|  |  |
| --- | --- |
| 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**\_\_**

|  |  |
| --- | --- |
| **Дисциплина** Функциональное и логическое программирование  **Студент** Зейналов З. Г.  **Группа** ИУ7-61Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Толпинская Н. Б. |  |

Москва.

2020 г.

1. **Ход работы**
2. **Контрольные вопросы**

2. Написать предикат set-equal, который возвращает t, если два его множество-

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

**А) Реализация с использованием функционалов**

(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))))

    )

)

Set1 – первый список-множество.

Set2 – второй список-множество.

Тесты:

\* (set-equal '(1 2 3) '(2 3 1))

T

\* (set-equal '(1 2 3) '(2 1 3))

T

\* (set-equal '(1 3) '(1 3))

T

\* (set-equal '(1 3) '(3 1))

T

\* (set-equal '(1) '(3))

NIL

**Б) Рекурсивная реализация**

(defun set-equal\_rec (set1 set2)

    (cond

        ((null set1) t)

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

        (t nil)

    )

)

(defun set-equal\_rec (set1 set2)

    (and (is\_in set1 set2) (is\_in set2 set1))

)

Set1 – первый список-множество.

Set2 – второй список-множество.

Предикат is\_in рекурсивно проверяет входит ли каждый элемент из множества set1 в set2 и возвращает T, если это так.

Тесты

\* (set-equal\_rec '(1 2 3) '(2 3 1))

T

\* (set-equal\_rec '(1 2 3) '(2 1 3))

T

\* (set-equal\_rec '(1 3) '(1 3))

T

\* (set-equal\_rec '(1 3) '(3 1))

T

\* (set-equal\_rec '(1) '(3))

NIL

3. Напишите необходимые функции, которые обрабатывают таблицу из точечных пар:

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

**A)Реализация с использованием функционалов**

(defun myrassoc (name lstCons)

    (car (car (remove nil(mapcar #'(lambda (x) (if (equal (cdr x) name) x nil)) lstCons))))

)

(defun myassoc (name lstCons)

    (cdr (car (remove nil(mapcar #'(lambda (x) (if (equal (car x) name) x nil)) lstCons))))

)

Name – имя, по которому производится поиск.

lstCons – список в котором осуществляется поиск.

Функция myrassoc возвращает аттрибут-связь к переданному параметру из списка точечных пар lst, в которой второй элемент равен name.

Функция myassoc аттрибут-связь к переданному параметру из списка точечных пар lst, в которой первый элемент равен name.

Тесты:

\* k

((RUSSIA . MOSCOW) (UK . LONDON))

\* (myassoc 'UK k)

LONDON

\* (myrassoc 'Moscow k)

RUSSIA

\*

**Б) Рекурсивная реализация**

(defun myassoc\_rec (name lstCons)

    (cond

        ((endp lstCons) nil)

        ((equal name (car (car lstCons))) (car lstCons))

        (t (myassoc\_rec name (cdr lstCons)))

    )

)

Name – имя, по которому производится поиск.

lstCons – список в котором осуществляется поиск.

(defun myrassoc\_rec (name lstCons)

    (cond

        ((endp lstCons) nil)

        ((equal name (cdr (car lstCons))) (car lstCons))

        (t (myrassoc\_rec name (cdr lstCons)))

    )

)

Тесты:

\* k

((RUSSIA . MOSCOW) (UK . LONDON))

\* (myassoc\_rec 'UK k)

LONDON

\* (myrassoc\_rec 'Moscow k)

RUSSIA

\*

7. Напишите функцию, которая умножает на заданное число-аргумент все числа

из заданного списка-аргумента, когда

a) все элементы списка -- числа

А) **Реализация с использованием функционалов**

(defun multiplyAll (num resNum)

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

)

Num – число, на которое должны быть умножены числа в списке resNum.

Б) **Рекурсивная реализация**

(defun multiplyAll\_rec(num resnum)

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

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

    )

)

6) элементы списка **-- любые объекты**.

Функция умножает число num на каждый элемент списка resNum. Если элемент не является числом, то на его место ставится число nums.

Num – число, на которое должны быть умножены числа в списке resNum.

(defun multiplyNumbers (num resNum)

    (mapcar #'(lambda (x) (if (numberp x) (\* x num) x)) resnum)

)

Тесты:

\* (multiplyAll 5 '(1 2 3 4))

(5 10 15 20)

\* (multiplyNumbers 5 '(1 2 3 4))

(5 10 15 20)

\* (multiplyNumbers 5 '(1 2 (1 2) 4))

(5 10 (1 2) 20)

\* (multiplyAll\_rec 5 '(1 2 3 4))

(5 10 15 20)

\*

**Лаб. работа № 6**

2. Напишите функцию, которая уменьшает на 10 все числа из списка

аргумента этой функции.

А) **Реализация с использованием функционалов**

(defun decreaseTen (lst)

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

)

Lst – список, в котором все числа уменьшаются на 10.

\* (decreaseTen '(1 2 3 4))

(-9 -8 -7 -6)

\* (decreaseTen '())

NIL

\* (decreaseTen '(5))

(-5)

Б) **Реализация с использованием рекурсии**

(defun decreaseTen\_rec(lst)

    (cond ((null lst) nil)

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

)

Lst – список, в котором все числа уменьшаются на 10.

\* (decreaseTen\_rec '(1 2 3 4))

(-9 -8 -7 -6)

\* (decreaseTen\_rec '())

NIL

\* (decreaseTen\_rec '(5))

(-5)

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

А) **Реализация с использованием функционалов**

(defun firstlst (lst)

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

)

Lst – список, в котором осуществляется поиск первого непустого списка.

\* (firstlst '(1 2 3 4 5))

NIL

\* (firstlst '(1 2 (1 2) 4 5))

(1 2)

\* (firstlst '(1 () (1 2) 4 5))

(1 2)

\* (firstlst '((5 3) () (1 2) 4 5))

(5 3)

**Б) Реализация с использованием рекурсии**

(defun firstlst\_rec (lst)

    (cond

        ((null lst) nil)

        ((and (listp (car lst)) (not (equal (car lst) nil))) (car lst))

        (t (firstlst (cdr lst)))

    )

)

Lst – список, в котором осуществляется поиск первого непустого списка

\* (firstlst\_rec '(1 2 3 4 5))

NIL

\* (firstlst\_rec '(1 2 (1 2) 4 5))

(1 2)

\* (firstlst\_rec '(1 () (1 2) 4 5))

(1 2)

\* (firstlst\_rec '((5 3) () (1 2) 4 5))

(5 3)

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

которые больше 1 и меньше 10.

(Вариант: между двумя заданными границами. )

А) **Реализация с использованием функционалов**

(defun between (from to lst)

    (remove nil (mapcar #'(lambda (x) (if (<= x to) (if (>= x from) x))) lst))

)

Выбираются числа, которые попадают в заданный интервал. Затем убираются nil’ы.

From – Значение от которого начинается диапазон сравнения

To – значение на котором заканчивается диапазон сравнения.

Lst – список чисел, которые необходимо проверить, на нахождение их в диапазоне от From до To

Тесты:

\* (between 1 10 '(1 2 100 200 -100 2 1))

(1 2 2 1)

\* (between 1 10 '(1))

(1)

\* (between 1 10 '())

NIL

\* (between 1 10 '(1 10))

(1 10)

Б**) Рекурсивная реализация**

(defun between\_rec(from to lst)

    (cond ((equal (cdr lst) Nil) (cond ((and (>= (car lst) from)(<= (car lst) to) (cons (car lst) nil)))(T Nil)))

          (T (cond

            ((and (>= (car lst) from)(<= (car lst) to))(cons (car lst) (between\_rec from to (cdr lst))))

            (T (between\_rec from to (cdr lst)))

            )

          )

    )

)

Тесты:

\* (between\_rec 1 10 '(1 2 100 200 -100 2 1)

)

(1 2 2 1)

\* (between\_rec 1 10 '(1 2 100 200 -100 2 1))

(1 2 2 1)

\* (between\_rec 1 10 '(1))

(1)

\* (between\_rec 1 10 '())

NIL

\* (between\_rec 1 10 '(1 10))

(1 10)

5. 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.)

**А) Реализация с использованием функционалов**

(defun decart (lst1 lst2)

    (mapcan #'(lambda (x1)(mapcar #'(lambda (x2) (cons x1(cons x2 nil)))lst2)) lst1)

)

Lst1 – список параметров множества А

Lst2 – список параметров множества Б

Создаются списки с парами из обоих списков, затем эти списки объединяются в один список.

Тесты:

\* (decart '(1 2) '(a b))

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

\* (decart '(1 2) '(3 4))

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

\*

Б) **Рекурсивная реализация**

(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(lst1 lst2)

 ( cond ( (null lst1) nil)

 (t (nconc (dec\_rec (car lst1) lst2) (decart\_rec (cdr lst1) lst2)))

 )

)

el – элемент, который состовляет пару с каждым из элементов списка lst.

Lst1 – список параметров множества А

Lst2 – список параметров множества Б

Тесты:

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

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

\* (decart\_rec '(1 2) '(a b))

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

\* (decart\_rec '(1 2 3) '(a b c))

((1 A) (1 B) (1 C) (2 A) (2 B) (2 C) (3 A) (3 B) (3 C))

\*

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. 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.