ЛАБОРАТОРНАЯ РАБОТА 4

ЛОКАЛЬНЫЕ ОПРЕДЕЛЕНИЯ

1. Цель и задачи.

Целью работы является практическое изучение различных видов локальных определений и особенностей их использования в рекурсивных программах.

Основные задачи :

- Изучить применение техники нисходящей и восходящей рекурсии при написании рекурсивных функций с использованием локальных определений;
- Сравнить возможности локальных определений LET и LAMBDA по организации вычислений в рекурсивных программах;

2. Краткие теоретические сведения.

Локальные определения относятся к управляющим структурам Лиспа и обеспечивают :
1). Сокращение количества рекурсивных вызовов функций;
2). Делают программу более удобочитаемой.

Существует две конструкции локальных определений в Лиспе : LET и LAMBDA.

Функция LET создает локальную связь и является синтаксическим видоизменением LAMBDA-вызова, в котором формальные и фактические параметры помещены совместно в начале формы :

```
(let ((формальный параметр 1 фактический параметр 1)
      . . .
      (формальный параметр 1 фактический параметр 1))
     <тело функции> )
```

В muLISPe LET является библиотечной функцией, ее можно использовать, вызвав COMMON.LSP через RDS.

Различают три разновидности рекурсивных определений :

- Восходящая рекурсия;
- Нисходящая рекурсия;
- Параллельная рекурсия (рекурсия по дереву),

из которых в данной работе мы рассмотрим первые две.

Нисходящая рекурсия последовательно разбивает рассматриваемую задачу на все более простые, пока не доходит до **терминальной ситуации.**

Под ***терминальной ситуацией*** принято понимать ситуацию, когда не требуется продолжения рекурсии. В этом случае значение определяемой функции получается без использования обращения к ней (применительно к другим значениям аргумента), характерного для рекурсивных определений.

Только после этого начинается формирование ответа, а промежуточные результаты передаются обратно - вызывающим функциям.

Примером использования техники нисходящей рекурсии может послужить построение копии списка в памяти виртуальной Лисп-машины (смотри *Лекцию 7*) :

Вариант для muLISP :

```
(DEFUN DCOPY (LAMBDA (LST)
     (COND ( (NULL LST) NIL )
          (  T  (CONS (CAR LST) (DCOPY (CDR LST))) ))))
```

Вариант для newLISP-tk :

```
(define (dcopy
 (lambda (lst)
  (cond
    ((null? lst) nil)
    (true (cons (first lst) (dcopy (rest lst) ) ) ) ) ) ))
```

В ***восходящей рекурсии*** промежуточные результаты вычисляются на каждой стадии рекурсии, так что ответ строится постепенно и передается в виде параметра рабочей памяти до тех пор, пока не будет достигнута терминальная ситуация. К этому времени ответ уже готов, и нужно только передать его вызывающей функции верхнего уровня.

Примером использования техники восходящей рекурсии может послужить реверсирование списка :

Вариант для muLISP :

```
 (DEFUN REVERSE2 (LAMBDA (LST)
     (COP LST NIL)))

; Вспомогательная функция копирования

   (DEFUN COP (LAMBDA (LST W)
     (COND ( (NULL LST) W )
          (  T  (COP (CDR LST) (CONS (CAR LST) W)) ))))
```

Вариант для newLISP-tk :

```
(define reverse2 (lambda (lst)
 (cop lst '())))

(define cop (lambda (lst w)
 (cond
   ((null? lst) w )
   (true (cop (rest lst) (cons (first lst) w)) )
 )))
```

3. Задание на лабораторную работу.

3.1 Задание 1.

Описать функцию вычисления факториала. Рассмотреть варианты решения задачи с применением локальных определений LAMBDA и LET.

3.2 Задание 2.

Разработать программу символьного дифференцирования в соответствии с правилами, изложенными в [3], стр. 194-196. Рассмотреть варианты решения задачи с применением локальных определений LAMBDA и LET.

3.3 Задание 3.

Решить задачу из лабораторной работы №2 с применением локальных определений LAMBDA и LET.

3.4 Задание 4.

Реализовать программу-простейший интерпретатор лисповских программ. На вход интерпретатора подается текст, который может быть интерпретирован как вызов или суперпозиция функций Лиспа, пример (для muLISP'a) : '(cons(car(cdr '(e r t w))) (cons (cdr '(g h 6)) nil)). Программа должна обеспечивать выполнение такого рода примеров.
Требования к программе :

- Должна обеспечивать интерпретацию базовых функций Лиспа и арифметических операций +, -, /, *;
- В программе должны использоваться локальные определения;
- Не допускается использование встроенной функции-интерпретатора EVAL;

3.5 Задание 5.

Дополнить интерпретатор из задания 4 в соответствии с вариантом индивидуального задания из Таблицы 1.

Таблица 1. Вариант индивидуального задания.

Вариант.	Задание.
1.	Функция вычисления степени.
2.	Функция вычисления корня n-й степени из числа x.
3.	Арксинус.
4.	Арккосинус.
5.	Арктангенс.
6.	Функция объединения двух списков.
7.	Функция определения принадлежности объекта списку.
8.	Функция объединения множеств.
9.	Функция пересечения множеств.
10.	Функция вычитания множеств.
11.	Функция дополнения до пересечения множеств.
12.	Функция конкатенации двух символьных строк.
13.	Функция преобразования списка символов в строку.
14.	Функция реверсирования списка.
15.	Функция нахождения среднего арифметического числовых элементов списка.
16.	Функция вычисления логарифма по заданному основанию.
17.	Функция определения четности.

4. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :

- формулировку цели и задач;
- описание процесса разработки программ. Для каждого задания в обязательном порядке приводится : обоснование выбранных структур функций и стиля рекурсивного определения (восходящая, либо нисходящая рекурсия), условия окончания рекурсии в каждом случае и формирование новых значений аргументов при рекурсивном вызове;
- выводы по проделанной реализации.

Литература.

1) Хювенен Э., Сеппянен Й. Мир Лиспа. Т.1. – М.: Мир, 1990. С. 128-131.
2) Lutz Mueller newLISP™ For BSDs, Linux, Mac OS X, Solaris and Win32. Users Manual and Reference v.9.1 // www.nuevatec.com
3) Клоксин У., Меллиш К. Программирование на языке Пролог. – М.: Мир, 1987. С. 194-196.