|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 предмету: «Функциональное и логическое программирование»*

**Тема:** Определение функций пользователя.

Студент: Лаврова А. А.,

Группа: ИУ7-65Б

Преподаватель: Толпинская Н. Б.

Москва, 2020 г.

№2 (Л/р 5)

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

(defun set-equal-rec (set1 set2)

(cond

((null set1) t)

((member (car set1) set2 :test 'equal) (set-equal (cdr set1) set2))

(t nil)

)

)

(defun set-equal (set1 set2)

(and

(eval (cons 'and (mapcar #'(lambda(x) (not (null x))) (mapcar #'(lambda (x) (member x set2 :test 'equal)) set1))))

(eval (cons 'and (mapcar #'(lambda(x) (not (null x))) (mapcar #'(lambda (x) (member x set1 :test 'equal)) set2))))

)

)

Пример:

> (set-equal ‘(1 2 3) ‘(2 3 1))

T

> (set-equal ‘(1 2 3) ‘(3 4 5))

NIL

> (set-equal-rec ‘(1 2 3) ‘(2 3 1))

T

> (set-equal-rec ‘(1 2 3) ‘(3 4 5))

NIL

№3 (Л/р 5)

Напишите необходимые функции, которые обрабатывают таблицу из точечных пар:

(страна. столица), и возвращают по стране - столицу, а по столице - страну.

(list (cons 'Argentina 'Buenos-Aires) (cons 'Russia 'Moscow) (cons 'Italy 'Rome) (cons 'France 'Paris))

(defun check (pair val)

(cond ((equal (car pair) val) (cdr pair))

((equal (cdr pair) val) (car pair))))

(defun generate-check (val) (lambda (pair) (check pair val)))

(defun find-in-table (base val)

(find-if #'(lambda (x) (not (eq x Nil)))

(mapcar (generate-check val) base)))

Пример:

> (find-in-table (list (cons 'Argentina 'Buenos-Aires) (cons 'Russia 'Moscow) (cons 'Italy 'Rome) (cons 'France 'Paris)) 'Paris)

FRANCE

> (find-in-table (list (cons 'Argentina 'Buenos-Aires) (cons 'Russia 'Moscow) (cons 'Italy 'Rome) (cons 'France 'Paris)) 'Argentina)

BUENOS-AIRES

№7 (Л/р 5)

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

a) все элементы списка --- числа,

б) элементы списка -- любые объекты.

А) (defun mult\_num (lst mul)

(mapcar #'(lambda (x) (\* x mul))

lst

)

)

(defun mult\_num\_rec (lst mul)

(cond ((equal (cdr lst) Nil) (cons (\* mul (car lst)) Nil))

(T (cons (\* mul (car lst))( mult\_num\_rec mul (cdr lst))))

)

)

Пример:

> (mult\_num ‘(1 2 3 4 5) 10)

(10 20 30 40 50)

> (mult\_num\_rec ‘(1 2 3 4 5) 10)

(10 20 30 40 50)

Б) ((defun mult\_obj (lst mul)

(mapcar #'(lambda (x) (if (numberp x) (\* x mul) x)) lst)

)

Пример:

> (mult\_obj ‘(1 2 3 4) 10)

(10 20 30 40)

> (mult\_obj ‘(1 (2 3) 4) 10)

(10 (20 30) 40)

> (mult\_obj ‘(1 (2) a) 10)

(10 (20) a)

№2 (Л/р 6)

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

(defun minusten(lst)

( mapcar #'(lambda(x) (- x 10) ) lst

)

)

(defun minusten\_rec(lst)

(cond ((null lst) nil)

(t (cons (- (car lst) 10) (minusten\_rec (cdr lst)))))

)

Пример:  
> (minusten ‘(10 100 89))

(0 90 79)

> (minusten\_rec ‘(10 100 89))

(0 90 79)

№3 (Л/р 6)

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

(defun sublist\_rec(lst)

(if (listp (car lst)) (car lst) (sublist\_rec(cdr lst)) )

)

(defun sublist(lst)

(car (remove nil (mapcar #'(lambda (x) (if (listp x) (if (null x) nil x) nil)) lst)))

)

Пример:

> (sublist\_rec ‘(1 2 3 4))

NIL

> (sublist\_rec ‘(1 2 (3 4 5) 6))

(3 4 5)

> (sublist ‘(1 2 3 4))

NIL

> (sublist ‘(1 2 (3 4 5) 6))

(3 4 5)

№4 (Л/р 6)

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

которые больше 1 и меньше 10.

(Вариант: между двумя заданными границами. )

(defun select\_between\_inner (lst left right result)

(mapcar #'(lambda (x)

(cond ((listp x) (select\_between\_inner x left right result))

((and (numberp x) (> x left) (< x right))

(nconc result (cons x nil))

)

)

)

lst

)

(cdr result)

)

(defun select\_between (lst left right)

(select\_between\_inner lst left right (cons nil nil)))

Пример:  
>(select\_between ‘(1 2 3 4 5) 2 4)

(3)

> (select\_between ‘(1 2 3 4 5) 3 4)

NIL

№5 (Л/р 6)

Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов. ( Напомним, что А х В это множество всевозможных пар (a b), где а принадлежит А, принадлежит В.)

(defun dec(lstx lsty)

(mapcar #'(lambda (x)

(

mapcar #'(lambda (y) (list x y)) lsty

)

) lstx)

)

Пример:

> (dec '(A B) '(B A))

(((A B) (A A)) ((B B) (B A)))

> (dec '(1 2) '(B A))

(((1 B) (1 A)) ((2 B) (2 A)))

№6 (Л/р 6)

Почему так реализовано reduce, в чем причина?

(reduce #’+ ()) -> 0

(reduce #’\* ()) -> 1

Ответ: reduce работает с аргументом initial-value. Он определяет значение, к которому будет применена функция при обработке первого элемента списка-аргумента. Однако если список пуст, то будет возвращено значение initial-value, которое в зависимости от операции, вернет нейтральный элемент.

**Теоретическая часть**

1. Способы организации повторных вычислений в Lisp

Функционал – функция, аргументом которого является функция.

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

1. Различные способы использования функционалов

Функционалы используются для реализации повторных вычислений или для обработки списковых таблиц.

Классификация функционалов:

* Применяющие
* Отображающие

1. Что такое рекурсия? Способы организации рекурсивных функций  
   Рекурсия — это ссылка на определяемый объект во время его  
   определения. Виды рекурсии: хвостовая, дополняемая, множественная,  
   взаимная и рекурсии высокого порядка.
2. Способы повышения эффективности реализации рекурсии  
   Организация хвостовой рекурсии повышает эффективность, однако для этого может потребоваться использование дополнительных параметров. Такая рекурсия может быть заменена на итерацию.