|  |  |
| --- | --- |
| 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 курсу “Функциональное и логическое программирование”**

**по теме “Функционалы”**

|  |  |
| --- | --- |
| Студент: | Уласик Е.А. |
| Группа: | ИУ7-61 |
| Преподаватель: | Толпинская Н.Б. |

*2020 г.*Практическая часть

Задание 1. Написать предикат set-equal, который возвращает T, если два его множество-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

**Через функционалы:**

(defun set-equal (lst1 lst2)

(and

(eval (cons 'and (mapcar #'(lambda(x) (not (null x)))

(mapcar #'(lambda (x) (member x lst2 :test 'equal)) lst1))))

(eval (cons 'and (mapcar #'(lambda(x) (not (null x)))

(mapcar #'(lambda (x) (member x lst1 :test 'equal)) lst2))))

)

)

Описание параметров:

lst1 – первый список;

lst2 – второй список;

Тесты:

1. (set-equal '(1 2) '(2 1))

T

1. (set-equal '(1 2) '(4 1 2))

NIL

1. (set-equal '(1 2 3) '(3 2 1))

T

1. (set-equal '(1) '(1))

T

1. (set-equal '(1 2) '(1 2 5))

NIL

**Через рекурсию:**

(defun in (lst1 lst2)

(cond

((null lst1) t)

((member (car lst1) lst2 :test 'equal) (in(cdr lst1) lst2))

(t nil)

)

)

(defun set-equal (lst1 lst2)

(and (in lst1 lst2) (in lst2 lst1))

)

Описание параметров:

lst1 – первый список-множество;

lst2 – второй список;

Тесты:

1. (set-equal '(1 2) '(2 1))

T

1. (set-equal '(1 2) '(4 1 2))

NIL

1. (set-equal-recur '(1 2 3) '(3 2 1))

T

1. (set-equal '(1 2) '(1 2 5))

NIL

1. (set-equal '() '())

T

Задание 2. Написать необходимые функции, которые обрабатывают таблицу из точечных пар (страна . столица) и возвращает по стране – столицу, а по столице – страну.

**Через функционалы:**

Поиск по первым элементам точечных пар:

(defun find-first (el lst)

(car (remove nil (mapcar #'(lambda(item)

(cond ((equal (car item) el) item) (T nil))) lst)))

)

Поиск по вторым элементам точечных пар:

(defun find-second (el lst)

(car (remove nil (mapcar #'(lambda(item)

(cond ((equal (cdr item) el) item) (T nil))) lst)))

)

Поиск заданного элемента el в списке точечных пар:

(defun find-in-table (el lst)

(or (cdr(find-first el lst))

(car(find-second el lst))

)

)

Описание параметров:

el – название страны или столицы, то есть искомый элемент;

lst – список точечных пар

Тесты:

1. (find-in-table 'Moscow '((Japan . Tokio) (Russia . Moscow) (Poland . Warsaw)))

RUSSIA

1. (find-in-table 'Russia '((Japan . Tokio) (Russia . Moscow) (Poland . Warsaw)))

MOSCOW

1. (find-in-table 'Warsaw '((Japan . Tokio) (Russia . Moscow) (Poland . Warsaw)))

POLAND

1. (find-in-table 'Poland '((Japan . Tokio) (Russia . Moscow) (Poland . Warsaw)))

WARSAW

1. (find-in-table 'Poland '((Japan . Tokio)))

NIL

1. (find-in-table 'Tokio '((Japan . Tokio)))

JAPAN

**Через рекурсию:**

Поиск по первым элементам точечных пар:

(defun find-first(el lst)

(cond

((endp lst) nil)

((equal el (car (car lst)))

(car lst))

(t (find-first el (cdr lst)))

)

)

Поиск по вторым элементам точечных пар:

(defun find-second(el lst)

(cond

((endp lst) nil)

((equal el (cdr (car lst)))

(car lst))

(T (find-second el (cdr lst)))

)

)

Поиск заданного элемента el в списке точечных пар:

(defun find-in-table-recur (el lst)

(or (cdr(find-first el lst))

(car(find-second el lst))

)

)

Тесты:

1. (find-in-table-recur 'Berlin '((Japan . Tokio) (Russia . Moscow) (Poland . Warsaw)))

Nil

1. (find-in-table-recur 'Russia '((Japan . Tokio) (Russia . Moscow) (Poland . Warsaw)))

MOSCOW

1. (find-in-table-recur 'Warsaw '((Japan . Tokio) (Russia . Moscow) (Poland . Warsaw)))

POLAND

1. (find-in-table-recur 'Poland '((Japan . Tokio) (Russia . Moscow) (Poland . Warsaw)))

WARSAW

1. (find-in-table-recur 'Poland '((Japan . Tokio)))

NIL

1. (find-in-table-recur 'Tokio '((Japan . Tokio)))

JAPAN

Задание 3. Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда:

1. все элементы списка --- числа

**Через функционалы**:

(defun multiplyAll(num lst)

(mapcar #'(lambda(x)(\* x num)) lst)

)

Описание параметров:

num – множитель;

lst – список элементов;

Тесты:

1. (multiplyAll 10 '(1 2 3))

(10 20 30)

1. (multiplyAll 10 '(3))

(30)

1. (multiplyAll 10 '())

NIL

1. (multiplyAll 10 '(12 -10))

(120 -100)

1. (multiplyAll -1 '(1 2 -10))

(-1 -2 10)

**Через рекурсию**:

(defun multiplyAll(num res)

(cond ((equal (cdr res) Nil) (cons (\* num (car res)) Nil))

(T (cons (\* num (car res))(multiplyAll num (cdr res))))

)

)

Описание параметров:

num – множитель;

res – список элементов;

Тесты:

1. (multiplyAll 5 '(1 2 3 4 5))

(5 10 15 20 25)

1. (multiplyAll -1 '(1 2 3 4 5))

(-1 -2 -3 -4 -5)

1. (multiplyAll 0 '(1 2 3 4 5))

(0 0 0 0 0)

1. (multiplyAll -10 '(-1 -2 3 4 5))

(10 20 -30 -40 -50)

1. элементы списка -- любые объекты.

**С функционалами:**

(defun multiplyAll(num lst)

(mapcar #'(lambda(x)(cond ((numberp x) (\* x num)) (T x))) lst)

)

Описание параметров:

num – множитель;

lst – список элементов;

Тесты:

1. (multiplyAll 5 '(1 2 3 4 5))

(5 10 15 20 25)

1. (multiplyAll -1 '(1 2 3 4 5))

(-1 -2 -3 -4 -5)

1. (multiplyAll -10 '(-1 -2 3 4 5))

(10 20 -30 -40 -50)

1. (multiplyAll 0 '(1 2 3 4 5))

(0 0 0 0 0)

1. (multiplyAll 10 '(1 a 2 3 r 1 4 v))

(10 A 20 30 R 10 40 V)

Задание 4. Напишите функцию, которая уменьшает на 10 все числа из списка аргумента этой функции.

**Через функционалы:**

(defun decreaseAll(lst)

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

)

Описание параметров:

lst – список элементов, из которых будет произведено вычитание;

Тесты:

1. (decreaseAll '(1 2 3 4 5))

(-9 -8 -7 -6 -5)

1. (decreaseAll '(10 20 30))

(0 10 20)

1. (decreaseAll '(10))

(0)

1. (decreaseAll '())

NIL

**Через рекурсию:**

(defun decreaseAll\_rec(lst)

(cond ((null lst) Nil)

(T (cons (- (car lst) 10)(decreaseAll\_rec (cdr lst))))

)

)

Тесты:

1. (decreaseAll\_rec '(1 2 3 4 5))

(-9 -8 -7 -6 -5)

1. (decreaseAll\_rec '(10 20 30))

(0 10 20)

1. (decreaseAll\_rec '(10))

(0)

1. (decreaseAll ())

NIL

Задание 5. Написать функцию, которая возвращает первый аргумент списка -аргумента, который сам является непустым списком.

**Через функционалы:**

(defun empty(lst)

(car (remove Nil (mapcar #'(lambda(x)(cond ((listp x)

(cond ((null x) nil)(T x)))

(T nil))) lst)

)

)

)

Описание параметров:

lst – список элементов, в котором производится поиск непустого списка;

Тесты:

1. (empty '(1 2 (3 1)))

(3 1)

1. (empty '(1 2 () 3 5 (2 4)))

(2 4)

1. (empty '(()))

NIL

1. (empty '(1 2 3 5))

NIL

1. (empty '(1 2 3 4 (1 2) ()))

(1 2)

**Через рекурсию:**

(defun empty\_rec(lst)

(cond ((and (listp (car lst))(not (eq (car lst) nil))) (car lst))

((equal (cdr lst) Nil) Nil)

(T (empty\_rec (cdr lst)))

)

)

Тесты:

1. (empty\_rec '(1 2 (3 1)))

(3 1)

1. (empty\_rec '(1 2 () 3 5 (2 4)))

(2 4)

1. (empty\_rec '(()))

NIL

1. (empty\_rec '(1 2 3 4 (1 2) ()))

(1 2)

Задание 6. Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10.

(Вариант: между двумя заданными границами. )

**Через функционалы:**

(defun between(min max lst)

(remove Nil (mapcar #'(lambda(x)

(cond ((and (>= x min)(<= x max)) x)(T Nil))) lst)

)

)

Описание параметров:

min – нижняя граница поиска (входит);

max – верхняя граница поиска (входит);

lst – список элементов, в котором производится поиск;

Тесты:

1. (between 1 10 '(1 2 3 4 5 10 21 24 2 21 51 -12 2342 12 44))

(1 2 3 4 5 10 2)

1. (between 3 5 '(1 2 3 4 5))

(3 4 5)

1. (between 1 1 '(-5 -3 -1 1 2 3 4 5))

(1)

1. (between -4 0 '(-4 -2 0))

(-4 -2 0)

**Через рекурсию:**

(defun between\_rec(min max lst)

(cond ((equal (cdr lst) Nil)

(cond ((and (>= (car lst) min)(<= (car lst) max)

(cons (car lst) Nil)))(T Nil)))

(T (cond ((and (>= (car lst) min)(<= (car lst) max))

(cons (car lst) (between\_rec min max (cdr lst))))

(T (between\_rec min max (cdr lst)))

)

)

)

Тесты:

1. (between 1 10 '(1 2 3 4 5 10 21 24 2 21 51 -12 2342 12 44))

(1 2 3 4 5 10 2)

1. (between\_rec 3 5 '(1 2 3 4 5))

(3 4 5)

1. (between\_rec 1 1 '(-5 -3 -1 1 2 3 4 5))

(1)

1. (between\_rec -4 0 '(-4 -2 0))

(-4 -2 0)

Задание 7. 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.)

**Через функционалы:**

(defun decart(lst1 lst2)

(mapcan #'(lambda(x1)

(mapcar #'(lambda(x2)(cons x1 (cons x2 Nil))) lst2)) lst1)

)

Описание параметров:

lst1 – первый список элементов;

lst2 ­ второй список элементов;

Тесты:

1. (decart '(1 2) '(3 4))

((1 3) (1 4) (2 3) (2 4))

1. (decart '(1) '(5))

((1 5))

1. (decart '() '())

NIL

**Через рекурсию:**

(defun dec\_rec(el lst)

(cond ((null lst) Nil)

(t (cons (cons el (cons (car lst) nil)) (dec\_rec el (cdr lst)))))

)

(defun decart\_rec(x y)

(cond ((null x) nil)

(t (nconc (dec\_rec (car x) y) (decart\_rec (cdr x) y))))

)

Тесты:

1. (decart\_rec '(1 2) '(3 4))

((1 3) (1 4) (2 3) (2 4))

1. (decart\_rec '(1) '(5))

((1 5))

1. (decart\_rec '() '())

NIL

Задание 8. Почему так реализовано reduce, в чем причина?

(reduce #'+ ()) -> 0

(reduce #'+ ()) -> 0

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*. Результатом вычисления функции + без аргументов будет 0, а результатом вычисления функции \* без аргументов будет 1, т.к. это нейтральные элементы для данных операций.

**Теоретические вопросы:**

* Способы организации повторных вычислений в Lisp

Для организации многократных вычислений в Lisp могут быть использованы функционалы — функции, которые особым образом обрабатывают свои аргументы. Также для организации многократных вычислений в Lisp может быть использована рекурсия. Рекурсия — это ссылка на определяемый объект во время его определения.

* Различные способы использования функционалов

Функционалы используются в Lisp для организации повторных вычислений. Также могут использоваться для обработки списковых таблиц с помощью функционалов.

* 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.

* Способы повышения эффективности реализации рекурсии

Один из методов повышения эффективности рекурсии является организация хвостовой рекурсии. Для этого может потребоваться использовать дополнительные параметры. Такая рекурсия может быть путём формальной и гарантированно корректной перестройки кода заменена на итерацию. Такая оптимизация реализована во многих оптимизирующих компиляторах, а в трансляторах Scheme, одного из диалектов Lisp, такая оптимизация является обязательной.