Федеральное государственное бюджетное образовательное учреждение высшего

образования

«Чувашский государственный университет им. И.Н. Ульянова»

Кафедра вычислительной техники

Лабораторная работа №10

**Рекурсия в Лиспе**

Вариант 9

Выполнил: Иванов В.С.

студент группы ИВТ-41-22

Проверил: кандидат технических наук

Обломов Игорь Александрович

Чебоксары, 2024

Цель работы: изучить основы рекурсивного программирования

Теоретические сведения.

Основное средство функционального программирования – *рекурсия*. Любое определение, описывающее в терминах самого себя, является рекурсивным.

Рекурсивная функция всегда должна содержать, по крайней мере, одну терминальную ветвь – условие окончания. В тот момент, когда вычисления доходят до рекурсивной ветви, функционирующий процесс приостанавливается и запускается новый такой же процесс, но на новом уровне.

В Лиспе рекурсия может быть по *значению* или по *аргументам*. Если рекурсивный вызов является выражением, формирующим результат функции, то имеет место рекурсия по значению. Если же рекурсивный вызов является аргументом некоторой другой функции, формирующей окончательный результат, то имеет место рекурсия по аргументам.

Различают следующие формы рекурсии:

– простая рекурсия, если рекурсивный вызов встречается в некоторой ветви не более одного раза;

– параллельная рекурсия, если рекурсия встречается одновременно в нескольких аргументах функции;

– взаимная рекурсия, если при определении одной функции, вызывается другая, которая в свою очередь вызывает первую;

– рекурсия более высокого порядка, если в качестве аргумента рекурсивного вызова выступает рекурсивный вызов.

***Простая рекурсия***. Простой рекурсии в процедурном программировании соответствует обыкновенный цикл.

Большую помощь в написании и отладке рекурсивных функции в Лиспе может оказать встроенный трассировщик. Трассировка функции включается с помощью директивы trace, в частности, для нашего примера: **(trace copy-lst).**

Трассировка функции отключается директивой untrace.

***Параллельная рекурсия***. Рекурсия называется параллельной, если она встречается одновременно в нескольких аргументах функции. Параллельная рекурсия применяется в том случае, если необходимо обработать список по всем уровням.

***Взаимная рекурсия***. Рекурсия является взаимной между одной и более функциями, если они вызывают друг друга. Взаимная рекурсия часто применяется в системном программировании, в частности, при определении функций высшего порядка.

***Рекурсия более высокого порядка***. Более сложные вычисления можно осуществить с помощью рекурсии более высокого порядка. В качестве аргумента рекурсивного вызова может выступать рекурсивный вызов, в этом случае имеет место рекурсия более высокого порядка.

Индивидуальное задание.

Определить функцию замены произвольного элемента в списке на верхнем уровне

Код программы.

;прямая рекурсия

(defun replace\_elem (lst index val)

    (cond ((null lst) nil)

        ((= index 0) (cons val (cdr lst)))

        (t (cons (car lst) (replace\_elem (cdr lst) (1- index) val)))

    )

)

;взаимная рекурсия

(defun replace\_elem\_vz (lst index val)

    (cond

        (t

            (replace\_elem lst index val)

        )

    )

)

(setq my\_lst '(1 2 3 4 5))

(write (replace\_elem my\_lst 2 '11))

(terpri)

(write (replace\_elem\_vz my\_lst 0 '22))

(defun sum (lst)

    (cond

        ((null lst) 0)

        (t (+ (car lst) (sum (cdr lst))))

    )

)

(defun check\_atom (lst)

    (cond

        ((null lst) nil)

        ((atom (car lst)) T)

        (t (check\_atom lst))

    )

)

;(write(check\_atom my\_lst))

(defun ackerman (m n)

       (cond ((= m 0) (+ n 1))

                ((= n 0) (ackerman (- m 1) 1))

                (t (ackerman (- m 1)

                (ackerman m (- n 1)))

)))

Вывод: изучил основы рекурсивного программирования.