ЛАБОРАТОРНАЯ РАБОТА 3

Локальные определения и функционалы

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.

2.1. Локальные определения

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 GCLispe 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 GCLisp :

```
(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 GCLisp :

```
(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))
)
)))
```

2.2 Применяющие функционалы.

Лиспе определено три применяющих функционала :

- (FUNCALL <функциональный аргумент> аргументы), имеется в GCLisp'e.
- (APPLY <функциональныйаргумент> список аргументов), имеется в GCLisp'e и newLISP-tk.
- (EVAL <любое лисповское выражение>), имеется в GCLisp'e и newLISP-tk.

качестве функционального аргумента можно использовать имя функции, lambda-вызов или лисповское выражение, значением которого является имя функции или lambda-вызов.

2.3. Отображающие функционалы.

GCLisp'e определено шесть отображающих функционалов :

- (MAPCAR <функциональный аргумент>списки)
- (MAPLIST <функциональный аргумент> списки)
- (MAPCAN<функциональный аргумент> списки)
- (MAPCON<функциональный аргумент> списки)
- (MAPC <функциональный аргумент> списки)
- (MAPL <функциональный аргумент> списки).

Основным назначением этих функционалов является отображение аргументов в новую последовательность. Перечисленные функционалы можно разделить на две группы: в первую группу включаем функционалы MAPCAR, MAPCAN,MAPC и во вторую группу MAPLIST,MAPCON и MAPL. Первый вид отображающих функционалов отображает функциональный аргумент отдельно на каждый элемент списка. Второй на последовательность, состоящую из списков, каждый последующий список представляет собой хвост предыдущего. Результатом повторяющихся вычислений будет список, содержащий результаты отображений.

Примеры.

```
(MAPCAR '* '(1 2 3 4 5)'(10 20 30 40 50)) – результатом будет :
(10 40 90 160 250); (MAPCAR '(lambda (y)(list (print y)))'(tom
anny mary)).
```

В результате вызова такого применяющего функционала, во-первых, в текущий выходной поток будут выведены :

tom
anny
mary,

во-вторых, в качестве значения возвращается список ((tom)(anny)(mary)).

```
(MAPCAR '(lambda (var)(if (atom var) var (car
var))) list)
```

Результат такого вызова зависит от вида аргумента list : если list представляет собой список атомов, то этот же список и будет возвращен. Если list является списком списков, то возвращается список, соде ржащий головы подсписков. Если, к примеру, list представляет собой '((first second)(lisp ll) 7 8) то возвращается (first lisp 7 8).

В следующем примере воспользуемся функцией sum :

```
(defun sum
 (lst)
 ((null lst)
 0)
 (+ (car lst)(sum (cdr lst))))

(MAPLIST 'sum '(10 20 30 40 50)).
```

В результате вызова отображающего функционала MAPLIST получим (150 140 120 90 50) т.е. возвращается список, составленный из результатов применения функции sum сначала ко всему списку, затем к хвосту списка,затем к хвосту хвоста и т.д. до пустого списка.

Количество списков в каждом вызове определяется функциональным аргументом : если это функция одного а ргумента, то в вызове функционала должен присутствовать один список, если двух, то два и т.д.

```
(MAPCAR 'LIST '(A S D F))-возвращает
 ((A)(S)(D)(F)); (MAPCAN 'LIST '(A S D F))-
 возвращает (A S D F); (MAPCAN '(lambda (var)

((atom var) nil)
((caddr var)(list (car var)(caddr var))) )
 '(((a)(2) (+ a 2))((s)(3)(- s 1)))
```

Вызов такого функционала возвращает список следующего вида: :
(((a)(+ a 2)) ((s)(- s 1))), в котором элементы попарно обьединены в списки.

В newLISP-tk определен отображающий функционал map. Он отображает аргументы-списки в новый список применением к одинаково расположенным элементам этих списков функции, представленной первым аргументом.

Примеры :

```
(map + '(1 2 3) '(50 60 70)) возвращает '(51 62 73)

(map if '(true nil true nil true) '(1 2 3 4 5) '(6
 7 8 9 10)) возвращает '(1 7 3 9 5)
```

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 GCLisp'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.

3.6. Задание 6.

Написать программу обработки текста естественного языка с использованием отображающих функционалов в соответствии с заданием из таблицы. Текст рекомендуется представлять списком списков : каждое предложение- список слов, весь текст- список предложений.

Таблица 2. Вариант задания 6.

Вариант.	Задание.
1,13.	Дан текст. Сделать заглавной первую букву первого слова каждого предложения. Предполагается, что первое слово предложения может как начинаться, так и не начинаться с заглавной буквы.
2,14.	Дан текст. Сделать заглавной каждую букву каждого слова, начинающегося с заглавной буквы.
3,15.	Дан текст. В каждом слове текста заменить заданную литеру заданной литерой (сочетанием литер). Пример : Заменяемая литера : “б”, заменяющее сочетание литер : “ку”, слово : “а***б***ракада***б***ра”, результат : “а***ку***ракада***ку***ра”.
4,16.	В каждом слове удалить литеру, стоящую между двумя заданными.
5,17.	Сформировать список, информирующий о вхождении заданной литеры в текст в текст в виде ((<0 1 5 2 0>) (<3 0 1 5 2 0 1 0>)...). Цифры указывают количество вхождений литеры в каждое слово предложения.
6,18.	Дан текст. Заменить в каждом предложении все вхождения заданного слова на заданное новое слово.
7,19.	Дан текст. Удалить из каждого слова в каждом предложении все повторяющиеся литеры.
8,20.	Дан текст. В каждом слове каждого предложения для повторяющихся литер произвести следующую замену : повторные вхождения литер удалить, к первому вхождению литеры приписать число вхождений литеры в слово. Пример : ‘((aaabb ccccddd)(eeefggg hhkl)) преобразуется в ‘((a3b2 c4d3)(e3fg3 h2kl))
9,21.	Дан текст. В каждом слове вставить после заданного 3-буквенного сочетания заданное 2-буквенное.
10,22.	Дан текст. Вставить заданное новое слово после каждого вхождения другого заданного слова.
11,23.	Дан текст. Записать каждое предложение текста в порядке возрастания количества гласных букв в слове.
12,24.	Дан текст. Переписать каждое предложение, расположив слова в обратном алфавитном порядке.
25.	Написать программу, которая в каждом слове исходного текста меняет местами первую и последнюю буквы.

3.7. Задание 7.

Дана фраза украинского (русского) языка. Написать программу, которая разбивает каждое слово фразы на слоги.

3.8. Задание 8.

“Язык сплетника ”. Есть ключевое слово, например, “сплетня”. Слово переводится на язык сплетника путем отделения первого слога в переводимом и ключевом слове (например, слово и спле-тня) с последующей перестановкой по определенным правилам :
‘(слово сплетня) преобразуется в ‘(сплево слотня).
Каждое слово преобразуется в пару слов. Первое слово есть конкатенация первого слога ключевого слова и части переводимого слова, оставшейся после отделения от него первого слога. Второе слово есть конкатенация первого слога переводимого слова и части ключевого слова, оставшейся после отделения от него первого слога.

Написать программу перевода предложения украинского языка на заданный таким образом “тайный” язык.

3.9. Задание 9.

Написать программу в соответствии с заданием из Таблицы 3.

Таблица 3. Варианты задания 9.

Вариант.	Задание.
1,12,14,20.	“Тайныеязыки”. Используя разработанную по заданию 3 программу, построить программу перевода предложения русского языка на так называемый цыганский жаргон, в котором ключевым словом всегда является следующее слово. Если последнее слово остается без пары, то его можно переводить или в себя, или с некоторым заданным вспомогательным ключевым словом, например, “сплетня”.
2,7,15,21.	Написать программу, которая заменяет слова исходного текста номерами их семантических эквивалентов по словарю в зависимости от значения трехбуквенного концы слова (см. [2]). Если слово содержит менее трех букв, то замену не производить.
3,8,16,22.	“Частотныйсловарь”. Написать программу, которая по заданному тексту строит список пар : (<слово> <частота встречаемости в тексте>), см. [3].
4,9,17,23.	Написать программу, исключающую в исходном тексте из каждого слова его окончание по словарю. Словарь окончаний представлять списком строк.
5,6,10,18,24.	Написать программу, которая в исходном тексте заменяет слова, являющиеся значениями Лексических Функций (ЛФ) [4] от других слов того же текста, списками вида : {<символ ЛФ по словарю> <ключевое слово>}. Словарь-справочник ЛФ представлять в виде списка троек : (<ключевое слово> <символьное обозначение ЛФ> <значение ЛФ для ключевого слова>). Пример : (“дождь” “Magn” “ливень”).
11,13,19,25.	Написать программу, которая кодирует исходный текст по методу Юлия Цезаря : каждая буква в каждом слове заменяется на следующую.

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 // http://www.newlisp.org/downloads/manual_frame.html
3) Клоксин У., Меллиш К. Программирование на языке Пролог. – М.: Мир, 1987. С. 194-196.
4) Вирт Н. Алгоритмы + структуры данных = программы : Пер. с англ. – М.:Мир, 1985. С. 203.