Nil; далее с помощью reduce и логического умножения получает результат

```
(defun set_equal2 (lst1 lst2)
      (if (eq (length lst1) (length lst2))
        (reduce (lambda (x y) (and x y))
                (mapcar (lambda (pair) (= (car pair) (cdr pair)))
                        (mapcar #'cons (sort lst1 #'<) (sort lst2 #'<))))
        nil
      )
)
```

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



Считается что аргумент функции гарантировано является списком точечных пар вида (страна. столица).

возвращают по стране столицу	по столице - страну
(defun fcity (lst country) ** (if lst** ** (if (eq (caar lst) country)** ** (cdar lst)** ** (fcity (cdr lst) country)** **)** **)** **)**	**(defun fcountry (lst city)** ** (if lst** ** (if (eq (cdar lst) city)** ** (caar lst)** ** (fcountry (cdr lst) city)** **)** **)** **)**
(defun fcity2 (lst country) ** (if (not (null lst))** ** (reduce** ** (lambda (x y) (or x y))**	**(defun fcountry2 (lst city)** ** (if (not (null lst))** ** (reduce** ** (lambda (x y) (or x y))**

```
    (mapcar (lambda (pair)
      (if (eq (car pair) country)
        (cdr pair)
        nil
      ))
    lst)
  )
 )
)
```

```
    (mapcar (lambda (pair)
      (if (eq (cdr pair) city)
        (car pair)
        nil
      ))
    lst)
  )
 )
)
```

```
(setq lofpairs (list (cons 'Russia 'Moscow) (cons 'France 'Paris) (cons
'Germany 'Berlin) (cons 'UK 'London)))
```

```
(print (fcity lofpairs 'France)) -> Paris
```

```
(print (fcountry lofpairs 'Berlin)) -> Germany
```

Если переданный элемент не найден - результатом функции будет nil

5.7) Напишите функцию, которая умножает на заданное число-аргумент все числа

из заданного списка-аргумента, когда

а) все элементы списка --- числа

функционал	Дополняемая рекурсия

(defun func (lst a)
  (mapcar
  (lambda (x) (* a x))
  lst)
)
	(defun r_func (lst a)
	  (cond ((null lst) nil)
	  (t
	  (cons (* a (car lst))
	  (r_func (cdr lst) a)
	  )
	  )
	  )
	)

```
(func '(1 2 3) 2) -> (2 4 6)
```

б) элементы списка -- любые объекты.

функционал	Дополняемая рекурсия

(defun func2 (lst a)
  (mapcar
  (lambda (x)
  (if (numberp x)
  (* a x)
	(defun r_func2 (lst a)
	  (if (not (null lst))
	  (cons
	  (if (numberp (car lst))
	  (* a (car lst))

```
      x
    )
  )
  lst
 )
)
```

```
            (car lst)
        )
        (r_func2 (cdr lst) a)
    )
  )
)
```

(func2 '(1 d (3 4) 3) 2) -> (1 d (3 4) 6)

6.2) Напишите функцию, которая уменьшает на 10 все числа из списка

аргумента этой функции.

функционал	рекурсия

```
(defun func3 (lst)
  (mapcar
    (lambda (x)
      (if (numberp x)
        (- x 10)
        x
      )
    )
  lst)
)
```

```
(defun r_func3 (lst)
  (if (not (null lst))
     (cons
        (if (numberp (car lst))
             (- (car lst) 10)
             (car lst)
        )
        (r_func3 (cdr lst))
     )
  )
)
```

(func3 '(10 d (3 4) 5)) -> (0 d (3 4) -5)

6.3) Написать функцию, которая возвращает первый аргумент списка -аргумента,

который сам является непустым списком.

Рекурсия

```
(defun first_list (lst)
    (cond
        ((null lst) nil)
        ((and (listp (car lst)) (not (null (car lst)))) (car lst))
        (t (first_list (cdr lst)))
    )
)
```

Функционал

```
(defun f_first (lst)
      (reduce (lambda (x y) (or x y))
            (mapcar
                  (lambda (elem)
                  (if (and (listp elem) (not (null elem))) elem nil)
```

```
            lst)
        )
    )

    (first_list '(1 2 () (4 3) (2 3) 4 3)) -> (3 4)
```

6.4) Написать функцию, которая выбирает из заданного списка только те числа,

которые больше 1 и меньше 10. (Вариант: между двумя заданными границами.)

```
(defun filter (lst a b)
  (cond
    ((null lst) nil)
    ((and (numberp (car lst))
          (> (car lst) a)
          (< (car lst) b)
      )
      (cons (car lst)(filter (cdr lst) a b)))
    (t (filter (cdr lst) a b))
  )
)

(filter '(1 4 (2) 3 d 5 7) 0 4) -> (1 3)
```

6.5) 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 (X Y)

  (mapcan #'(lambda (x) (mapcar #'(lambda (y) (list x y)) Y)) X)

)
```

рекурсия:

```
(defun r_decart (x Y)
    (cond
        ((null Y) nil)
        ( t (append
                (list (list x (car Y)))
                (r_decart x (cdr Y))
            )
        )
    )
)
```

```
(defun r_decart_top (X Y)
    (cond
        ((null X) nil)
        ((null Y) nil)
        ( t (append
                (r_decart (car X) Y)
                (r_decart_top (cdr X) Y)
            )
        )
    )
)
```

```
(r_decart_top ‘(1 2) ‘(3 4 5)) -> ((1 3) (1 4) (1 5) (2 3) (2 4) (2 5))
```

6.6) Почему так реализовано reduce, в чем причина?

(reduce #'+ ()) -> 0

(reduce #'* ()) -> 1

Сначала функция проверяет список-аргумент. Если он пуст, возвращается значение функции при отсутствии аргументов.

Также reduce использует аргумент :initial-value. Этот аргумент определяет значение, к которому будет применена функция при обработке первого элемента списка-аргумента. Если список-аргумент пуст, то будет возвращено значение initial-value.

Теоретические вопросы:

· Способы организации повторных вычислений в Lisp

- Использование функционалов (функция, которая принимает другую функцию в качестве параметра)
- Использование рекурсии (ссылка на себя)

· Различные способы использования функционалов

В Lisp используются применяющие и отображающие функционалы, функционалы, являющиеся предикатами, функционалы, использующие предикаты в качестве функционального объекта. К тому же функция-параметр может также использовать функционалы или (и) быть рекурсивной.

· Что такое рекурсия? Способы организации рекурсивных функций

Рекурсия — это ссылка на определяемый объект во время его определения.

Существуют следующие типы рекурсивных функций:

- Хвостовая рекурсия
- Дополняемая рекурсия
- Множественная рекурсия
- взаимная рекурсия
- рекурсия более высокого порядка

· Способы повышения эффективности реализации рекурсии

В целях повышения эффективности рекурсивных функций используется хвостовая рекурсия, суть которой в формировании результата не на выходе из рекурсии, а на входе в рекурсию, выполнении всех действий до ухода на следующий шаг рекурсии.

Преобразование не хвостовой рекурсии в хвостовую возможно путем использования дополнительных параметров. В этом случае необходимо использовать функцию-оболочку для запуска рекурсивной функции с начальными значениями дополнительных параметров.