ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

НОВГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ ЯРОСЛАВА МУДРОГО

ФУНКЦИОНАЛЬНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Часть 1.
Функциональное программирование

Лабораторный практикум

ВЕЛИКИЙ НОВГОРОД
2007

УДК 681.3.06 (075.8)
ББК 32.973.26 – 018.2

Печатается по решению
РИС НовГУ

Рецензент
Профессор кафедры Математического обеспечения и применения ЭВМ
Санкт-Петербургского государственного
электротехнического университета “ЛЭТИ”,
доктор технических наук **Геппенер В.В**.

Функциональное и логическое программирование. Лабораторный практикум. Часть 1. Функциональное программирование /

Авт.-сост. Д. В. Михайлов, Г.М. Емельянов; НовГУ им. Ярослава Мудрого.– Великий Новгород, 2007. 80 с.

Данное учебное пособие предназначено для организации лабораторного практикума по дисциплине «Функциональное и логическое программирование» у студентов специальности 230105 и других специальностей, в учебных планах которых предусмотрены аналогичные дисциплины. В пособии приводятся общие сведения и рекомендации по использованию функциональных языков в задачах интеллектуализации обработки символьной информации. Обращается особое внимание на решение прикладных задач компьютерной лингвистики и интеллектуального общения с ЭВМ. Пособие содержит типовые задания, позволяющие приобрести навыки написания и отладки функциональных программ при построении интеллектуальных систем различного назначения.

УДК 681.3.06 (075.8)
© ББК 32.973.26 – 018.2я73
Новгородский Государственный
Университет, 2007

Содержание

Введение

Современный этап развития вычислительной техники характеризуется расширением сфер ее применения. На первый план выдвигаются задачи интеллектуализации процессов машинной обработки информации, моделирования рассуждений человека-эксперта при решении прикладных задач из различных областей человеческого знания. Кроме того, актуальным является также полное и непротиворечивое представление самих знаний. Тенденция развития современных языков программирования показывает необходимость разработки предметно-ориентированных языков, максимально приближенных к Естественному Языку (ЕЯ). Интеллектуализация инструментального ПО немыслима без изучения различных аспектов языкового поведения человека, разработки и исследования математических моделей различных сторон языковой деятельности. В этих условиях инженер по направлению "Информатика и вычислительная техника" должен наряду с хорошей технической подготовкой уметь решать сложные научные задачи представления знаний о языке общения (ЕЯ) во взаимосвязи с предметными знаниями человека-эксперта, организации работы со знаниями, обучения интеллектуальных систем. Решение указанных задач немыслимо без привлечения в той или иной мере декларативного подхода в программировании, позволяющего описывать решение задачи на языке соотношений между объектами в заданной предметной области.

Дисциплина “Функциональное и логическое программирование” для специальности 230105 относится к числу специальных дисциплин. Она включает в себя рассмотрение основных вопросов современной теории и практики логического и функционального программирования и опирается на учебные курсы : «Дискретная математика», «Математическая логика и теория алгоритмов”, ”Алгоритмические языки и программирование”.

Лабораторный практикум по разделу “Функциональное программирование” имеет две составные части.

Первая часть (работы 1–5) включает :

- изучение способов описания и вызова функций в языке Лисп;
- ознакомление с концепцией Строго Функционального Языка (СФЯ);
- изучение основных методов разработки функциональных программ с позиций СФЯ.

Вторая часть (работы 6–9) предназначена для ознакомления с возможностями Лиспа, актуальными для построения прикладных интеллектуальных систем.

Лабораторные работы по разделу “Функциональное программирование” ориентированы на язык программирования Microsoft muLISP. По желанию студента при выполнении заданий практикума могут быть использованы другие известные на сегодняшний день диалекты Лиспа : Visual LISP, Common Lisp, PC-Lisp, X-Lisp.

ОПИСАНИЕ ЛАБОРАТОРНЫХ РАБОТ ПО РАЗДЕЛУ “ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ” КУРСА “ФУНКЦИОНАЛИНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ”

ЛАБОРАТОРНАЯ РАБОТА № 1

ОПИСАНИЕ И ВЫЗОВ ФУНКЦИЙ В ЯЗЫКЕ muLISP.

Цель работы.

Целью работы является изучение базовых функций организации и обработки списков, а также способов описания и вызова нерекурсивных функций в языке программирования Лисп.

Основные задачи :

- Получение навыков работы с интерпретатором Microsoft muLISP.
- Изучение работы примитивных базовых функций списочного ассемблера.
- Изучение работы базовых функций из расширения набора примитивных функций и их сведения к примитивным базовым функциям.
- Ознакомление с описанием неименованных функций в muLISP'e.
- Изучение приемов описания именованных функций через неименованные и с применением современной сокращенной нотации.

1.1. Работа с интерпретатором Microsoft muLISP.

Исполняемым файлом интерпретатора Лиспа является mulisp.com (или mulisp_2.com - для русифицированной версии). Русифицированную версию интерпретатора рекомендуется использовать в комплекте с программой-русификатором для MS DOS, например, keyrus.com или rk.com.

Для работы с интерпретатором необходимо после запуска mulisp.com и появления приглашения $ набирать либо описание функции, либо ее вызов.

Примеры :
$(* 4 5)- вызов функции умножения для двух аргументов;
$(* 5 6 7 8)- тоже для четырех аргументов;
$(- 12 3)- вызов функции вычитания 12-3

Для выхода из среды Microsoft muLISP необходимо набрать в командной строке интерпретатора :
$(SYSTEM).

1.1.1. Рекомендуемая последовательность действий при работе с интерпретатором.

- разработать текст программы;
- записать разработанный текст в файл с расширением .lsp;
- вызвать интерпретатор с указание выполняемого файла.

Для сокращения объема программы в первой лабораторной работе разрешается использовать псевдофункцию, связывающую имена и значения (SETQ имя значение). К примеру, нужно выполнять многократную обработку списка '(a c d (e f)). Обозначим этот список l1 и сделаем это так (SETQ l1 '(a c d (e f))). После такого связывания там, где используется значение списка, можно использовать имя l1.

1.1.2. Рекомендуемая структура программы.

- связывание имен и значений;
- вызовы функций, либо описание функций, т.е. собственно текст программы;
- функция переключения входного потока (RDS).

1.2. Базовые функции Лиспа.

Примитивные (простейшие) базовые функции можно сравнить с основными арифметическими действиями: они просты и их мало.

Назначение этих функций состоит в:
- расчленении S-выражений (функции-селекторы);
- составлении S-выражений (функции-конструкторы);
- анализ S-выражений (функции-предикаты).

Таблица 1.1 Примитивные базовые функции.

Назначение.	Вызов.	Результат.
Селектор	(CAR список)	S-выражение
Селектор	(CDR список)	список
Конструктор	(CONS S-выраж.,список)	список
Предикат	(ATOM S-выражение)	T, либо NIL
Предикат	(EQ атом атом)	T, либо NIL

1.2.1. Функция CAR.

Значением функции является голова списка-аргумента.

Примеры :

$ (CAR '(P Q R))-результат P;
$ (CAR '(A B))- результат A;
$ (CAR '(DOG CAT))-результат DOG;
$ (CAR '((A B) C))-результат (A B);
$ (CAR (A B))-результат не определен, т.к. (A B) интерпретатором воспринимается как вызов функции A с аргументом B.

1.2.2. Функция CDR.

Значением функции является хвост списка-аргумента.

Примеры :

$ (CDR '(P Q R))-результат (Q R);
$ (CDR '(B))-результат NIL.

В muLISP наряду с простейшими базовыми функциями-селекторами CAR и CDR определены более сложные функции, выделяющие любой из десяти первых элементов :
FIRST - выделяет первый элемент списка, SECOND- второй и т.д.

1.2.3. Функция CONS.

Функция CONS имеет два аргумента - первый S-выражение, вторым аргументом обязательно должен быть список. Назначение функции CONS- конструирование нового списка. Новый список получаем путем добавления первого аргумента к списку- второму аргументу.

Для получения списка, содержащего один элемент, нужно в качестве второго списка указать пустой список NIL.

Примеры :

$ (CONS 'A '(B C))-результат есть список (A B C);
$ (CONS '(FIRST SECOND)'(THIRD FOURTH FIFTH)) –
  результат есть ((FIRST SECOND) THIRD FOURTH FIFTH);
$ (CONS NIL NIL) – результат есть одноэлементный список (NIL);

$ (CONS 12 NIL) - результат есть список (12);
$ (CONS '(A B C) NIL) – результат есть список ((A B C)).

1.2.4. Связь между функциями CAR,CDR,CONS.

Список, разделенный с помощью CAR и CDR на голову и хвост, можно объединить с помощью функции-конструктора CONS.

Пример.

$ (CAR ‘(FIRST SECOND THIRD)) – результатом будет FIRST;
$ (CDR ‘(FIRST SECOND THIRD)) – результатом будет список (SECOND THIRD);
$ (CONS (CAR ‘(FIRST SECOND THIRD)) (CDR ‘(FIRST SECOND THIRD))) – результатом будет исходный список (FIRST SECOND THIRD).

1.2.5. Предикат ATOM.

Предикат- это логическая функция. Предикат ATOM возвращает значение T (эквивалент TRUE), если его аргументом является атом и NIL(эквивалент FALSE), иначе.

Примеры использования ATOM :

$ (ATOM 'X)-результатом является T, т.к. аргумент – атом;
$ (ATOM '(1 2 3))-результатом является NIL, т.к. аргумент – список;
$ (ATOM (CDR '(1 2 3))) – результат NIL;
$ (ATOM (CAR '(A B C))) – результат T.

1.2.6. Предикат EQUAL.

Предикат EQUAL проверяет эквивалентность любых S-выражений.

Примеры :

$ (EQUAL '(3 3)'(4 3)) – результат есть NIL, так как списки не совпадают;
$ (EQUAL 'Y 'Y) – результат есть T, так как атомы совпадают;
$ (EQL 5.0 5.0) – результат есть T.

1.2.7. Расширение базовых функций в muLISPe.

Кроме базовых функций в muLISPe есть целый набор встроенных функций, выполняющих преобразования S-выражений и дополняющих базовые функции.

1.2.7.1. Вложенные вызовы функций CAR и CDR.

Путем использования функций CAR и CDR можно выделить любой элемент списка. Например, для выделения второго элемента второго подсписка нужно записать :
$ (CAR (CDR (CAR (CDR '((A B C) (D E) (F H))))))

В muLISP для таких комбинаций CDR и CAR можно использовать более короткую запись в виде одного вызова функции :
(C...R список), где вместо многоточия записывается нужная комбинация букв A (CDR) и D (CDR). **Но максимум – 4 буквы !**

Например, записанная выше конструкция может быть представлена как (CADADR '((A B C) (D E) (F H))).

Для выделения из списка какого-либо элемента : первого, второго, ..., десятого можно использовать специальные функции muLISP : FIRST, SECOND, THIRD,FOURTH, FIFTH, SEVENTH, EIGHTH, NINTH, TENTH. Все эти функции имеют один аргумент – список.

Более того, в muLISP существуют две дополнительные функции для выделения элемента из списка :

- (NTH n список) – выделяет n-ый элемент из списка-второго аргумента функции NTH;
- (LAST список) – возвращает список из одного последнего элемента списка-аргумента.

Примеры использования LAST и NTH :

$ (NTH 4 '(F D C C F H)) – выделяет элемент F, т.к. номера элементов начинается с нуля;
$ (LAST '(A B C D E F)) – возвращает список (F).

1.2.7.2. Дополнительные предикаты сравнения аргументов.

1.2.7.2.1. Предикат EQ.

Предикат EQ сравнивает два атома и принимает значение T, если атомы идентичны, NIL – в противном случае. **EQ сравнивает только атомы и константы T, NIL.**

Примеры :

$ (EQ 'X 'Y) – атомы неэквивалентны, значение функции равно NIL;
$ (EQ () NIL) – значением функции будет T;
$ (EQ T (ATOM 'CAT)) – значением функции будет T.

1.2.7.2.2. Предикат "=".

Предикат "=" применяется для сравнения чисел различных типов. Предикат принимает значение T, если значение чисел совпадают вне зависимости от типа, иначе NIL.

Примеры :

$ (= 3 3) – результат T;
$ (= 3 7) - результат NIL.

1.2.7.3. Предикат NUMBERP.

Предикат NUMBERP имеет один аргумент, который является S-выражением. Предикат принимает значение T, если аргумент является числом любого типа, и NIL – в любом другом случае.

Примеры :

$ (NUMBERP 3.066) – значением является T;
$ (NUMBERP T) – значением является NIL, т.к. T – не число.

1.2.7.4. Предикат NULL.

NULL имеет один аргумент-список. Если список пустой, то NULL принимает значение T, иначе NIL.

1.2.7.5. Функция LIST.

Эта функция может иметь любое количество аргументов. Значением функции является список из заданных аргументов.

Примеры :

$ (LIST '1 '2) – реэультатом является список (1 2);
$ (LIST 'A 'B 'C 'D (+ 3 4)) – результатом является список (A B C D 7).

1.3. Объявление функций.

В функциональных языках программирования различают описания именованных и неименованных функций. Для описания неименованных функций в LISPe используются две конструкции : LAMBDA и NLAMBDA.

1.3.1. Неименованные функции.

Неименованные функции используются для выполнения однократных преобразований или вычислений, такие действия носят локальный характер и используются только одной функцией. Наиболее часто неименованные функции используются в функционалах и макросах.

Для описания неименованных функций используются lambda-вызовы, имеющие следующий вид :

```
((lambda (<список формальных параметров>)
        < тело функции> )
        < список фактических параметров> )
```

Пример. Для функции f1(x,y), которая возвращает в качестве результата y в случае атомарного x и список ‘(x,y) – в противном случае, примером реализации в виде lambda-вызова может послужить :

```
((lambda (x y)
        ((atom (car x)) y)
        (cons x y))
 (car '((f) g h)) '(r t y))
```

1.3.2. Описание именованных функций.

Для объявления именованных функций используются конструкции :

```
(DEFUN <имя функции> (список формальных параметров)
        <lambda -вызов> )

(DEFUN <имя функции> (список формальных параметров)
        <тело функции> )
```

DEFUN - служебное слово (DEfine FUNction - определение функции), с которого обязательно должно начинаться описание именованной функции. Имя функции может быть любым символьным атомом, рекомендуется давать такие имена функциям, которые отражают смысловое содержание функции. Список формальных параметров представляет собой

список символьных атомов, записанных через пробел. Тип параметров не указывается. Тело функции может содержать либо ветвления, либо представлять собой суперпозицию вызовов функций.

Результат, полученный при выполнении функций, связывается с именем функции.

Для примера рассмотрим описание и вызов функции, которая из списка выделяет второй и четвертый элементы и конструирует из них список.

Задачу конструирования нового списка из элементов старого можно решить различными способами, используя базовые функции CONS, CDR, CAR и функции muLISP: SECOND, FOURTH, LIST.

Описание функции непосредственно в интерпретаторе будет выглядеть следующим образом :

```
$ (DEFUN SFLIST (LST) (CONS (SECOND LST) (CONS (FOURTH LST)
NIL)))
```

При использовании lambda-вызова описание имеет вид :

```
$ (DEFUN SFL (LST)
      ((lambda (X Y)
           (CONS X(CONS Y NIL)))
   (SECOND LST)(FOURTH LST)))
```

Эта же функция может быть описана с использованием других функций muLISP: LIST, CADR, CADDDR :

```
$ (DEFUN SFILST1 (LST)
            (LIST (CADR LST)(CADDDR LST)))
```

1.4. Вызов функции.

Вызов функции записывается следующим образом :

(имя функции <список фактических параметров>)

Например, вызов функции SFLIST :
$ (SFLIST '(DOG CAT COW PIG))
дает в качестве результата список (CAT PIG).

1.5. Задание на лабораторную работу.

1) Ознакомиться с описанием лабораторной работы.
2) Выполнить примеры.

3) Выполнить свой вариант задания, вариант выдает преподаватель. Задание выполнить различными способами, применяя простейшие и функции из расширения базовых функций muLISPa.

Задание 1.

Описать неименованную функцию для объединения голов трех списков в один список, исходные данные взять из Таблицы 1.2.

Задание 2.

Описать именованную функцию для создания нового списка из элементов нескольких исходных списков. В качестве исходных списков использовать списки Таблицы 1.2. Номера элементов списков взять в Таблице 1.3.

Таблица 1.2. Исходные списки.

Вариант	Исходные списки		
	1	2	3
1	(Y U I)	(G1 G2 G3)	(KK LL MM JJJ)
2	(G55 G66 G777)	(9 (F G) I)	(N I L T D J (II JJ))
3	((PI) V (H J K))	(R YU (H KJ KL)	(U II OO LL PP (3 4 5))
4	(T (U U1 U2) (U4 U6 U8))	(4 6 (7 8 9))	(78 89 90 67 45)
5	(9 (() 8 88 888))	(H (J K L) (UJN))	(C B (N M I) (T Y U))
6	(T Y D E F (NL KM LM) JL)	(+ 2 3)	(*(+ 6 8) (- 70 8))
7	(TYPE PRINT DEL)	(H (H J O) (UJ N))	(READ SAVE LOAD (TXT))
8	(GOAL FUNCTOR CLAUSE (DATA BASE))	(2 5(5 4 6)8)	(L (K (K I)U))
9	(DOG (CAT) FOX ())	(RET GET PUT OUT IN)	(MOV ADD (MUL DEV))
10	(FIR SED (1 2 3) (5) ())	(H J U K (L M N) (D E L))	(4 5(6 7))
11	(PRIM SD FLAG () (GHG))	(1 56 98 52)	(T 2 3 4 Y H)
12	(H G (U J) (T R))	(2 1 (+ 4 5))	(TYPE CHAR REAL (H G))

Продолжение таблицы 1.2

Вариант	Исходные списки		
	1	2	3
13	(REM FFG HHJ (H)J G D)	(2 34 56 78 (7 8))	(UN Y LOOP)
14	(T (HJ (JH KL)) K)	(67 54 (8 9 0)(4 6))	(K F G H)
15	(3 (3 4 5 Y U)((T Y))	(G H (6 7 8) 8 9 0 7 6)	((5 T 7 Y H) U)
16	(Z X C S A D F)	((R)(30)(3) 23))	(U I 8 9 6 5 4 3 (1 2 3))
17	(V B N J H)	((Y U I)(H J K) (8) 78)	(df FG HJ K L (O 0 9))
18	(D F G H J K)	(1 2 3 4 5 6 (4 5) 4)	(ER RT TY 5 6 6 5)
19	(H G (2 3) 8 7 (T R))	(2 1(+ 4 5))	(TY PE CH AR RE AL (H G))
20	(RM F G H J (J G D))	(2 3 4 5 6 (7 8))	(U N Y L O O P)
21	(T HJ JH K L (K)	(6 7 5 4 (8 9 0)(4 6))	(K 2 T F G H)
22	(3 3 4 5 Y U)	(G (6 8) 8 9 7 6)	((5) T () 7 Y H U)
23	(Z 2 A D F)	((R) (30) 3 4 23)	(U I 8 9 6 5 4 3 (1 2 3))
24	(V B N J H)	(Y U I H (J K) (8) 78)	(df F K L (O 0 9))
25	(DF GH JK)	(1 2 5 6 (4 5) 4)	(ER RT TY 5 6 6 5)

Таблица 1.3. Номера элементов.

Вариант	Список 1	Список 2	Список 3
1	2	2	3
2	3	2	6
3	1	3	6
4	2	3	4
5	2	3	3
6	6	2	2
7	3	2	3
8	4	3	2
9	3	4	3
10	4	4	3
11	5	3	2
12	3	3	3

Продолжение таблицы 1.3

Вариант	Список 1	Список 2	Список 3
13	4	4	2
14	2	3	3
15	2	3	2
16	5	4	7
17	3	4	6
18	5	3	2
19	3	3	3
20	4	6	2
21	2	6	3
22	2	3	2
23	5	4	7
24	3	4	6
25	6	6	6

Задание 3.

Описать именованную функцию в соответствии с вариантом индивидуального задания в Таблице 1.4.

Таблица 1.4. Варианты индивидуального задания.

Вариант.	Задача.
1.	Есть два списка. Если первый элемент списка есть натуральное число, то возвратить второй список, иначе возвратить список из головы второго и хвоста первого.
2.	Написать функцию ,которая возвращает квадратный корень из аргумента, если аргумент является числом, последний элемент аргумента-списка, если аргумент – список, аргумент – иначе.
3.	Есть пять чисел. Написать функцию , формирующую список из максимального и минимального по модулю чисел, если минимальное и максимальное числа – целые, среднее арифметическое минимального и максимального чисел - иначе.
4.	Написать функцию, которая для аргумента-числа проверяет, является ли оно степенью двойки.
5.	Написать функцию, которая для заданных координат X1,Y1 и X2,Y2 возвращает расстояние между ними. Координаты могут иметь отрицательное значение.

Продолжение таблицы 1.4

Вариант.	Задача.
6.	Написать функцию, которая по заданному вещественному числу формирует список из трех элементов. Первый элемент – знак числа, второй – модуль числа, третий – ближайшее к нему целое число.
7.	Написать функцию, которая для трех аргументов-чисел проверяет, является ли третье число результатом возведения в степень первого числа с показателем, равным второму числу.
8.	Есть список. Найти сумму первого, третьего и седьмого элементов списка, если указанные элементы – числа, возвратить последний элемент списка – иначе.
9.	Написать функцию вычисления дискриминанта квадратного уравнения.
10.	Написать функцию, которая для аргумента-списка формирует список-результат по правилу : если первый и последний элементы списка-аргумента есть четные натуральные числа, то включить в список-результат первым элементом – квадрат последнего элемента исходного списка, вторым – четвертую степень первого; в противном случае сформировать список из первого и последнего элементов.
11.	Написать функцию, которая для аргумента-списка формирует список-результат по правилу : если первый и последний элементы списка-аргумента –символы, то сформировать список из первого и последнего элементов, в противном случае возвратить исходный список, из которого удален второй элемент.
12.	Есть два списка. Написать функцию формирующую список из трех подсписков. Первый подсписок содержит голову первого списка и третий элемент второго. Второй подсписок содержит второй элемент второго списка и последний элемент первого.
13.	Есть списки, к примеру, '(1 2 3 4 5) '(7 6 5 7). Если произведение первых элементов исходных списков есть положительное число, то объединить в результирующий список последние элементы. В противном случае построить список из последнего элемента первого списка и хвоста второго.
14.	Есть три числа. Построить список из кубов этих чисел, если все три числа – нечетные, возвратить сумму чисел – иначе.
15.	Есть список и некоторый объект. Написать функцию, возвращающую новый список, в котором объект замещает первый элемент списка, если первый элемент списка и объект являются атомами, последний элемент списка - иначе.

Продолжение таблицы 1.4

Вариант.	Задача.
16.	Дан список чисел. Написать функцию, возвращающую в случае первого четного элемента исходный список, в котором первые три числа возведены в квадрат, иначе - исходный список, в котором первые три числа возведены в куб.
17.	Даны два произвольных лисповских объекта. Написать функцию, которая возвращает список из суммы и произведения объектов, если оба объекта – числа, список из результатов проверки объектов на атомарность – иначе.
18.	Дан список. Написать функцию, которая в случае атомарности первого и последнего элементов списка возвращает список из указанных элементов, результат проверки третьего элемента на принадлежность к списочным объектам – иначе.
19.	Дан список lst. Написать функцию, которая для случая, когда первый элемент lst является списком, возвращает список из последнего элемента lst в качестве головы и первого элемента lst в качестве хвоста, список lst, из которого удален второй элемент – иначе.
20.	Написать функцию, которая возвращает квадрат аргумента-четного числа, натуральный логарифм аргумента-нечетного числа и результат проверки аргумента на принадлежность к списочному типу – иначе.
21.	Есть список lst, описывающий вызов арифметической функции. Написать функцию, которая а случае четности результата вычисления lst производит его проверку на положительность, а в противном случае выдает сам lst. Вычисление lst производить с помощью встроенной функции eval.
22.	Есть две лисповские формы : expr1 и expr2. Написать функцию, которая сравнивает результаты вычисления форм и в случае эквивалентности результатов вычисления формирует список из этих форм. В противном случае формируется список из результатов вычисления форм. Вычисление форм производить с помощью встроенной функции eval.
23.	Написать функцию, которая по заданному целому числу формирует список двух элементов. Первый элемент списка - это символьный атом, обозначающий знак числа, второй элемент – остаток от деления числа на 2.

Продолжение таблицы 1.4

Вариант.	Задача.
24.	Написать функцию, которая для заданных списков lst1 и lst2 возвращает список, содержащий их первые и последние элементы. Порядок следования элементов в результирующем списке определяется вторым элементом lst2 : если это число, то вначале идут первый и последний элементы lst1, иначе – первый и последний элементы lst2.
25.	Написать функцию, которая по двум числам формирует список из трех элементов. Первый элемент – это результат целочисленного деления чисел, второй есть результат умножения чисел, третий элемент есть символьный атом, обозначающий знак числа. **Функция должна содержать проверку делителя на ноль !**

Все результаты должны быть обоснованы.

1.6. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :
- формулировку цели и задач;
- результаты выполнения заданий по пунктам, обоснование полученных результатов и выбранных структур функций;
- выводы по проведенным экспериментам.

ЛАБОРАТОРНАЯ РАБОТА № 2

ОПИСАНИЕ ПРОСТЕЙШИХ РЕКУРСИВНЫХ ФУНКЦИЙ В ЯЗЫКЕ muLISP.

Цель работы.

Целью работы является изучение основных правил написания рекурсивных функций в функциональном языке.

Основные задачи :

- На примере функций в muLisp'e научиться формулировать условие завершения рекурсии, описывать формирование результата функции и новых значений аргументов для рекурсивного вызова;
- Получить практические навыки работы со списочными структурами в muLisp'e.

2.1. Краткие сведения о рекурсивных функциях.

Определение. Функция называется рекурсивной, если в ее теле содержится вызов самой этой функции.

Принято различать рекурсию по значению и рекурсию по аргументам.

В случае рекурсии по значению сам рекурсивный вызов непосредственно вычисляет значение функции. Пример - принадлежность объекта списку.

В случае рекурсии по аргументам в качестве значения функции возвращается значение некоторой другой функции и рекурсивный вызов участвует в вычислении аргументов этой функции. Пример – нахождение суммы элементов списка.

Рекурсия называется простой, если вызов функции встречается в некоторой ветви лишь один раз. Простой рекурсии в процедурном программировании соответствует обыкновенный цикл.

В ЛИСПе рекурсия - основной метод программирования. Как и в других языках программирования, описание рекурсивных функций требует особой тщательности. Во-первых, функция обязательно должна содержать хотя бы одно условие окончания рекурсии, во-вторых, значение аргумента при рекурсивных вызовах должно изменяться.

Часто условием окончания рекурсии является пустой список-аргумент. В процессе работы все элементы исходного списка обрабатываются последовательного до тех пор, пока не будет достигнут конец списка, т.е. пока список не пуст. Указанное типовое условие окончания рекурсивной обработки можно записать как :

((NULL lst) <действие>)

Второе требование к рекурсивной функции, касающееся изменения аргумента при рекурсивных вызовах, чаще всего обеспечивается рекурсивным вызовом с аргументом-хвостом списка.

Можно выделить следующие (основные) шаги при построении рекурсивной функции :

1) Определить количество и вид аргументов.
2) Определить характер результата.
3) Задать как минимум одно условие окончания рекурсии.
4) Определить формирование результата вызова функции.
5) Описать формирование новых значений аргументов для рекурсивного вызова.

Рассмотрим примеры описания рекурсивных функций.

Пример 1. Описать функцию, которая проверяла бы поэлементную эквивалентность двух списков.

Из задачи ясно, что формальных параметров у функции должно быть два и оба – списки. Назовем параметры lst1 и lst2.

Функция должна возвращать значение T, если длина списков одинакова и совпадают все элементы: первые, вторые, и т. д.

Определим, сколько и каких условий окончания рекурсии должно присутствовать в функции. Благополучный исход, при котором функция возвращает T, достигается при поэлементном совпадении lst1 и lst2. При этом должно выполниться условие :

(and (null lst1)(null lst2))

Два условия завершения рекурсивного вызова надо предусмотреть для неблагоприятного исхода, когда функция возвращает NIL. Это произойдет, во-первых, если один список короче другого :

(or (and (null lst1)(not (null lst2)))
(and (not (null lst1))(null lst2)))

И, во-вторых, если найдены несовпадающие элементы :

(NOT (EQUAL (CAR lst1)(CAR lst2)))

Если ни одно из условий окончания рекурсии не выполняется, то функция вызывает себя же уже для хвостов списка.

Описание функции на muLISPe :

```
(defun eqlists (lst1 lst2)
  ((and (null lst1)(null lst2)) T)
  ((or (and (null lst1)(not (null lst2)))
       (and (not (null lst1))(null lst2))
       (not (equal (car lst1)(car lst2)))) nil)
  (eqlists (cdr lst1)(cdr lst2)))
```

<u>Пример 2.</u> Суммирование элементов списка.

Количество аргументов : 1 – список.

Результат : число.

Условие окончания рекурсии – пустой список :

((null list) 0).

По достижению этого условия в качестве значения функции возвращается 0.

Формирование значения функции :

(+ (car list) сумма хвоста)

Сумма хвоста есть результат рекурсивного вызова проектируемой функции с хвостом списка в качестве аргумента.

Описание функции :

```
(defun sum (list)
   ((null list) 0)
```

```
(+ (car list)(sum (cdr list))))
```

Результат вызова (sum '(9 7 5 6 4)) есть 31.

Пример 3. Принадлежность объекта заданному списку.

Количество аргументов : 2 – произвольное s-выражение (obj) и список (lst).

Результат : значение T (истина), либо NIL (ложь).

Условий окончания рекурсии два :

((null lst) nil) - пустой список, искомый объект не найден,

((equal obj (car lst)) T) - объект найден.

В случае несовпадения искомого объекта и головы списка результирующее значение вычисляется путем вызова проектируемой функции с хвостом списка в качестве второго аргумента. Полное описание функции :

```
(defun member (obj lst)
    ((null lst) nil)
    ((equal obj (car lst)) T)
    (member obj (cdr lst)))
```

Результатом вызова (member 7 '(1 2 3 4)) есть nil.

Результатом вызова (member 3 '(1 2 3 4)) есть T.

Пример 4. Объединение списков.

Рассмотри три варианта решения данной задачи.

Вариант 1.

Количество аргументов : 2 – списки lst1 и lst2.

Результат : список.

Условий окончания рекурсии три :

((and (null lst1)(null lst2)) nil) - оба списка пустые,

((null lst1) lst2) - первый список пустой,

((null lst2) lst1) - второй список пустой.

Значение функции вычисляется путем присоединения головы первого списка в качестве головы к результату вызова проектируемой функции с хвостом первого списка в качестве первого аргумента и вторым списком в качестве второго аргумента.

Полное описание первого варианта функции объединения списков :

```
(defun append (lst1 lst2)
    ((and (null lst1)(null lst2)) nil)
    ((null lst1) lst2)
    ((null lst2) lst1)
    (cons (car lst1)(append (cdr lst1) lst2)))
```

Результат вызова (append '(1 2 3 4) '(5 6 7 8)) есть список (1 2 3 4 5 6 7 8).

Рассмотрим варианты более компактной записи условия окончания рекурсии.

Вариант 2.
Количество аргументов : 2 – списки lst1 и lst2.
Результат : список.
Условий окончания рекурсии два :
((null lst1) lst2) - первый список пустой,
((null lst2) lst1) - второй список пустой.
Результат функции формируется аналогично варианту 1.
Полное описание функции :

```
(defun append (lst1 lst2)
    ((null lst1) lst2)
    ((null lst2) lst1)
    (cons (car lst1)(append (cdr lst1) lst2)))
```

Вариант 3.
Количество аргументов : 2 - списки lst1 и lst2.
Результат : список.
Условие окончания рекурсии :
((null lst1) lst2).
Результат функции формируется аналогично варианту 1.
Полное описание функции :

```
(defun append (lst1 lst2)
    ((null lst1) lst2)
    (cons (car lst1)(append (cdr lst1) lst2)))
```

2.2. Задание на лабораторную работу.

1) Ознакомиться с основными правилами описанием рекурсивных функций в Лиспе. Выполнить Примеры 1-4.
2) Описать функцию в соответствии со своим вариантом задания из Таблицы 2.1, вариант выдает преподаватель.

Таблица 2.1. Варианты заданий.

Вариант.	Задача.
1.	Описать функцию, которая для заданного списка lst формирует список-результат путем объединения результата реверсирования lst, результата реверсирования хвоста lst, результата реверсирования хвоста хвоста lst и так далее. Пример : для списка ‘(1 2 3 4 5 6) результатом будет : ‘(6 5 4 3 2 1 6 5 4 3 2 6 5 4 3 6 5 4 6 5 6).
2.	Есть список lst и два произвольных лисповских объекта obj1 и obj2. Описать функцию, которая в списке lst заменяет все вхождения объекта obj1 объектом obj2.

Продолжение таблицы 2.1

Вариант.	Задача.
3.	Описать функцию, которая находила бы сумму всех числовых элементов списка с учетом наличия подсписков. Пример : для списка ‘(1 ((2 3) 4) 5 6) результатом будет 21.
4.	Описать функцию, которая на основе списка чисел формирует список-результат следующим образом : первый элемент есть произведение элементов списка, второй – произведение элементов хвоста, третий – произведение элементов хвоста хвоста и так далее. Пример : для списка ‘(1 2 3 4 5 6) результатом будет : ‘(720 720 360 120 30 6).
5.	Реализовать функцию включения объекта на заданное место в списке (нумерация элементов – от начала списка).
6.	Реализовать функцию, которая в исходном списке заменяет все элементы-списки результатами их реверсирования. Реверсирование производить на всех уровнях вложения. Пример : для списка ‘(1 ((2 3) 4) 5 6) результатом будет : ‘(1 (4 (3 2)) 5 6).
7.	Реализовать функцию, которая выдавала бы элемент списка по заданному номеру с конца.
8.	Дан список lst и число n. Реализовать функцию, которая удаляет все i+n – е элементы списка.
9.	Даны списки lst1 и lst2. Реализовать функцию, которая удаляет из lst1 все элементы-списки, которые соответствуют тому же множеству, что и lst2. Пример : для списков : lst1=’(1 (2 2 3) 4 (3 2 3) 5), lst2=’(3 2 3 2) результатом будет ’(1 4 5).
10.	Реализовать функцию, возвращающую T при идентичности порядка расположения одинаковых атомов в исходных списках.
11.	Реализовать функцию, меняющую местами первый и последний элементы исходного списка.
12.	Описать функцию, которая, выдавала бы атомарный злемент списка по заданному номеру n, считая от начала. Пример : для списка ‘((2) (3) 4 5 a (e r) g) и n=3 результатом будет a.
13.	Написать функцию подсчета числа элементов-списков исходного списка на всех уровнях вложения.
14.	Описать функцию, которая создавала бы список только из числовых элементов списка-аргумента. Список может содержать подсписки произвольной глубины.
15.	Опишите функцию, которая из исходного списка формирует список, содержащий только символьные атомы.
16.	Описать функцию, которая вставляла бы на заданное место элементы второго списка-аргумента.

Продолжение таблицы 2.1

Вариант.	Задача.
17.	Описать функцию, аргументами которой являются два списка, а результатом список, содержащий элементы первого списка, не принадлежащие второму списку.
18.	Описать функцию, которая в заданном списке заменяет все элементы-списки значениями сумм входящих в них числовых элементов с учетом вложенных подсписков.
19.	Описать функцию, которая в заданном списке заменяет все элементы-списки значениями количества входящих в них элементов-символов с учетом вложенных подсписков.
20.	Описать функцию, которая для заданного списка проверяет, является ли он отсортированным по возрастанию (убыванию).
21.	Опишите функцию, аргументами которой являются два множества, а результатом - множество, содержащее элементы, принадлежащие только одному из исходных множеств.

2.3. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :

- формулировку цели и задач;
- результаты выполнения заданий по пунктам, обоснование выбранных структур функций, включая условие окончания рекурсии в каждом случае и формирование новых значений аргументов при рекурсивном вызове;
- выводы по проделанной реализации.

ЛАБОРАТОРНАЯ РАБОТА № 3

МЕТОДЫ РАЗРАБОТКИ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ.

Цель работы.

Целью работы является изучение основных методов разработки функциональных программ с позиций Строго Функционального Языка.

Основные задачи :

- Освоить приемы нисходящего и восходящего проектирования функциональных программ;

- Научиться выделять основные и вспомогательные функции с учетом разбиения задачи на подзадачи;
- Овладеть приемами использования накапливающих параметров во вспомогательных функциях;
- Ознакомиться с упреждающим использованием результата вызова функции.

3.1. Краткие теоретические сведения.

Использование вспомогательных функций и применение накапливающих параметров являются основными методами разработки функциональных программ. Понятия основной и вспомогательной функции являются относительными. Одна и та же функция может использовать для вычисления своего значения другие функции из числа описанных в программе, но в то же время и ее могут вызывать как вспомогательную.

3.1.1. Нисходящее и восходящее проектирование.

Различают нисходящее и восходящее проектирование функциональных программ. Основным отличием нисходящего проектирования является решение задачи с использованием разработанных ранее функций в качестве вспомогательных.

Пример 1. Использование нисходящего проектирования для задачи преобразования списка в множество.

Рассмотрим вначале взаимные отличия списков и множеств.

Список :

- Упорядоченная последовательность : (1 2 3)≠(3 2 1);
- Один и тот же элемент может встречаться дважды.

Множество :

- Наличие отношения порядка не является обязательным;
- Каждый элемент встречается ровно один раз.

Аргумент : список.
Результат : множество.
Условие выхода из рекурсии : ((null lst) nil).

Вариант 1.

Генерация результата :

Включить (car lst) в множество, полученное из (cdr lst), из которого удалены все вхождения (car lst).

Предположим, что мы имеем функцию удаления всех вхождений объекта в список :

```
(defun delete (obj list)
        ((null list) nil)
        ((equal obj (car list))(delete obj (cdr list)))
        (cons (car list)(delete obj (cdr list))))
```

Будем использовать функцию delete для удаления из исходного списка вхождений повторяющихся элементов :

```
(defun list_set1 (list)
        ((null list) nil)
        (cons (car list)(list_set1 (delete (car list)(cdr list)))))
```

Вариант 2.

Генерация результата :

Если (car list) содержится в (cdr list), то возвратить в качестве результата множество, полученное из (cdr list). Иначе включить (car list) в множество, полученное из (cdr list). Для определения принадлежности объекта списку будем использовать разработанную нами ранее функцию member :

```
(defun member (obj list)
        ((null list) nil)
        ((equal obj (car list)) T)
        (member obj (cdr list)))

(defun list_set2 (list)
        ((null list) nil)
        ((member (car list)(cdr list))(list_set2 (cdr list)))
        (cons (car list)(list_set2 (cdr list))))
```

Пример 2. Использование восходящего проектирования для задачи объединения множеств.

Аргументы : 2 множества lst1 и lst2. Результат : множество.

Генерация результата : включить (car lst1) в множество-результат объединения (cdr lst1) и lst2. Для этого необходимо построить функцию, включающую заданный объект в список, если он там отсутствует :

```
(defun put (obj list)
        ((member obj list) list)
        (cons obj list))

; Функция объединения множеств
```

```
(defun unit (lst1 lst2)
        ((null lst1) lst2)
        (put (car lst1)(unit (cdr lst1) lst2)))
```

Путем использования функции put получаем еще один вариант функции преобразования списка в множество :

```
(defun list_set3 (list)
        ((null list) nil)
        (put (car list) (list_set3 (cdr list))))
```

Теперь рассмотрим вариант преобразования списка в множество для случая наличия в исходном списке элементов-списков.

```
; Функция сравнения множеств
(defun compare_sets (set1 set2)
        ((and (null set1)(null set2)) T)
        ((member (car set1) set2)
                (compare_sets (delete (car set1)(cdr set1))(delete (car set1) set2)))
        ((member_of_set (list_set4 (car set1)) (list_set4 set2))
                (compare_sets  (list_set4 (del_set (car set1)(cdr set1)))
                               (list_set4 (del_set (car set1) set2)))))

; Расширенная функция принадлежности множества другому множеству
(defun member_of_set (set1 set2)
        ((and (null set2)(not (null set1))) nil)
        ((and  (not (atom (car set2)))
                (compare_sets set1 (car set2))) T)
        (member_of_set set1 (cdr set2)))

; Функция удаления элемента-множества
(defun del_set (set1 set2)
        ((null set2) nil)
        ((compare_sets set1 (car set2))
            (del_set set1 (cdr set2)))
        (cons (car set2)(del_set set1 (cdr set2))))

(defun list_set4 (list)
        ((null list) nil)
; Очередной элемент списка является атомом
        ((atom (car list))
            (cons (car list)(list_set4 (delete (car list)(cdr list)))))
; Очередной элемент списка является списком
```

(cons (list_set4 (car list))(list_set4 (del_set (car list)(cdr list)))))

3.1.2. Использование накапливающих параметров.

Пример 3. Функция реверсирования списка.
Аргумент : список lst.
Результат : тот же список, переписанный в обратном порядке.
Условие окончания рекурсии : пустой список.
Генерация результата : (append (reverse (cdr lst))(cons (car lst) nil)).
Полное описание функции :

```
(defun reverse (lst)
    ((null lst) nil)
    (append (reverse (cdr lst))(cons (car lst) nil)))
```

Рассмотрим возможность снижения вычислительной сложности задачи путем уменьшения числа вызовов простейших базовых функций. Функция выполняется тем эффективнее, чем меньше число вызовов cons она предполагает. Исследуем зависимость числа N вызовов функции cons функцией reverse от числа n элементов реверсируемого списка.

Пусть дан список '(1 2 3 4 5). Вызываем (reverse '(1 2 3 4 5))
Шаг 1. 1-й аргумент append : '(2 3 4 5). Результат вызова cons : '(1) - 2-й аргумент append.
Шаг 2. 1-й аргумент append : '(3 4 5). Результат вызова cons : '(2) - 2-й аргумент append.
Шаг 3. 1-й аргумент append : '(4 5). Результат вызова cons : '(3) - 2-й аргумент append.
Шаг 4. 1-й аргумент append : '(5). Результат вызова cons : '(4) - 2-й аргумент append.
Шаг 5. 1-й аргумент append : '(). Результат вызова cons : '(5) - 2-й аргумент append. Достигнуто условие окончания рекурсии (пустой список).

Итого на этапе развертывания рекурсии получилось n, то есть 5 вызовов cons. Рассмотрим теперь свертывание рекурсии.
Посредством функции append конструируем список-результат из извлекаемых из стека результатов рекурсивных вызовов :

(append nil ‘(5)) → ‘(5) - 0 вызовов cons.
(append '(5) ‘(4)) → ‘(5 4) - 1 вызов cons.
(append '(5 4) ‘(3)) → ‘(5 4 3) - 2 вызова cons.
(append '(5 4 3) ‘(2)) → ‘(5 4 3 2) - 3 вызова cons.
(append '(5 4 3 2) ‘(1)) → ‘(5 4 3 2 1) - 4 вызова cons.

Число вызовов cons на этапе свертывания рекурсии соответствует сумме n-1 первых членов арифметической прогрессии : $\frac{(1+(n-1))*(n-1)}{2}=\frac{n*(n-1)}{2}$.

Общее число вызовов cons при работе функции reverse составляет $n+\frac{n*(n-1)}{2}=\frac{2*n+n^2-n}{2}=\frac{n^2+n}{2}=\frac{n*(n+1)}{2}$, т.е. $N=\frac{n*(n+1)}{2}$.

Рассмотрим теперь вариант функции реверсирования, который предполагает $N=n$ вызовов cons. Данный вариант функции реверсирования использует вспомогательную функцию с накапливающим параметром, который передается в качестве результата в основную функцию по завершению рекурсии.

```
(defun reverse (lst)
        (rev lst nil))
```

Накапливающий параметр

```
(defun rev (lst1 lst2)
        ((null lst1) lst2)
        (rev (cdr lst1)(cons (car lst1) lst2)))
```

Функция rev работает следующим образом :

lst1	lst2
(1 2 3 4)	()
(2 3 4)	(1)
(3 4)	(2 1)
(4)	(3 2 1)
()	(4 3 2 1)

Возвращается в качестве результата

Пример 4. Нахождение суммы и произведения элементов списка.

Дан список. Сформировать список, содержащий два элемента : сумма и произведение элементов списка.

```
; Суммирование элементов списка
(defun sum (lst)
        ((null lst) 0)
        (+ (car lst)(sum (cdr lst))))
; Произведение элементов списка
(defun mult (lst)
        ((null lst) 1)
        (* (car lst)(mult (cdr lst))))
; Формирование списка из суммы и произведения
; элементов исходного списка
; Первый вариант решения - с применением sum и mult
; в качестве вспомогательных функций.
(defun s_m1 (lst)
```

```
        (list (sum lst)(mult lst)))
; Вариант функции формирования списка "сумма и произведение"
; с применением накапливающих параметров
(defun s_m2 (lst)
        (sm lst 0 1))
; Накопление результата
(defun sm (lst s p)
        ((null lst)(list s p))
        (sm (cdr lst)(+ (car lst) s)(* (car lst) p)))
; Еще один вариант решения задачи.
(defun s_m3 (lst)
        ((null lst)(list 0 1))
        (list  (+ (car lst)(car (s_m3 (cdr lst))))
              (* (car lst)(cadr (s_m3 (cdr lst))))))
```

3.2. Задание на лабораторную работу.

ВНИМАНИЕ ! в лабораторной работе можно использовать только средства строго функционального языка программирования. Иными словами при описании функций нельзя использовать функции SET, SETQ, SETF и циклы. Все функции должны быть разработаны самостоятельно.

Задание 1.

Написать программу сортировки списка методом Шелла. Вычисление последовательности шагов сортировки производится в соответствии с вариантом в Таблице 3.1.

Таблица 3.1. Вычисление шага сортировки Шелла.

Вариант.	Вычисление последовательности шагов.
1 – 8.	Методом Р. Седжвика. Длина очередного s -го шага : $inc[s]=\begin{cases}9*2^{s}-9*2^{s/2}+1, \text{ если } s \text{ четно}\\ 8*2^{s}-6*2^{(s+1)/2}+1, \text{ если } s \text{ нечетно}\end{cases}$
9 – 16.	Методом, предложенным Дональдом Кнутом : $h_{k-1}=3*h_k+1$, $h_t=1$, где h_i - шаг сортировки, $t=[\log_3 n]-1$ - число шагов сортировки, n – длина списка.
17 - 25.	Методом, предложенным Дональдом Кнутом : $h_{k-1}=2*h_k+1$, $h_t=1$, где h_i - шаг сортировки, $t=[\log_2 n]-1$ - число шагов сортировки, n – длина списка.

Задание 2.

Написать программу сортировки списка в соответствии с вариантом в Таблице 3.2.

Таблица 3.2. Методы сортировки списков.

Вариант.	Реализуемый метод сортировки.
1.	Сортировка простыми включениями.
2.	Сортировка бинарными включениями.
3.	Сортировка методом прямого выбора.
4.	Сортировка методом пузырька.
5.	Шейкер-сортировка.
6.	Сортировка Хоара.

Сравнить эффективность реализованной сортировки и реализованного в ***Задании 1*** варианта сортировки Шелла.

Задание 3.

Написать программу объединения двух отсортированных списков в один. При этом порядок сортировки в списке-результате должен сохраняться.

Задание 4.

Написать программу в соответствии с заданием из Таблицы 3.3.

Таблица 3.3. Вариант индивидуального задания.

Вариант.	Задание.
1, 13, 24	Написать программу, возвращающую T, если lst2 является подсписком lst1 глубины N. Элементами списка могут быть атомы и (или) списки любой глубины вложения.
2, 14	Написать функцию, вычисляющую сумму элементов-чисел на каждом уровне исходного списка. Рекомендуется следующая форма результата : ((1 <сумма числовых элементов на первом уровне>)(2 <на втором>)..) Пример : для списка (a (b (4 (2 e (3) k 15) e 5) 7)) результатом будет список : ((1 0)(2 7)(3 9)(4 17) (5 3)).
3, 15	Написать программу, возвращающую список, содержащий информацию о количестве подсписков на каждом уровне вложенности : ((<уровень><количество подсписков>)...).

Продолжение таблицы 3.3

Вариант.	Задание.
4, 16	Написать программу, которая в исходном списке заменяет все элементы-символы соответствующими им ASCII-кодами [9]. Список может содержать подсписки произвольной глубины вложения.
5, 17	Написать программу, которая в исходном списке заменяет все элементы-целые числа остатками от их деления на 2. Список может содержать подсписки произвольной глубины вложения.
6, 18	Написать функцию, генерирующую все циклические перестановки списка. Элементами списка являются списки. Пример : ((a b)(c d)) дает (((a b)(c d))((b a)(c d))((a b)(d c))..).
7, 19	Функция должна возвращать список позиций вхождения и глубин нахождения списка lst2 в список lst1.
8, 20	Есть список, написать программу, возвращающую максимальную глубину списка.
9, 21	Написать функцию, удаляющую из исходного списка подсписки заданной глубины.
10, 22	Заданы глубина подсписка, позиция и s-выражение. Включить s-выражение во все имеющиеся подсписки заданной глубины и на заданную позицию.
11, 23	Заданы глубина подсписка и позиция. Удалить из всех имеющихся подсписков заданной глубины элементы, находящиеся на указанной позиции.
12, 24	Написать программу сортировки списка методом Хоара.

3.3. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :

- формулировку цели и задач;
- результаты выполнения заданий по пунктам, обоснование выбранных структур функций, включая условие окончания рекурсии в каждом случае и формирование новых значений аргументов при рекурсивном вызове;
- выводы по проделанной реализации.

ЛАБОРАТОРНАЯ РАБОТА № 4

ЛОКАЛЬНЫЕ ОПРЕДЕЛЕНИЯ.

Цель работы.

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

Основные задачи :

- Изучение применения нисходящей и восходящей рекурсии при написании рекурсивных функций с использованием локальных определений;
- Проведение сравнительного анализа возможностей локальных определений LET и LAMBDA по организации вычислений в рекурсивных программах.

4.1. Краткие теоретические сведения.

Локальные определения относятся к управляющим структурам Лиспа и обеспечивают :

1) Сокращение количества рекурсивных вызовов функций;
2) Удобочитаемость программы.

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.

Пример. Построение копии списка в памяти виртуальной Лисп-машины.

Концепция СФЯ не предполагает наличия специальных функций изменения списочных структур. Созданные структуры никогда не изменяются. Структуры, значения которых уже не нужны, не уничтожаются. В СФЯ для ранее созданных структур допускается :

– Анализ;
– Расчленение;
– Копирование.

Участок оперативной памяти, в котором работает Лисп-система, разбивается в представлении последней на списочные ячейки (виртуальные Лисп-ячейки). Списочная ячейка состоит из двух частей, полей CAR и CDR (Рис. 1).

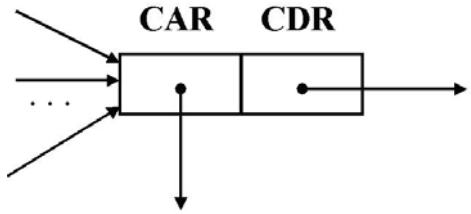

Рис.1 Ячейка памяти виртуальной Лисп-машины

Каждое из полей содержит указатель. Указатель может ссылаться на другую списочную ячейку или на некоторый другой списочный объект, например, атом. На каждую списочную ячейку может ссылаться произвольное количество указателей. Указателем списка является указатель на первую ячейку списка. На ячейку могут указывать :

– Поля CAR и CDR других ячеек;
– Указатели на значения у символов.

Графически одноуровневый список представляется последовательностью ячеек, связанных друг с другом через указатели в правой части ячеек (рис. 2).

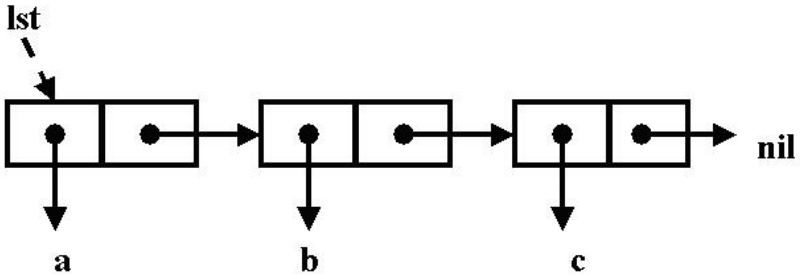

Рис.2 Структура одноуровневого списка

Системные свойства символа включают указатель на значение. Побочным эффектом функции присваивания SETQ является замещение указателя в поле значения символа. Например, вызов (setq lst '(a b c)) создает в качестве побочного эффекта изображенную на рис.2 штриховую стрелку.

Правое поле последней ячейки списка на рис.2 в качестве признака конца списка ссылается на другой список, т.е. атом nil. Графически ссылку на пустой список часто изображают в виде перечеркнутого поля. Указатели из полей CAR ячеек списка ссылаются на структуры, являющиеся элементами списка. Если список содержит подсписки, то на месте атомов будут находиться первые ячейки подсписков (рис.3).

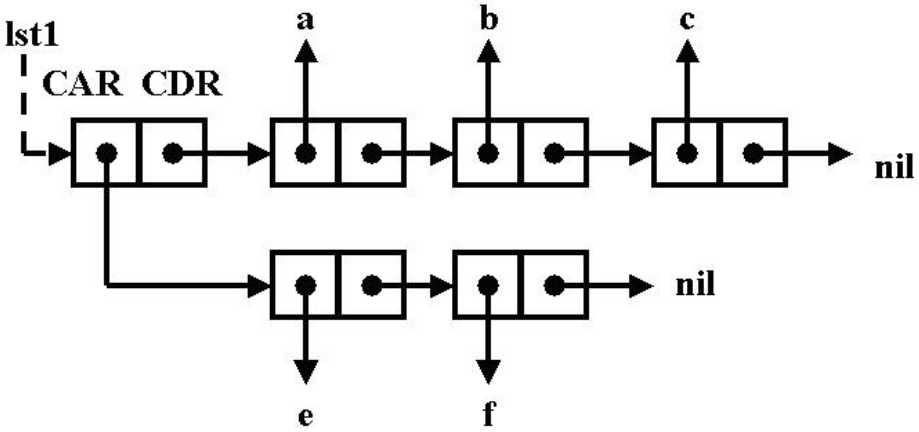

Рис.3 Структура многоуровневого списка-результата вызова (setq lst1 '((e f) a b c))

Логически идентичные атомы содержатся в памяти виртуальной Лисп-машины один раз, однако логически идентичные списки могут быть представлены различными Лисп-ячейками. Рассмотрим две последовательности вызовов.

Вначале вызываем (setq lst2 '((b c) a b c), результат представлен на рис.4.

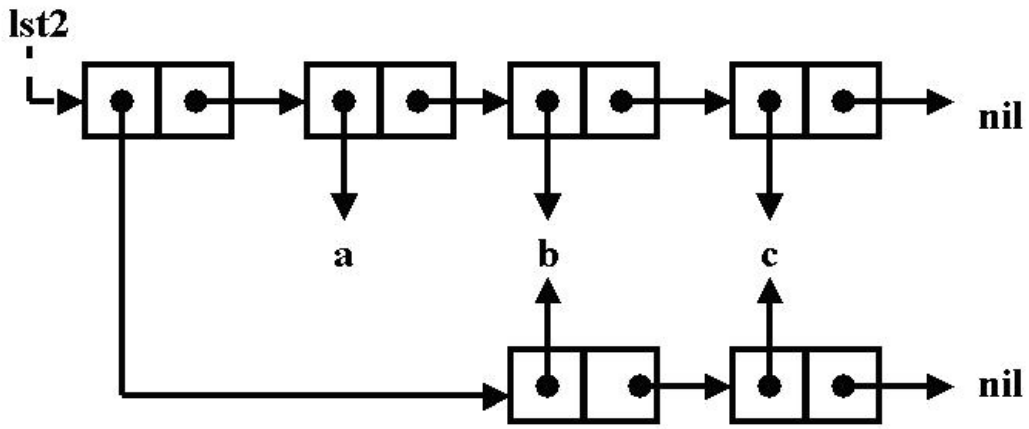

Рис.4 Физическая структура многоуровневого списка-результата вызова (setq lst2 '((b c) a b c)

Теперь последовательно выполняем :
(setq bc '(b c))
(setq abc (cons 'a bc))
(setq lst3 (cons bc abc)),
результат представлен на рис.5.

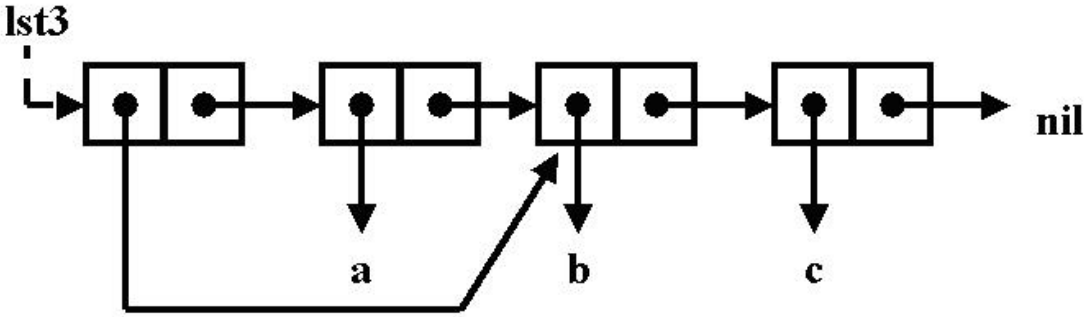

Рис.5

Логическая структура списка представляется в скобочной нотации и всегда имеет форму дерева, в то время как физическая структура, изображаемая графически совокупностью Лисп-ячеек, в общем случае есть ациклический граф.

Новая виртуальная списочная ячейка создается всякий раз при вызове CONS. Содержимым левого поля новой ячейки становится содержание первого аргумента вызова, а правого - значение второго аргумента (рис. 6). Применение функции CONS не изменяет значения головы и хвоста.

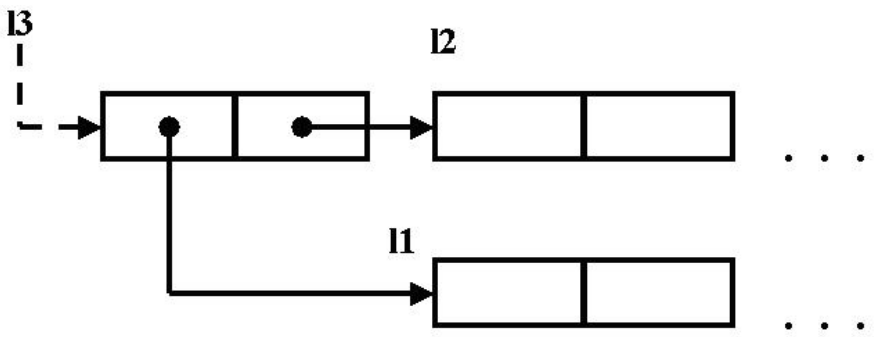

Рис.6 Инициализация Лисп-ячейки при вызове (setq l3 (cons l1 l2))

Идея функции копирования списка в памяти виртуальной Лисп-машины основывается на том, что конструктор CONS каждый раз при вызове инициализирует новую Лисп-ячейку. За идейную основу построения данной функции возьмем работу ранее рассмотренной нами функции APPEND – объединения двух списков. Фактически наша функция должна строить в памяти виртуальной Лисп-машины копию списка-аргумента подобно тому, как APPEND при вызове создает копию своего первого аргумента (рис. 7).

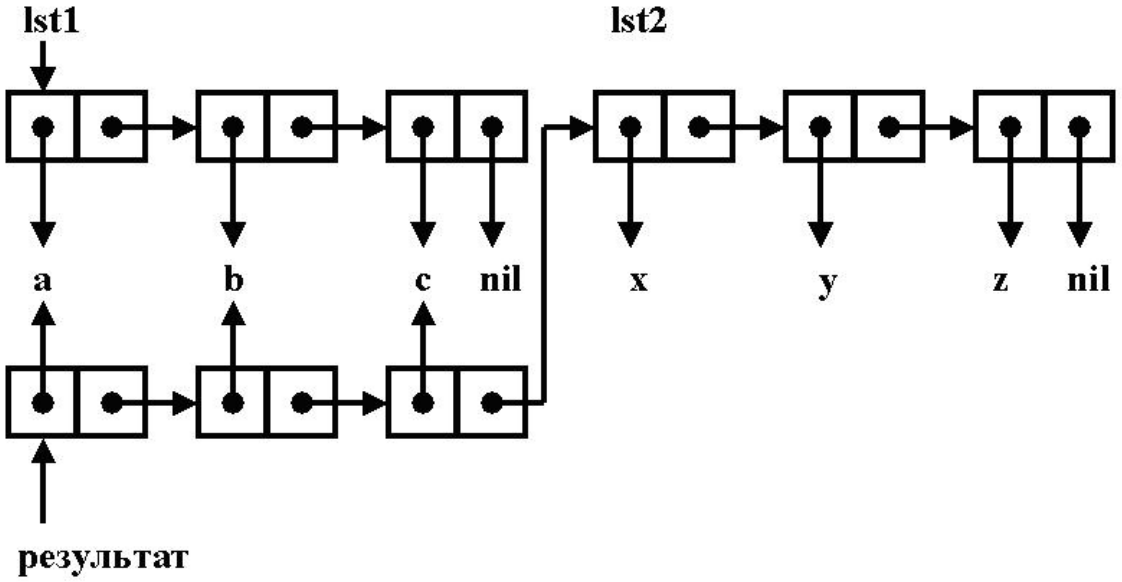

Рис.7 Копирование первого аргумента (списка lst1) функции APPEND при вызове (append lst1 lst2)

Основываясь на изложенных выше принципах работы CONS, получаем приведенный ниже вариант искомой функции построения копии списка в памяти виртуальной Лисп-машины.

```
; Функция копирования списка
(DEFUN DCOPY (LAMBDA (LST)
```

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

```
; Пример использования восходящей рекурсии
; для решения задачи реверсирования списка

(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)) ))))
```

4.2. Задание на лабораторную работу.

Задание 1.

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

Задание 2.

Разработать программу символьного дифференцирования. Рассмотреть варианты решения задачи с применением локальных определений LAMBDA и LET.

Теоретические сведения.

Символьным дифференцированием в математике называется операция преобразования одного арифметического выражение в другое выражение, которое рассматривается как производная исходного. Пусть U – некоторое арифметическое выражение, которое содержит переменную x. Производная от U по x записывается в виде dU/dx и определяется рекурсивно с помощью некоторых правил преобразования, применимых к U. В качестве

условий окончания рекурсии следует рассматривать производную от константы и самой переменной x :

$dc/dx \to 0$

$dx/dx \to 1$

Аналогичным образом мы можем описать оставшуюся часть таблицы производных :

$d(U+V)/dx \to dU/dx+dV/dx$

$d(U-V)/dx \to dU/dx-dV/dx$

$d(cU)/dx \to c(dU/dx)$

$d(U*V)/dx \to U*(dV/dx)+V*(dU/dx)$

$d(U/V)/dx \to d(U*V^{-1})/dx$

$d(U^c)/dx \to c*U^{c-1}(dU/dx)$

$d(ln(U))/dx \to U^{-1}(dU/dx)$

Задание 3.

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

Задание 4.

Реализовать программу-простейший интерпретатор лисповских программ. На вход интерпретатора подается текст, который может быть интерпретирован как вызов или суперпозиция функций Лиспа, пример : '(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;

Задание 5.

Дополнить интерпретатор из задания 4 в соответствии с вариантом индивидуального задания из Таблицы 4.1.

Таблица 4.1. 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 целого числа.

4.3. 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.

ЛАБОРАТОРНАЯ РАБОТА № 5

ФУНКЦИОНАЛЫ.

Цель работы.

Целью лабораторной работы является изучение отображающих и применяющих функционалов.

5.1. Краткие теоретические сведения.

Определение. Аргумент, значением которого является функция, называют в функциональном программировании функциональным аргументом.

В роли функционального аргумента может выступать :

- имя функции, с которым связано описание;
- Лямбда-выражение '(lambda (<список формальных параметров>) <тело лямбда-выражения>);
- Всякий лисповский объект, значением которого является функция.

Следует отметить, что функциональным аргументом может быть только "настоящая" функция. Специальные формы, такие как QUOTE, SETQ и макросы для этих целей не подходят.

Определение. Функционалом называется функция, аргумент которой может быть интерпретирован как функция.

Определение. Функционалом с функциональным значением называется функционал, вызов которого возвращает в качестве результата новую функцию. Причем в построении этой функции могут использоваться функции, получаемые функционалом в качестве аргументов.

Определение. Аппликативным или применяющим функционалом называется функция, которая позволяет применять функциональный аргумент к его параметрам.

В Лиспе определено три применяющих функционала :

- (FUNCALL <функциональный аргумент> аргументы) - вызывает функцию с аргументами;
- (APPLY <функциональный аргумент> список аргументов) - применяет функцию к списку аргументов;
- (EVAL <любое лисповское выражение>) – встроенный интерпретатор, вычисляет значение выражения Лиспа.

APPLY есть функция двух аргументов, из которых первый представляет собой функцию, которая применяется к элементам списка - второго аргумента : (apply <функция> <список>).

Пример : (apply '+ '(2 3)) дает в качестве результата 5.

FUNCALL по своему действию аналогичен APPLY, но аргументы для вызываемой функции он принимает не списком, а по отдельности : (funcall <функция> <арг1> : <аргN>).

Примеры : (funcall '+ 2 3) дает в качестве результата 5.
(funcall '+ '(2 3)) дает в качестве результата (2 3).

Различие между APPLY и FUNCALL состоит в обязательности списочного представления аргументов у APPLY. FUNCALL аналогичен по действию APPLY, но аргументы для вызываемой функции принимаются не списком, а по отдельности.

Рассмотрим пример использования применяющих функционалов.

Задача. Требуется описать функцию, которая выполняет заданное действие над каждым элементом списка и объединяет результаты в список.

Решение.

```
; Описание функционала :
(defun mapping (fun lst)
        ((null lst) nil)
        (cons (funcall fun (car lst))
        (mapping fun (cdr lst))))
; Примеры возможных действий над эелементами списка.
; Функция увеличения элемента на 1 :
(defun p1 (obj)
        (+ 1 obj))
; Остаток от деления на 2 :
(defun m2 (obj)
        (mod obj 2))
```

Примеры вызовов :
(mapping p1 '(2 3 4)) дает в качестве результата '(3 4 5)
(mapping m2 '(2 3 4)) возвращает '(0 1 0)

Определение. Отображающие функционалы Лиспа или MAP-функционалы есть функции, отображающие некоторым образом список (последовательность) в новую последовательность или порождают побочный эффект, связанный с этой последовательностью.

Имена MAP - функций начинаются на MAP, их вызов имеет вид :

(MAPx fn l1 l2 ... lN).

Здесь l1 ... lN - списки, а fn - функция от N аргументов.

Как правило, MAP - функция применяется к одному аргументу-списку, то есть fn - функция от одного аргумента :

(MAPx fn список).

В Лиспе определено шесть отображающих функционалов (рис. 8).

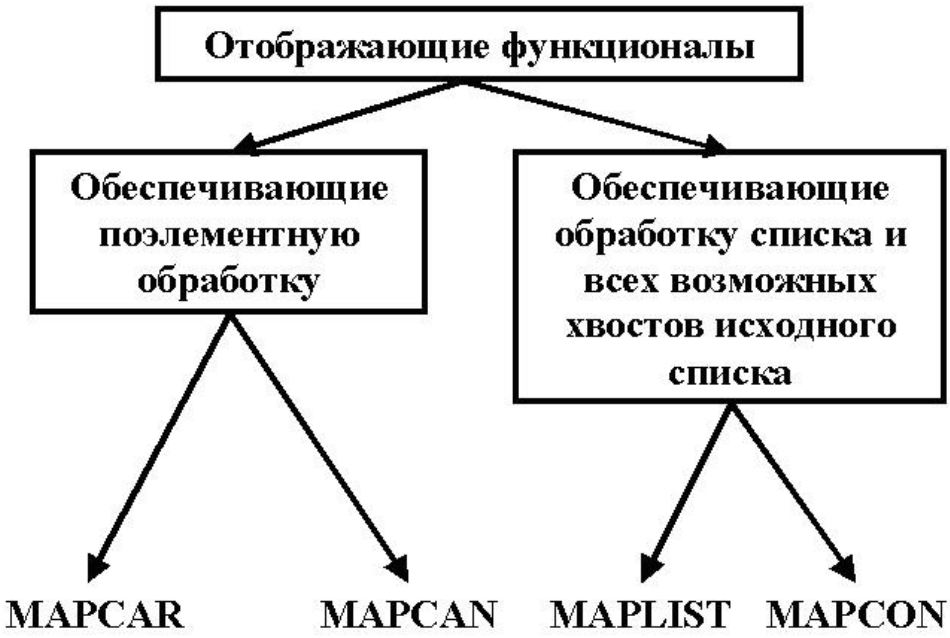

Рис.8 Виды отображающих функционалов

Функционал MAPCAR обеспечивает реализацию функционального аргумента над всеми элементами списка и объединяет результаты в список.

Пример : (mapcar 'list '(a b c)) дает '((a)(b)(c)).

Функционал MAPCAN аналогичен MAPCAR, отличие состоит в объединении списков-результатов с использованием структуроразрушающей псевдофункции NCONC.

Пример : (mapcan 'list '(a b c)) дает '(a b c).

Функционал MAPLIST обеспечивает реализацию функционального аргумента над списком и всеми его хвостовыми частями.

Пример : (maplist 'list '(a b c)) дает '(((a b c))((b c))((c))).

Функционал MAPCON аналогичен MAPLIST, отличие состоит в использовании структуроразрушающей псевдофункции NCONC.

Пример : (mapcon 'list '(a b c)) дает '((a b c)(b c)(c)).

Псевдофункционалы MAPC и MAPL используются для получения побочного эффекта. Аналогичны по действию MAPCAN и MAPCON и отличаются тем, что не объединяют и не собирают результаты, а теряют их.

Пример.
(mapc 'list '(a b c))
(mapl 'list '(a b c))
В обоих случаях будет возвращен список '(a b c).

Рассмотрим пример использования применяющих функционалов.

Задача. Требуется описать преобразование локального определения LET в локальное определение LAMBDA.

Решение.

Зададим вначале список соответствия формальных и фактических параметров :

(setq letlist '((form1 fact1)(form2 fact2)(form3 fact3)))

Вызов (mapcar 'car letlist) дает список формальных параметров : (form1 form2 form3).

Вызов (mapcar 'cadr letlist) дает список фактических параметров : (fact1 fact2 fact3).

Результирующее локальное лямбда-определение получается следующим образом :

(list 'lambda (mapcar 'car letlist) 'body)

Как результат получаем : (lambda (form1 form2 form3) body)

Получение лямбда-вызова на основе описанного преобразования локального определения LET в локальное определение LAMBDA происходит следующим образом :

(cons (list 'lambda (mapcar 'car letlist) 'body) (mapcar 'cadr letlist))

В результате получаем :

((lambda (form1 form2 form3) body) fact1 fact2 fact3)

5.2. Задание на лабораторную работу.

Задание 1.

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 5.1. Вариант задания 1.

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.

Продолжение таблицы 5.1

Вариант.	Задание.
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.

Задание 2.

Дана фраза русского языка. Написать программу, которая разбивает каждое слово фразы на слоги. Для выполнения этого и последующего задания рекомендуется воспользоваться версией интерпретатора mulisp_2.com.

Теоретические сведения.

Слог - это звук или несколько звуков, произносимых, как правило, одним толчком выдыхаемого воздуха. С л о г о о б р а з у ю щ и м и являются гласные звуки, поэтому в слове обычно выделяется столько слогов, сколько в нем гласных звуков: *ра-ке-та*, *де-ле-га-ци-я*.

Артикуляция – совокупность работ произносительных органов человека при образовании звуков речи.

В русском литературном языке деление на слоги опирается на п р и н ц и п в о с х о д я щ е й з в у ч н о с т и . Это обозначает, что звуки в слоге (незаконченом) располагаются от наименее звучного к наиболее звучному. Если звучность условно обозначить цифрами, получится следующее: 3 - гласный звук, 2 - сонорный согласный звук, 1 - остальные (шумные) согласные звуки.

Артикулярно согласные отличаются наличием шумообразующей преграды в надгортанных полостях органов речи; они точно локализованы и имеют определенно фиксируемый фокус образования. Кроме шумового, при артикуляции согласных может участвовать и тоновый источник звука - гортань, которой благодаря периодическим колебаниям голосовых связок возникает тон голоса. В зависимости от наличия при артикуляции согласных только первого источника или двух, различают глухие согласные (пример – русские *п*, *т*, *с*, *х*) и звонкие (пример – русские *б*, *д*, *з*, *л*, н). Среди звонкие выделяется группа сонорных согласных, или сонантов (например, *л*, *р*, *м*, *н*, *й*), отличающихся от шумных согласных (как звонких, так и глухих) наличием четкой формантной структуры. Последнее сближает сонанты с гласными, однако их отличает меньшая общая энергия. К сонорным принадлежат, в частности, носовые. При их артикуляции нёбная занавеска опускается, благодаря чему включается носовой резонатор.

Примеры разделения на слоги по принципу восходящей звучности : *кни-га* (1 2 3 - 1 3), *и-на-че* (3 - 2 3 - 1 3), *по-ло-тно* (1 3 - 2 3 - 1 2 3).

Трудности при разделении слова на слоги могут возникнуть в случаях соседства согласных. При этом в русском литературном языке, опирающемся на московское произношение, разделение на слоги осуществляется с учетом следующего :

- Если на границе слогов рядом оказались два шумных или два сонорных звука (кроме [j]), они относятся к последующему гласному : *у-шко*, *звезда*, *во-лна*.
- Если в сочетании согласных первый [j], он всегда отходит к предшествующему гласному: *тай-на*, *чай-ка*.
- В сочетании согласных, первым из которых является сонорный, а вторым - шумный, сонорный может отходить к предшествующему гласному: *крон-штейн*, *Вол-га*.

Следует иметь в виду, что деление слов на слоги не всегда совпадает с делением слов для переноса. Хотя в большом числе случаев перенос

осуществляется в месте фонетического слогораздела (*мо-ло-ко*, *лам-па*), в ряде случаев слог для переноса и фонетический слог могут не совпадать.

Пример. Слово *“идея”* имеет три слога. Тем не менее, перенести это слово нельзя ввиду того, что нельзя одну букву оставить в строке или перенести на следующую строку. Слово *“пёстрый”* можно перенести тремя способами (*пе-стрый*, *пес-трый*, *пест-рый*), но на слоги оно делится только одним (*пе-стрый*), смотри выше.

Задание 3.

"Язык сплетника" (см. [1], т.2,с.203-216). Есть ключевое слово, например, "сплетня". Слово переводится на язык сплетника путем отделения первого слога в переводимом и ключевом слове (например, *сло-во* и *сплетня*) 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 русского языка на заданный таким образом "тайный" язык.

Задание 4.

Написать программу в соответствии с заданием из Таблицы 5.2.

Таблица 5.2. Вариант задания 4.

Вариант.	Задание.
1,12,14,20.	“Тайные языки”. 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 пятибуквенного конца слова ([7], с. 176-185). Если слово содержит менее пяти букв, то замену не производить.

Продолжение таблицы 5.2

Вариант.	Задание.
3,8,16,22.	“Частотный словарь” (см. [2], с.203). Написать программу, которая по заданному тексту строит список пар : (<слово> <частота встречаемости в тексте>).
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, [8], с. 78-109) 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.

5.3. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :
- формулировку цели и задач;
- описание процесса разработки программ;
- выводы по проделанной реализации.

ЛАБОРАТОРНАЯ РАБОТА № 6

СОЗДАНИЕ ПРОСТЕЙШЕГО ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА В СРЕДЕ Microsoft muLISP.

Цель работы.

Целью лабораторной работы является изучение возможностей функциональных языков по организации пользовательского интерфейса.

Основные задачи :

- Изучение (на примере muLISP'a) возможностей функциональных языков по созданию программ, управляемых событиями;
- Изучение функций работы с экраном дисплея и принципов организации видимых элементов программы как составляющих пользовательского интерфейса;
- Изучение принципов работы с потоками в программе на функциональном языке.

6.1. Краткие теоретические сведения.

6.1.1. Управление потоками.

В Common Лиспе, muLISPe ввод и вывод осуществляется независимо от конфигурации внешних устройств через потоки. Понятие потока в Лиспе сходно с аналогичным в C++ : потоки представляют собой логические каналы, из которых можно читать (поток ввода) или в которые можно писать (поток вывода) данные. Стандартным входным потоком (по умолчанию) считается поток из клавиатуры в программу, стандартным выходным потоком считается поток из программы на дисплей.

Для управления потоками в muLISPe имеются встроенные функции переключения потоков :

- Входного потока : (rds <устройство>);
- Выходного потока : (wrs <устройство>).

Устройством может быть клавиатура, дисплей, внешний файл, принтер. В случае использования внешнего файла нужно указать его имя (rds help.lsp) Переключение потоков на стандартные : (rds) - переключение входного потока на клавиатуру, (wrs) - переключение выходного потока на дисплей.

Для чтения данных из входного потока в Лиспе существует ряд функций, побочным эффектом которых является ввод s-выражений. Функции, применяемые для получения побочных эффектов, принято называть псевдофункциями. Основной функцией ввода является READ, которая читает s-выражение из входного потока и возвращает в качестве значения само это выражение. S-выражение может быть любым. Специализированные функции чтения (READ-CHAR, READ-LINE, READ-BYTE) позволяют читать из входного потока данные определенного типа. Если прочтенное значение необходимо сохранить, то вызов READ должен быть аргументом какой-нибудь формы, например, присваивания SETQ.

Функций типа EOF в Лиспе не существует. Для обозначения признака конца файла используется функция LISTEN. Вызов (listen) дает T, если во входном потоке есть хотя бы один символ.

Для идентификации входного и выходного потока в Лиспе существуют предикативные функции (inputfile имя_файла) и (outputfile имя_файла). Эти функции возвращают истинностное значение в том случае, когда имя входного/выходного потока совпадает с именем файла.

Последовательность действий по организации чтения из файла :

– Подготовить структуру для записи компонент файла;
– Переключить входной поток на внешний файл (rds file_name);
– Чтение из файла в цикле с признаком конца файла в качестве условия завершения ((not (listen)));
– Обратное переключение входного потока на устройство ввода по умолчанию (rds);
– Передать полученное в цикле значение s-выражения в программу.

В Лиспе имеется также ряд функций вывода информации в выходной поток.

Функция (print <s-выражение>) является псевдофункцией, значением которой является значение s-выражения, а побочным эффектом - вывод этого значения в выходной поток. Порядок выполнения :

1) Вычисляется s-выражение.
2) Полученное значение записывается в выходной поток.
3) Полученное значение возвращается в виде значения функции.
4) Осуществляется перевод курсора на новую строку.

Пример. (+ (print 2) 3) дает в качестве побочного эффекта вывод на экран числа 2, а в качестве значения - число 5.

Для последовательного вывода на одну строку более одного выражения следует использовать функции prin1 и princ. Действие аналогично print, отличие - в отсутствии перевода строки. Функция prin1 выводит в выходной поток значение с ограничивающими вертикальными скобками, princ - без. Рекомендуется prin1 использовать при записи в файл, а princ - для вывода на экран. Нежелательного явления печати дважды можно избежать при использовании функций print, prin1 и princ на самом верхнем уровне.

Помимо print, princ и prin1, для вывода в выходной поток данных определенного типа служат специализированные функции вывода write-line, write-string и write-byte. В отличие от print, princ и prin1, эти функции не вычисляют своего аргумента. Функцию write-byte с числом 26 в качестве аргумента рекомендуется использовать для обозначения конца файла.

Перевод строки можно осуществить специально предназначенной для этого функцией terpri, либо вызывая ее без аргументов (однократный

перевод), либо с некоторым натуральным числом в качестве аргумента (вставка нескольких пустых строк).

6.1.2. Работа с экраном дисплея.

Для работы с экраном дисплея в muLISP'е существует ряд встроенных функций, которые обеспечивают управление выводом на экран в текстовом режиме MS DOS. Рассмотрим важнейшие из них.

Функция (CLEAR-SCREEN) очищает текущее окно, перемещает курсор в верхний левый угол окна и возвращает T.

Функция (SET-CURSOR) возвращает список из двух целых чисел, задающих позицию курсора.

Функция (SET-CURSOR ROW COLUMN) перемещает курсор в строку с номером ROW и колонку с номером COLUMN. Если 0<=ROW<25 и 0<=COLUMN<80, то функция перемещает курсор в требуемую строку и колонку и выдает T. В противном случае возвращается NIL.

Функции (ROW) и (COLUMN) возвращают, соответственно, номер строки и номер колонки, в которых находится курсор относительно текущего окна.

Функция (INSERT-LINES N) в текущем окне вставляет N пустых строк, начиная со строки, помеченной курсором, и возвращает T, если N>=0. В противном случае возвращается NIL.

Функция (DELETE-LINES N) в текущем окне уничтожает N строк, начиная со строки, помеченной курсором, и возвращает T, если N>=0. В противном случае возвращается NIL.

Функция (MAKE-WINDOW ROW COLUMN ROWS COLUMNS) создает на экране прямоугольную область как текущее окно, перемещает курсор в левый верхний угол окна и возвращает T. Верхний левый угол окна определяется значениями аргументов ROW и COLUMN. Аргументы ROWS и COLUMNS представляют собой ширину и высоту окна. Аргумент ROW должен быть нулем или положительным целым числом меньшим, чем 25. Аргумент COLUMN должен быть нулем или положительным целым числом меньшим, чем 80. И ROW, и COLUMN по умолчанию принимаются за 0. Аргумент ROWS должен быть положительным целым числом меньшим или равным (25 - ROW). Значение ROWS по умолчанию рассматриваются как (25 - ROW). Аргумент COLUMNS должен быть положительным целым числом меньшим или равным (80 - COLUMN). Значение COLUMNS по умолчанию рассматриваются как (80 - COLUMN).

Функция (MAKE-WINDOW) возвращает список из четырех элементов: исходной строки, исходной колонки, количества строк и количества колонок текущего окна.

Функция (FOREGROUND-COLOR N) устанавливает цвет выводимых на экран символов (фон переднего плана) в зависимости от значения

числа N. Число N соответствует одной из 16 цветовых констант для режима работы со средним и высоким разрешением графического адаптера EGA [9] : 0 - черный, 1 - синий, 2 - зеленый и т.д. Если функция FOREGROUND-COLOR вызывается без аргументов, то они возвращают соответственно код текущего цвета текста.

Аналогичным образом работает функция (BACKGROUND-COLOR N), которая устанавливает цвет бордюра (фон заднего плана).

Функция (DISPLAY-PAGE N) устанавливает в качестве текущей страницу видеопамяти [9] с номером N и возвращает в качестве результата номер предыдущей видеостраницы. При этом необходимо, чтобы muLISP работал на компьютере IBM PC с графическим дисплеем, а N<=0<8. В противном случае функция DISPLAY-PAGE возвращает номер текущей страницы дисплея. Это дает возможность быстро чередовать содержимое нескольких экранов дисплея, заполненных текстом.

6.1.3. Некоторые дополнительные функции muLISP'a.

Функция (ASCII Character) для символа Character возвращает его ASCII-код. Функция ASCII является встроенной функцией muLISP’a.

Результат вызова функции (ASCII Code) есть символ, соответствующий заданному значению Code кода ASCII.

С применением функции ASCII в библиотеке COMMON.LSP описаны функции CHAR-CODE и CODE-CHAR.

Функция (CHAR-CODE Character) для символа Character возвращает его ASCII-код.

Функция (CODE-CHAR Code) возвращает символ, соответствующий заданному значению Code кода ASCII.

Данные функции могут быть, в частности, полезны при написании оболочек для задания рамки окна, заполнения экрана заданным символом для создания специфического фона (пример – заполнение панели экрана полутеневым шаблоном) и т.п.

6.2. Задание на лабораторную работу.

Задание 1.

Написать программу, создающую два окна : окно меню и рабочее окно. Окно меню может быть расположено горизонтально или вертикально, сверху или, соответственно, справа экрана. Размеры экранов выбрать самостоятельно. Меню должно обеспечивать вызовы всех функций, реализованных в лабораторных работах 2 и 3. Ввод исходных данных и вывод результата осуществлять в рабочем окне.

Задание 2.

Вывести на экран прямоугольник, содержащий 6 на 6 квадратов различного цвета. Обеспечить перемещение цветов по квадратам, для нечетных вариантов по горизонтали, четных - по вертикали. Обеспечить движение в обе стороны.

Задание 3.

Вывести на экран прямоугольник, содержащий 6 на 6 квадратов различного цвета. Обеспечить циклическое перемещение цветов по спирали (рис. 9). Из центра должен обеспечиваться переход в начальную позицию.

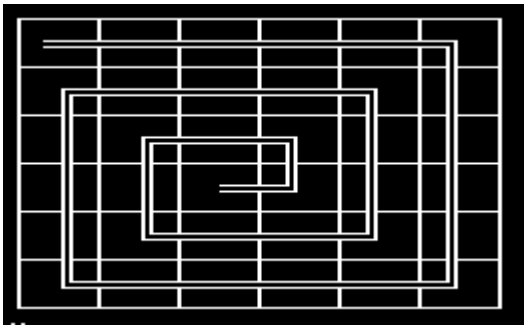

Рис.9 Перемещение по спирали

6.3. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :
- формулировку цели и задач;
- описание процесса разработки программ;
- выводы по проделанной реализации.

ЛАБОРАТОРНАЯ РАБОТА № 7

ОРГАНИЗАЦИЯ ДИНАМИЧЕСКИХ БАЗ ДАННЫХ СРЕДСТВАМИ Microsoft muLISP.

Цель работы.

Целью лабораторной работы является изучение возможностей использования свойств символов и ассоциативных списков для организации динамических баз данных.

7.1. Краткие теоретические сведения.
7.1.1. Символы и их свойства.

Определение. Под символом в Лиспе понимается имя, состоящее из букв, цифр, специальных знаков и обозначающее какой-либо предмет, объект, вещь, действие из реального мира.

В Лиспе символы могут обозначать любые лисповские объекты, включая функции.

Символ является структурным объектом, состоящим из четырех компонент, соответствующих имени, значению, а также связанных с символом определению функции и списку свойств.

Для чтения значений различных системных свойств символа в библиотеке COMMON.LSP существуют специальные функции, приведенные в таблице 7.1.

Таблица 7.1. Функции чтения системных свойств символа.

Характеристика.	Форма представления.	Функция чтения.
Имя	Набор литер	(SYMBOLP *имя*)
Значение	Произвольный лисповский объект	(SYMBOL-VALUE *имя*)
Определение функции	Лямбда-выражение	(SYMBOL-FUNCTION *имя*)
Список свойств	Список точечных пар	(SYMBOL-PLIST *имя*)

В Лиспе с символом можно связать именованные свойства. Свойства символа записываются в хранимый вместе с символом список свойств :

(<имя символа> (<имя свойства 1> <значение свойства 1>) . . .
(<имя свойства N> <значение свойства N>))

Для работы со списками свойств в Лиспе имеются три встроенные функции :

1) Включение свойства в список свойств.
(put <символ> <свойство> <значение свойства>)
2) Просмотр значения заданного свойства.
(get <символ> <свойство>)
3) Удаление заданного свойства из списка свойств.
(remprop <символ> <свойство>)

Задача 1.

Дан символ с некоторым именем name. Требуется сформировать список свойств.

Решение.

Данная задача решается в три этапа. Сначала необходимо задать (ввести) количество свойств, затем ввести список названий свойств и только затем - значения свойств.

```
; Формирование списка свойств символа
; Головная функция формирования списка свойств
(defun f1 (name)
        (print (pack* "Введите количество свойств символа " name))
        (f3 name (f2 name (read))))
; Функция формирования списка названий свойств
(defun f2 (name num)
        ((zerop num) nil)
        (print (pack* "Введите название " num "-го
                        свойства символа " name " : "))
        (cons (read)(f2 name (- num 1))))
; Ввод значений свойств
(defun f3 (symb_name prop_name_list)
        ((null prop_name_list) nil)
        (print (pack* "Введите значение свойства "
                      (car prop_name_list)
                      " символа " symb_name))
        (put symb_name (car prop_name_list)(read))
        (f3 symb_name (cdr prop_name_list)))
```

Здесь в качестве вспомогательной используется функция конкатенации : (pack* строка1 : строкаN).

Задача 2.

Дан список символов lst и некоторое свойство prop. Требуется : удалить свойство prop у всех тех символов списка lst, у которых это свойство имеется.

Решение. Для решения данной задачи воспользуемся функциями GET и REMPROP.

```
; Удаление заданного свойства у символов списка
; Вариант 1
(defun f4 (lst prop)
        ((null lst) nil)
        (remprop (car lst) prop)
        (f4 (cdr lst) prop))
; Вариант 2 - неправильный
(defun f5 (lst prop)
        ((null lst) nil)
        ((get (car lst) prop)(remprop (car lst) prop))
        (f5 (cdr lst) prop))
```

```
; Подготовка тестовых данных
(put one 'Num 'Nechet)
(put two 'Num 'Chet)
(put A 'Sym 'A-lat)
(put B 'Sym 'B-lat)
(put three 'Num 'Nechet)
(put four 'Num 'Chet)
(put C 'Sym 'C-lat)
(put D 'Sym 'D-lat)
```

Результатом вызова (f4 '(one A two B) 'Num) будет удаление свойства Num у символов one и two. Но вызов (f5 '(three C four D) 'Num) не приведет к удалению свойства Num у символа four, поскольку в случае обнаружения искомого свойства у одного из символов списка рекурсивного вызова функции f5 уже не произойдет.

Теперь рассмотрим еще один вариант решения задачи – с использованием локального определения LET.

```
; Удаление заданного свойства - вариант с использованием
; локального определения LET
(defun f6 (lst prop)
 (let
   ((obj_list lst)
    (property prop))
  ((null obj_list) nil)
  (remprop (car obj_list) property)
  (f6 (cdr obj_list) property)
 )
)
```

Тестовые данные – аналогично предыдущему примеру.

Задача 3.

Дан список символов lst и некоторое свойство prop. Требуется : заменить текущее значение свойства prop заданным значением val у всех тех символов списка lst, у которых это свойство имеется.

Решение.

```
; Изменение значения заданного свойства
(defun chngprop (lst prop val)
    ((null lst) T)
    ((and  (get (car lst) prop)
           (chngprop (cdr lst) prop val))
     (put (car lst) prop val))
    (chngprop (cdr lst) prop val))

; Тестовый набор данных
```

```
(put one 'What_is_it Number)
(put two 'What_is_it Number)
(put A 'What_is_it Symbolic_value)
(put B 'What_is_it Symbolic_value)
(setq input_list '(one A two B))
```

Вызов (chngprop input_list 'What_is_it 'Atom) приводит к тому, что у всех элементов списка input_list, для которых определено свойство What_is_it, в качестве нового значения этого свойства будет задано Atom.

Задача 4.

Дан список символов lst и некоторое значение old_val свойства prop. Требуется : заменить значение old_val свойства prop новым значением new_val у всех тех символов списка lst, у которых это свойство имеется.

Решение.

```
; Замена заданного значения заданного свойства новым
(defun chngprop2 (lst prop old_val new_val)
    ((null lst) T)
    ((and  (equal (get (car lst) prop) old_val)
           (chngprop2 (cdr lst) prop old_val new_val))
     (put (car lst) prop new_val))
    (chngprop2 (cdr lst) prop old_val new_val))

; Тестовый набор данных
(put one 'What_is_it Number)
(put two 'What_is_it Number)
(put A 'What_is_it Symbolic_value)
(put B 'What_is_it Symbolic_value)
(setq input_list '(one A two B))
```

Вызов (chngprop2 input_list 'What_is_it 'Symbolic_value 'Symbol) приводит к тому, что у всех элементов списка input_list, значением свойства What_is_it которых является Symbolic_value, в качестве нового значения этого свойства будет задано Symbol.

Показанные действия со свойствами символов используется при создании динамических Баз Данных (БД) и текстовых редакторов.

Задача 5.

Требуется обеспечить выполнение функций редактора для разных режимов работы при нажатии определенных клавиш. С нажатием одной и той же клавиши в разных режимах могут быть связаны разные функции. Отдельные клавиши задействуются только в определенных режимах : F3 -

загрузка текста из файла (при активизации главного меню), ↑, ↓, ← и → - перемещение курсора, F2 - сохранение текста в файле (в режиме правки текста).

Решение.

Свяжем с символом, соответствующим ASCII-коду каждой из задействованных клавиш, список свойств. Каждому из режимов, где клавиша будет задействована, поставим в соответствие название свойства, а имени вызываемой по нажатию клавиши функции - значение свойства. Для вызова самой функции воспользуемся функционалом MAPC.

Пример. Для клавиш Enter и Backspace в режиме редактирования будем иметь :

```
(MAPC '(LAMBDA (PAIR)
                    (PUT (ASCII (CAR PAIR)) 'EDITOR (CADR PAIR)))
      '((13 EDITOR-ENTER)
        (8  EDITOR-BACKSPACE)))
```

Задача 6.

Требуется реализовать БД машинного словаря основ [7] для задачи морфологического анализа текстов русского языка.

Теоретические сведения.

В основу построения описанных в [7] алгоритмов морфологического анализа и синтеза слов положено разбиение всех слов на морфологические классы.

Определение. Морфологический класс определяет характер изменения буквенного состава форм слов.

Изменение форм слов может быть связано с изменением буквенного состава либо основы слова, либо его окончания.

В силу вышесказанного морфологические классы слов принято подразделять на :

1) Основоизменительные классы, характеризующие систему изменения основ;
2) Флективные классы слов.

Флективные классы определены для изменяемых слов на основе анализа их синтаксической функции и систем падежных, личных и родовых окончаний. Флективный класс характеризуется либо системой признаков, либо словом-представителем.

Решение.

Согласно приведенному в [7] описанию, для каждой основы в словаре приводится порядковый номер (десятиричное число), буквенный код основы, номер флективного класса и номер основоизменительного класса. Рассмотрим принципиальную возможность построения подобного словаря

с применением имеющихся в muLisp е средств работы с файлами на внешних носителях и списками свойств. Каждой представленной в словаре основе мы поставим в соответствие символьный объект с именем, соответствующем буквенному коду основы по словарю (azot, balk, ball, bank1, bank2). Порядковый номер основы по словарю, номера флективных и основоизменительных классов будем рассматривать как свойства соответствующего символа, описываемые списком свойств. Для создания динамической БД в памяти таблицу основ мы представим как объект, имеющий в качестве свойств данные конкретных основ (рис. 10).

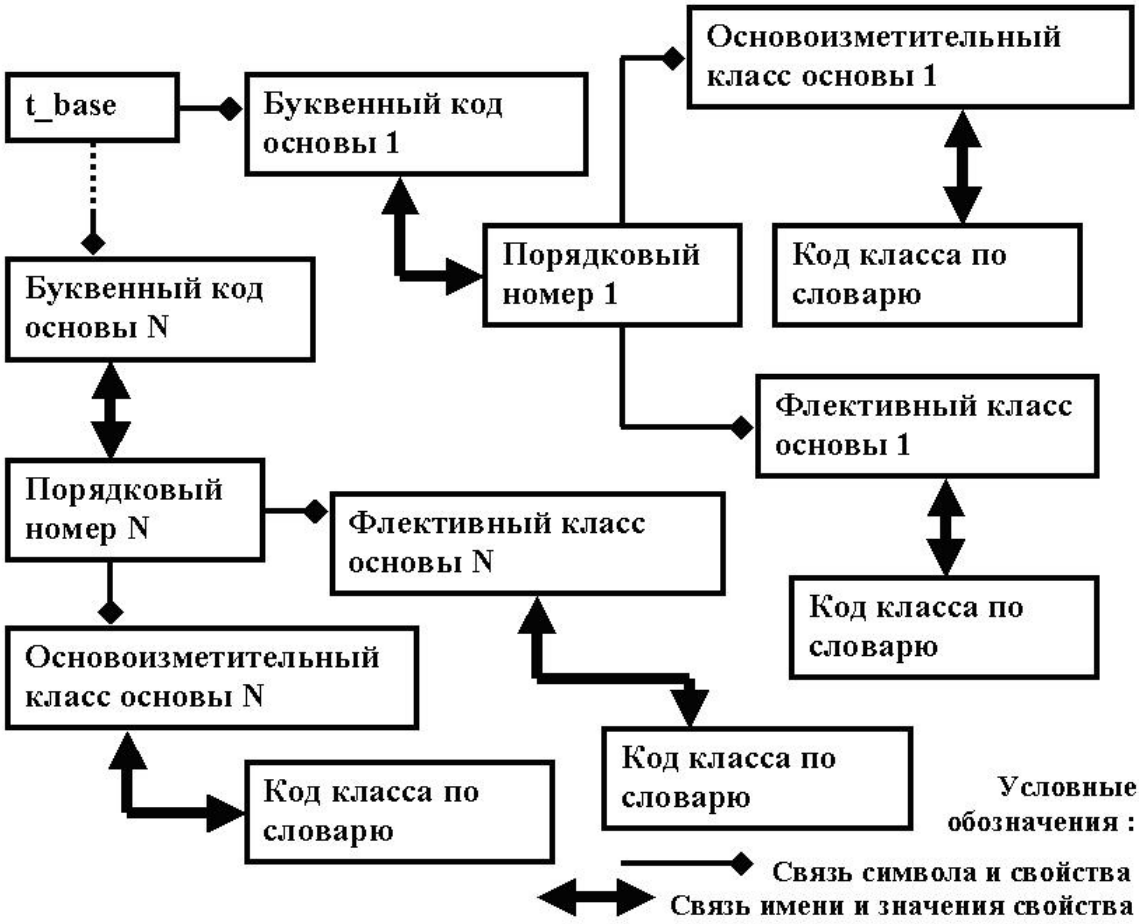

Рис.10 Представление словаря основ с использованием списков свойств

Этап 1 : считывание информации из внешних файлов.

В соответствие с особенностями ввода/вывода в Лиспе для облегчения последующего назначения свойств символам при формировании БД в памяти ЭВМ информацию словаря основ можно хранить в отдельных файлах : буквенных кодов основ (basesymb.txt), порядковых номеров основ (basenumb.txt), кодов основоизменительных классов (bchangcl.txt), кодов флективных классов (flect_cl.txt). Для записи компонент файлов мы созда-

ем соответствующие списки. Рассмотрим пример для файла буквенных кодов основ.

```
(defun load_bases_symb_codes (file_name)
   (setq bases_symb_codes_list nil)
   (rds file_name)
   (loop
      ((not (listen)))
      (setq bases_symb_codes_list (cons (read-line) bases_symb_codes_list)))
   (rds) bases_symb_codes_list)
```

Вызов (load_bases_symb_codes basesymb.txt) формирует в памяти список буквенных кодов основ.

Аналогичным образом организуется работа функций :

- load_bases_numbers – для формирования списка порядковых номеров основ;
- load_basechange_classes – для формирования списка кодов основоизменительных классов (в тестовых примерах мы используем двухразрядные восьмеричные коды, с которыми работают представленные в [7] алгоритмы);
- load_flect_classes – для формирования списка кодов флективных классов (в тестовых примерах мы используем трехразрядные восьмеричные коды).

<u>Этап 2 : построение списков свойств.</u>

В соответствии с выбранной структурой для представления информации словаря основ описание каждого из свойств с множественным значением будет представляться списком :

(<имя свойства> (<значение 1> : <значение N>)).

Так, для основоизменительных классов мы будем иметь следующее описание :

(basechange_class (<основоизменительный класс основы 1>. . .
<основоизменительный класс основы N>)),

для флективных классов :

(flect_class (<флективный класс основы 1 > . . .
<флективный класс основы N>)).

На основе считанных из файлов списков кодов основоизменительных (basechange_class_list) и флективных классов (flect_class_list) формируем списочное описание свойств с множественным значением :

```
(list (list 'basechange_class basechange_class_list)
   (list 'flect_class flect_class_list))
```

Опишем функцию формирования БД с составной структурой свойств описываемых объектов.

```
(defun db_complex_props_make (db_name props vals vals_of_props_list)
; db_name - имя объекта
; props - список названий ключевых свойств объекта
; vals - список значений ключевых свойств объекта
; vals_of_props_list - списочное описание неключевых свойств объекта
;                    (<атрибут> <список значений атрибута>)
    ((null vals_of_props_list)(db_make db_name props vals))
    ((db_make db_name props vals)
        (complex_props_make vals vals_of_props_list))
)

; Функция формирования БД в оперативной памяти
(defun db_make (db_name props vals)
; Случай различной длины списков имен и значений свойств
    ((or
        (and (null props)(not (null vals)))
        (and (not (null props))(null vals))
    ) nil)
; Условие окончания рекурсии
    ((and (null (cdr props))(null (cdr vals)))
        (put db_name (car props)(car vals)))
; Задание значения очередного свойства
    ((put db_name (car props)(car vals))
        (db_make db_name (cdr props)(cdr vals)))
)

; Формирование структуры свойств.
(defun complex_props_make (vals vals_of_props_list)
     ((and (null (cdr vals))
            (not (null vals_of_props_list)))
      (complex_prop_make (car vals) vals_of_props_list))
     ((complex_prop_make (car vals) vals_of_props_list)
     (complex_props_make (cdr vals)(vals_of_props_make vals_of_props_list))
))
```

Функция complex_props_make с помощью вспомогательной функции complex_prop_make для каждого символа из списка vals ставит в соответствие свойства и их значения из списка vals_of_props_list. Каждый элемент списка vals_of_props_list соответствует описанию списочному описанию свойства с множественным значением (рис.10). В содержательной интерпретации список vals содержит описание значений ключевых свойств объ-

ектов (в нашем случае - это порядковые номера основ по словарю). Вспомогательная функция vals_of_props_make для каждого очередного объекта с уже сформированным списком свойств удаляет описания значений его свойств из списка vals_of_props_list.

```
; Формирование элементов списка свойств отдельного объекта.
(defun complex_prop_make (val vals_of_props_list)
    ((null (cdr vals_of_props_list))
        (put val (caar vals_of_props_list)(caadar vals_of_props_list))
    )
    ((put val (caar vals_of_props_list)(caadar vals_of_props_list))
        (complex_prop_make val (cdr vals_of_props_list))
    )
)

; Удаление описания свойств очередного объекта из списочного
; описания неключевых свойств
(defun vals_of_props_make (vals_of_props_list)
    ((null vals_of_props_list) nil)
    (cons (list (caar vals_of_props_list)(cdadar vals_of_props_list))
            (vals_of_props_make (cdr vals_of_props_list))
    )
)

; Головная функция программы
(defun test5 (db_name obj1 obj2)
    (db_complex_props_make db_name
                           bases_symb_codes_list
                           bases_numbers_list
                           (list
                                (list obj1 basechange_class_list)
                                (list obj2 flect_class_list)
                           )
    )
)
```

После загрузки программы и последовательного вызова функций считывания информации из внешних файлов :

```
(load_bases_symb_codes basesymb.txt)
(load_bases_numbers basenumb.txt)
(load_basechange_classes bchangcl.txt)
(load_flect_classes flect_cl.txt)
```

с помощью вызова функции test5 :

(test5 't_base 'basechange_class 'flect_class)

строится совокупность списков свойств в соответствии с приведенной на рис. 10 схемой.

Для просмотра свойств элементов созданной структуры достаточно вызвать функцию get из командной строки интерпретатора.

Пример : вызов (get t_base "balk") возвращает в качестве значения номер основы по словарю основ, то есть "002"; (get "002" basechange_class) для основы с заданным номером выдает код основоизменительного класса, то есть "11"; (get "002" flect_class) для основы с заданным номером выдает трехразрядный восьмеричный код флективного класса, то есть "060".

Применительно к табличной модели данных [12] получаем, что имя символа соответствует (рис. 11) либо названию таблицы (имени отношения), либо заглавию табличной строки, названия свойств – заглавиям столбцов, значения свойств – содержимому ячеек.

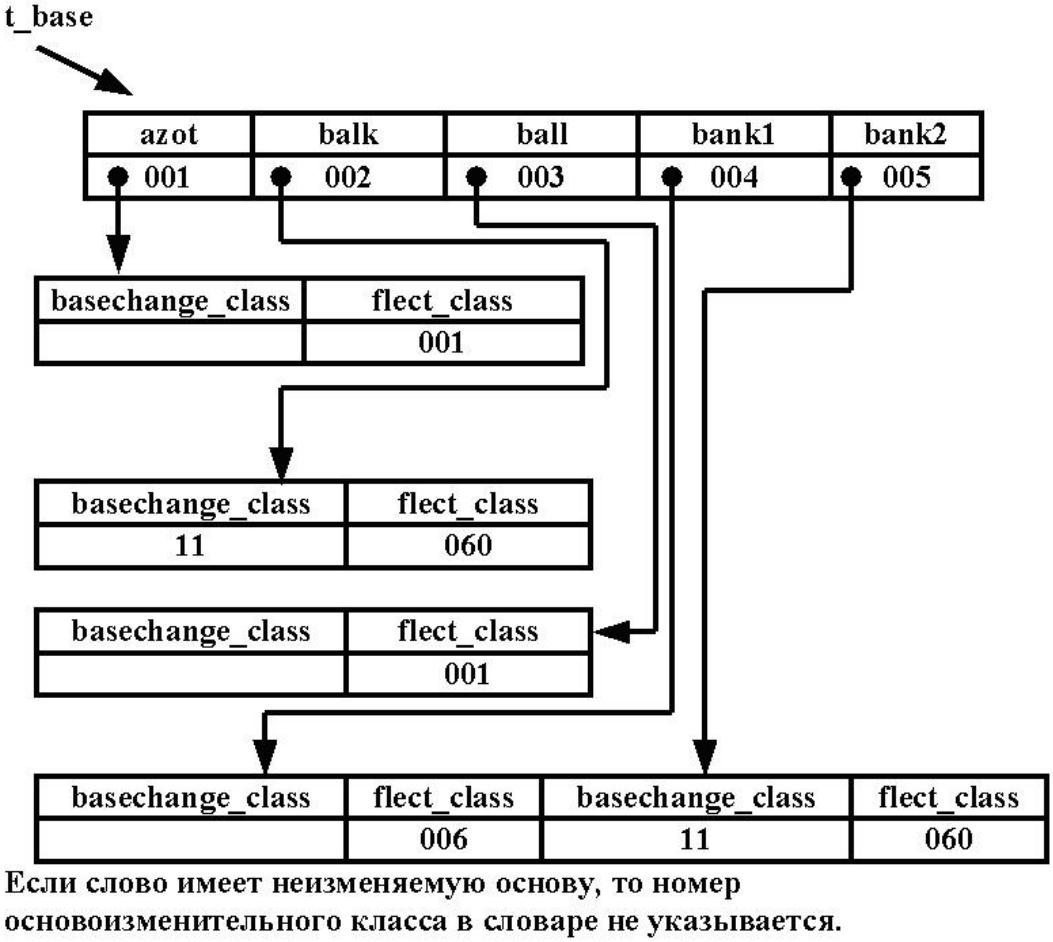

Рис.11 Сформированная БД для тестового примера

7.1.2. Ассоциативные списки.

7.1.2.1. Создание ассоциативного списка.

Определение. Ассоциативный список или просто а-список (a-list) есть основанная на списках и точечных парах структура данных, описывающая связи наборов данных obji и ключевых полей keyi, для работы с которой существуют готовые функции.

Ассоциативные списки могут быть следующих двух видов :

((key1.obj1)(key2.obj2) : (keyN.objN))

((key1 obj1)(key2 obj2) : (keyN objN))

Первый вид ассоциативных списков имеет место в тех случаях, когда связанные с ключами объекты являются атомарными.

Ассоциативный список формируется с помощью встроенной функции PAIRLIS из списка ключей keys и списка objects соответствующих им объектов. Формат вызова :

(pairlis keys objects a_list)

Третий аргумент функции pairlis есть формируемый а-список, в начало которого добавляются новые пары "ключ-объект". При вызове в качестве значения a_list либо задается nil, либо предполагается, что a_list был сформирован ранее.

В качестве примера опишем построение в оперативной памяти машинного словаря основ, рассмотренного нами в предыдущем разделе.

```
; Построение в оперативной памяти машинного словаря основ слов
; с помощью ассоциативного списка.
(setq t_base nil)
(pairlis '(base_number symb_code basechange_class
           flect_class)
          '(("001" "002" "003" "004" "005")
           ("azot" "balk" "ball" "bank1" "bank2")
           ("" "11" "" "" "11")
           ("001" "060" "001" "006" "060"))
        t_base)
```

Как результат вызова функции pairlis значением t_base становится список :

```
((base_number "001" "002" "003" "004" "005")
 (symb_code "azot" "balk" "ball" "bank1" "bank2")
 (basechange_class "" "11" "" "" "11")
 (flect_class "001" "060" "001" "006" "060"))
```

7.1.2.2. Поиск элементов в ассоциативном списке.

Ассоциативный список можно рассматривать как отображение множества ключей на множество соответствующих им объектов. Конкретные данные можно получить по значению ключа с помощью функции :

(assoc key a_list)

В качестве значения assoc возвращает пару "ключ-объект".

Пример (для машинного словаря основ) :

```
(setq t_base nil)
(setq t_base
      (pairlis '(base_number symb_code basechange_class flect_class)
               '(("001" "002" "003" "004" "005")
                 ("azot" "balk" "ball" "bank1" "bank2")
                 ("" "11" "" "" "11")
                 ("001" "060" "001" "006" "060")) t_base))
```

Вызов (assoc 'base_number t_base) возвращает в качестве результата пару, соответствующую значению ключа base_number, то есть список : (base_number "001" "002" "003" "004" "005").

В muLISP’e, Common Lisp’e имеется функция RASSOC (обратная ASSOC), которая находит ключ по заданному объекту :

(rassoc obj a_list).

В качестве значения rassoc возвращает пару "ключ-объект", но только в том случае, если a_list есть список точечных пар.

Пример :

```
(setq t_base1 nil)
(setq t_base1 (pairlis '(base_number symb_code basechange_class flect_class)
                       '("001" "azot" "" "001") t_base1))
; Формирование t_base описано в предыдущем примере
```

Вызов :

(rassoc '("azot" "balk" "ball" "bank1" "bank2") t_base)

возвращает в качестве результата nil, а вызов :

(rassoc '"azot" t_base1)

возвращает в качестве результата точечную пару :

(symb_code . "azot").

7.1.2.3. Добавление элементов в ассоциативный список.

Ассоциативный список можно обновлять и использовать в режиме стека. Новые пары "ключ-объект" добавляются к нему только в начало списка, хотя в списке уже могут быть данные с тем же ключом. Добавление осуществляется функцией ACONS :

(acons key obj a_list)?(cons (cons key obj) a_list)

Примеры.

```
; Построим в оперативной памяти два фрагмента машинного словаря основ
(setq t_base2 nil)
(setq t_base2 (pairlis '(base_number symb_code basechange_class)
                       '(("001" "002" "003" "004" "005")
                         ("azot" "balk" "ball" "bank1" "bank2")
                         ("" "11" "" "" "11")) t_base2))
(setq t_base3 nil)
(setq t_base3 (pairlis '(base_number symb_code basechange_class)
                        '("001" "azot" "") t_base3))
```

Результатом вызова :

```
(setq t_base2 (acons flect_class '("001" "060" "001" "006" "060") t_base2))
```

будет добавление пары (flect_class "001" "060" "001" "006" "060") в начало списка t_base2.

Результатом вызова :

```
(setq t_base3 (acons flect_class '"001" t_base3))
```

будет добавление пары (flect_class . "001") в начало списка t_base3.

Поскольку ASSOC просматривает список слева направо и доходит лишь до первой пары с искомым ключом, то более старые пары остаются вне рассмотрения. Использование ассоциативных списков подобным образом может, в частности, решить проблему поддержки меняющихся связей переменных и контекста вычисления. С такой целью ассоциативные списки используются при программировании самого интерпретатора Лиспа. Преимущества : простота изменения связей и возможность возврата к значениям старых связей. Недостаток : поиск данных замедляется пропорционально длине списка.

7.1.2.4. Модификация ассоциативных списков.

Ассоциативный список можно изменить путем :

- Физического изменения данных, связанных с ключом;
- Физического удаления данных, связанных с ключом;
- Изменения ключа.

Во всех трех случаях изменение ассоциативных списков происходит с применением структуроразрушающих функций. Описанные далее функ-

ции модификации ассоциативных списков в результате своего выполнения теряют старые значения ассоциативных списков.

7.1.2.4.1. Изменение данных, связанных с ключом.

```
; Изменение данных, связанных с ключом
(defun putassoc (key obj a_list)
    ((null a_list) nil)
    ((equal (caar a_list) key)(rplacd (car a_list) obj))
    ((null (cdr a_list))(rplacd a_list (list (cons key obj))))
    (putassoc key obj (cdr a_list)))
```

Пример.

Предположим, что при формировании машинного словаря основ мы задали неверное значение порядкового номера основы слова :

```
(setq t_base4 nil)
(setq t_base4 (pairlis '(base_number symb_code basechange_class)
                        '("002" "azot" "") t_base4))
```

Значением t_base4 изначально будет :

((base_number . “002”)(symb_code . “azot”)(basechange_class .””))

В результате вызова (putassoc ' base_number “001” t_base4) значение t_base4 изменится на :

((base_number . “001”)(symb_code . “azot”)(basechange_class .””)).

7.1.2.4.2. Изменение ключа.

```
(defun keyassoc (old_key new_key a_list)
    ((null a_list) nil)
    ((equal (caar a_list) old_key)(rplaca (car a_list) new_key))
    (keyassoc old_key new_key (cdr a_list)))
```

Пример.

Предположим, что при формировании машинного словаря основ мы задали неверное название одного из ключей :

```
(setq t_base5 nil)
(setq t_base5 (pairlis '(number symb_code basechange_class)
                        '("001" "azot" "") t_base5))
```

Значением t_base5 изначально будет :

((number . “001”)(symb_code . “azot”)(basechange_class .””))

В результате вызова (keyassoc 'number 'base_number t_base5) значение t_base5 изменится на :

((base_number . “001”)(symb_code . “azot”)(basechange_class .””)).

7.1.2.4.3. Изменение ключа.

```
; Удаление данных, связанных с ключом
(defun remassoc (key a_list)
    ((null (cdr a_list)) nil)
    ((and  (equal (caar a_list) key)
            (rplaca a_list (cadr a_list)))
      (rplacd a_list (cddr a_list)))
    (remassoc key (cdr a_list)))
```

Пример.

Удалим из машинного словаря основ информацию о флективном классе слова.

```
; Построим в оперативной памяти фрагмент машинного словаря основ
(setq t_base6 nil)
(setq t_base6 (pairlis '(base_number symb_code
                          basechange_class flect_class)
                        '("001" "azot" "" "001") t_base6))
```

Значением t_base6 изначально будет :

((base_number . “001”)(symb_code . “azot”)
(basechange_class .””)(flect_class . “001”))

В результате вызова (remassoc ‘flect_class t_base6) значение t_base6 изменится на :

((base_number . “001”)(symb_code . “azot”)(basechange_class .””)).

7.2. Задание на лабораторную работу.

Написать программу, обеспечивающую создание на диске базы данных и работу с ней. В функции программы должно входить :

- создание базы данных;
- добавление информации в базу данных;
- модификацию (редактирование) информации;
- запись базы данных на диск;
- загрузку базы данных в оперативную память;
- просмотр информации;
- удаление информации из базы данных;
- поиск информации в базе данных;
- сортировка информации.

Программа должна предоставлять пользователю дружественный интерфейс. Вызов функций программы должен осуществляться из меню. Ввод исходных данных и вывод результата осуществлять в рабочем окне.

За основу рекомендуется взять пользовательский интерфейс из лабораторной работы №6. Вариант задания указан в Таблице 7.2.

Таблица 7.2. Варианты заданий.

Вариант.	Предметная область.	Рекомендуемая литература.
1.	Системы общения на естественном языке с базами данных.	[10], кн. 1, с. 42-51, 65-94 [10], кн. 3, с. 213-235
2.	Интеллектуальные вопросно-ответные системы.	[10], кн. 1, с. 32-42
3.	Диалоговые системы решения задач.	[10], кн. 1, с. 51-59
4.	Системы обработки связных текстов.	[10], кн. 1, с. 59-64 [10], кн. 2, с. 115-126
5.	Системы речевого общения.	[10], кн. 1, с. 95-139
6.	Системы машинного перевода.	[10], кн. 1, с. 201-261
7.	Специализированные процессоры для интеллектуальных систем.	[10], кн. 3, с. 213-293
8.	Специализированные процессоры для языков высокого уровня.	[10], кн. 3, с. 293-328
9.	Инструментальные средства для разработки интеллектуальных систем.	[10], кн. 3, с. 109-168
10.	Компьютерные словари русского языка.	[11]
11.	Обучающиеся системы.	[10], кн. 2, с. 206-231

7.3. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :

- Формулировку цели и постановку задач проводимых исследований;
- Анализ задания, выбор метода решения с обоснованием, описание процесса разработки программ, полученные результаты и их анализ (обоснование);
- Текст программы с комментариями и обоснованиями;
- Выводы по проведенным машинным экспериментам.

ЛАБОРАТОРНАЯ РАБОТА № 8

РАЗРАБОТКА ИНТЕРФЕЙСОВ НА ЕСТЕСТВЕННОМ ЯЗЫКЕ. СИНТАКСИЧЕСКИЙ АНАЛИЗ ФРАЗ РУССКОГО ЯЗЫКА С ПРИМЕНЕНИЕМ ФОРМАЛЬНЫХ ГРАММАТИК.

Цель работы.

Целью лабораторной работы является изучение возможностей функциональных языков по реализации методов структурного распознавания образов (на примере формальных грамматик).

Основные задачи :

- Изучение методов приближенного представления фраз естественного языка формальными грамматиками и языками;
- Изучение методов реализации грамматических правил средствами функционального языка.

8.1. Краткие теоретические сведения.

8.1.1. Некоторые сведения из теории формальных языков и грамматик.

8.1.1.1. Основные определения.

Определение 1. Под алфавитом понимается множество допустимых символов $X = \{x_1, x_2, \ldots, x_n\}$, между которыми существует отношение порядка.

Определение 2. Под языком *L* над алфавитом *X* понимается множество *L(X)* допустимых цепочек, составленных из символов алфавита *X*.

Определение 3. Формальной грамматикой *G* над языком *L* называется конечный набор правил, по которым из элементов алфавита *X* строятся цепочки (выводы). Говорят также, что язык *L* порождается грамматикой *G* : *L(G)*, если любая цепочка $\alpha \in L$ может быть построена путем применения правил из *G* и никакая цепочка $\beta \notin L$ не может быть построена применением правил из *G*.

8.1.1.2. Требования к формальной грамматике.

1) Должно быть задано множество V_N нетерминальных символов. Под ***нетерминальными*** символами (нетерминалами) понимаются некото-

рые обозначения (абстракции), которые играют вспомогательную роль и служат посредниками при построении правил. Примерами таких символов могут послужить <Предложение>, <Подлежащее>, <Сказуемое>.

2) Должно быть определено множество V_T терминальных символов. Под терминальными символами (терминалами) понимаются зарезервированные слова, которые включаются в словарь,. Примеры : IF, ELSE, FOR, REPEAT, UNTIL (для языка Паскаль), ‘шел’, ‘дождь’, ‘мелкий’ (для подмножества Естественного Языка).
3) Должно быть определено множество правил вывода или продукций.
4) Для порождающей грамматики должен быть задан особый нетерминал – начальный символ, из которого начинается любой вывод. Примерами таких нетерминалов могут послужить : <программа>, <Предложение>.

8.1.1.3. Контекстно-свободная грамматика.

Определение 4. ***Контекстно-Свободной*** (КС-грамматикой) называется формальная грамматика с заданными V_N и V_T : $V_N \cap V_T = \varnothing$, множество продукций которой характеризуется тем, что в левой части любого правила содержится единственный символ-нетерминал : $\langle A \rangle \to \beta$, где $\beta \in (V_T \cup V_N)^*$, $(V_T \cup V_N)^* = \Lambda \cup (V_T \cup V_N)^+$ - множество слов произвольной длины, Λ - пустое слово.

Пример 1.

<Предложение> $\to$ <Сказуемое> <Подлежащее>.
<Сказуемое> $\to$ <Глагол>
<Глагол> $\to$ ‘шел’
<Глагол> $\to$ ‘шли’
<Глагол> $\to$ ‘дует’
<Глагол> $\to$ ‘прошла’
<Подлежащее> $\to$ <Существительное>
<Существительное> $\to$ ‘дождь’
<Существительное> $\to$ ‘дожди’
<Существительное> $\to$ ‘ветер’
<Существительное> $\to$ ‘гроза’

Правила грамматики раскрываются последовательно до тех пор, пока результат не будет соответствовать обрабатываемому предложению, либо не будет выполнено условие завершения (отсутствие новых продукций).

Недостаток : отсутствие согласования характеристик символов-терминалов.

8.1.1.4. Грамматика определенных дизъюнктов.

Грамматика Определенных Дизъюнктов (ГОД) отличается от КС-грамматики тем, что для терминалов вводятся характеристики (атрибуты), а нетерминалы могут иметь переменные для согласования характеристик терминальных символов. В частности, для подмножества русского языка в качестве таких характеристик можно ввести род, число и время у глагола, род и число – у существительного. Количество атрибутов у нетерминала не ограничивается.

Пример 2. Грамматика из Примера 1, преобразованная в грамматику определенных дизъюнктов в целях согласования рода и числа.

<Предложение> → <Сказуемое> <Подлежащее>.
<Сказуемое> → <Глагол (Род Число Время)>
<Глагол (муж ед прош)> → ‘шел’
<Глагол> → ‘шли’
<Глагол (муж ед наст)> → ‘дует’
<Глагол (жен ед наст)> → ‘дует’
<Глагол (жен ед прош)> → ‘прошла’
<Подлежащее> → <Существительное (Род Число)>
<Существительное (муж ед)> → ‘дождь’
<Существительное> → ‘дожди’
<Существительное (муж ед)> → ‘ветер’
<Существительное (жен ед)> → ‘гроза’

В данном случае род и число учитывается при помощи переменных, указанных в правилах вывода. При этом одноименные переменные должны обозначать одинаковый род/число.

8.1.2. Строки в Лиспе.

Строки относятся к простым типам данных. В Лиспе используется рассмотрение строк как одномерных массивов или векторов, элементами которых являются знаки. Многие более общие функции Лиспа, определенные для массивов и последовательностей (чтение, сравнение элементов), наследуются строками.

Кроме этого, в Лиспе определен и ряд специальных функций для работы с этим типом данных :

– Функции сравнения строк;
– Функции преобразования атомов и списков в строки и наоборот.

8.1.2.1. Функции Лиспа по работе со строками.

Функция (findstring <строка 1> <строка 2> N) показывает, начиная с какой позиции в строке 2 содержится строка 1 в качестве подстроки. Причем отсчет ведется с 0. Если N - нуль или положительное целое число, то поиск начинается с N-го символа строки 2.

Пример.

Результатом вызова (findstring 'tri 'string) будет 1, в то время как результатом вызова (findstring 'tri 'string_trip 4) будет 7.

Функция pack преобразует список символов в строку. Пример : (pack '(s t r I n g m e t t e r)) выдает в качестве результата строку STRINGMETTER.

Функция print-length выдает в качестве результата длину строки до первого пробела. Пример : (print-length 'string metter) возвращает в качестве результата 6.

Функция (string-left-trim <строка 1> <строка 2>) в случае, когда строка 2 начинается строкой 1, возвращает в качестве результата подстроку строки 2 (до первого пробела) после вхождения строки 1. В противном случае возвращается пустая строка. Пример : (string-left-trim 'at 'atom and list) возвращает в качестве результата строку OM.

Функция (char <строка> <n>) возвращает в качестве результата n-й символ строки, отсчет производится с 0. Пример : (char “cat Kuzya” 5) возвращает \u в качестве результата.

Функция (string= <строка 1> <строка 2>) возвращает результат сравнения строк, при этом строчные и заглавные буквы различаются.

Функция (substring <строка> <n> <m>) возвращает в качестве результата подстроку исходной строки, начиная с n-го и кончая m-м символом. Отсчет символов производится с 0.

Функция (unpack <строка>) преобразует строку в список символов, который возвращается в качестве результата. Пример : (unpack “Эксперимент на собаках”) возвращает в качестве результата список : (Э к с п е р и м е н т | | н а | | с о б а к а х).

8.2. Задание на лабораторную работу.

Часть 1. Реализация текстового редактора.

Разработать программу, реализующую основные функции текстового редактора, а именно :

- Работу в окне заданного размера, размер окна задается пользователем, окно должно быть заключено в рамку;
- Загрузку текста из файла на внешнем носителе;
- Сохранение текста в файл;
- Перемещение курсора по тексту;
- Прокрутку текста в окне на одну страницу вверх и вниз;
- Переход к началу и концу текущей строки;
- Вставку новой строки;
- Форматирование текста – вывод его на экран с заданным количеством символов в строке.

При реализации редактора должны быть задействованы управляющие клавиши. Пользовательский интерфейс программы должен включать в меню. Рекомендуется использовать глобальные переменные для запоминания таких текущих параметров редактирования, как координаты текущего положения курсора, параметры окна.

Часть 2. Реализация контекстно-свободной грамматики.

- В соответствии с номером варианта по Таблице 8.1 формализовать синтаксическое правило русского языка и построить КС-грамматику;
- Реализовать генератор синтаксически корректных фраз русского языка в соответствии с построенной КС-грамматикой.
- На основе построенной КС-грамматики реализовать проверку грамматической корректности произвольной последовательности слов, задаваемой пользователем. Программа должна обеспечивать подготовку текста в текстовом редакторе, сохранение созданного текста в текстовом файле на диске и загрузку текста из файла.
- Построить и графически представить синтаксическое дерево для генерируемой/проверяемой фразы русского языка.

Кроме того, должны быть предусмотрены следующие возможности :

- Возможность расширения моделируемого подмножества русского языка путем добавления новых продукций к грамматике, для чего должен быть реализован редактор продукций. Продукции грамматики должны сохраняться в отдельном файле на диске.
- Возможность просмотра в отдельном окне перечня продукций (правил грамматики), задействованных в процессе генерации или проверки грамматической корректности.

Таблица 8.1. Варианты заданий.

Вариант.	Синтаксическое правило
1, 13	Двусоставное простое предложение с простым глагольным сказуемым.
2, 14	Односоставное простое предложение с составным глагольным сказуемым.
3, 15	Двусоставное простое предложение с составным именным сказуемым.
4, 16	Простое предложение с вводными словами.
5, 17	Простое предложение с обособленным определением.
6, 18	Простое предложение с обособленным приложением.
7, 19	Простое предложение с обособленным обстоятельством
8, 20	Сложносочиненное предложение.
9, 21	Сложноподчиненное предложение.
10, 22	Бессоюзное сложное предложение.
11, 23	Двусоставное простое предложение с составным глагольным сказуемым.
12, 24	Простое предложение с однородными членами.

Часть 3. Построение грамматики определенных дизъюнктов.

Для синтаксического правила, реализованного в части 2, построить ГОД. При построении ГОД необходимо ввести учет согласования рода и числа во фразах языка, а также предусмотреть механизм учета семантического контекста для исключения генерации семантически абсурдных фраз.

Все генерируемые и анализируемые фразы для частей 2 и 3 задания должны быть простыми распространенными предложениями русского языка !

8.3. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :

- Формулировку цели и постановку задач проводимых исследований;
- Анализ задания, формальное определение грамматик применительно к конкретному синтаксическому правилу русского языка, тестовые примеры для демонстрации отличий в представлении правил;
- Выбор метода решения с обоснованием, описание процесса разработки программ;

- Сравнительный анализ (на реализованном тестовом примере) формализмов КС-грамматик и ГОД применительно к задаче синтаксического анализа фраз русского языка;
- Текст программы с комментариями и обоснованиями.

ЛАБОРАТОРНАЯ РАБОТА № 9

ОРГАНИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА НА ЕСТЕСТВЕННОМ ЯЗЫКЕ К ДИНАМИЧЕСКОЙ БАЗЕ ДАННЫХ.

Цель работы.

Целью лабораторной работы является изучение возможностей функциональных языков по представлению и использованию семантической информации в задаче организации интерфейса на естественном языке к структурированному источнику данных.

9.1. Краткие теоретические сведения.

В лабораторной работе №7 мы рассмотрели простейший пример реализации табличной (реляционной) модели данных средствами функционального языка. В настоящий момент реляционные БД являются наиболее распространенным типом структурированных источников данных.

В настоящей работе рассматривается подход к организации интерфейса на Естественном Языке, основанный на использовании ключевых слов.

Анализ ключевых слов - метод анализа ЕЯ-высказываний на предмет наличия ключевых слов, которые становятся значениями объектов предикатов. При этом компьютер одинаково реагирует на различные варианты входного текста, наличие грамматической правильности предложений не является обязательным, роль играет лишь наличие ключевых слов.

При разработке ЕЯ-интерфейса к БД с ключевым словом связывается либо характер действия программы (что хочет пользователь), либо значение некоторого атрибута искомого объекта. Например, при поиске по БД предприятий таким атрибутом может быть торговая марка (Старорусприбор, Новтрак, Квант, Планета), а ключевыми словами, определяющими действия – "Найти", "Выдать", "Вывести", "Выход". Процедура анализа входного текста с целью выделения ключей состоит из трех шагов :

- Ввод команды как строки символов;
- Преобразование введенной строки в список слов;

– Идентификация ключевых слов.

Для преобразования строки в список слов требуется написание функции, которая преобразует строку в список слов, причем под словом здесь понимается любая строка последовательность символов от одного разделителя до другого. В качестве разделителя может выступать пробел, символ возврата каретки, перевода строки, !,",#,$. Указанная функция может быть построена с применением встроенных функций обработки строк, функций ввода и вывода информации.

```
; Функция выделения подстроки до ближайшего разделителя :
; пробела, символа возврата каретки, перевода строки, !,",#,$

; Строка обрабатывается в распакованном виде - как список символов.

(defun substr_before (input_lst)
  ((or (null input_lst)
       (equal (car input_lst)(ascii 32))
       (equal (car input_lst)(ascii 10))
       (equal (car input_lst)(ascii 13))
       (equal (car input_lst)(ascii 33))
       (equal (car input_lst)(ascii 34))
       (equal (car input_lst)(ascii 35))
       (equal (car input_lst)(ascii 36))
       (equal (car input_lst)(ascii 44))
       (equal (car input_lst)(ascii 59))) nil)
  (cons (car input_lst)(substr_before (cdr input_lst)))
)

; Функция выделения подстроки после разделителя
; Строка обрабатывается в распакованном виде - как список символов.

(defun substr_after (input_lst)
  ((null input_lst) nil)
  ((or (equal (car input_lst)(ascii 32))
       (equal (car input_lst)(ascii 10))
       (equal (car input_lst)(ascii 13))
       (equal (car input_lst)(ascii 33))
       (equal (car input_lst)(ascii 34))
       (equal (car input_lst)(ascii 35))
       (equal (car input_lst)(ascii 36))
       (equal (car input_lst)(ascii 44))
       (equal (car input_lst)(ascii 59)))(cdr input_lst))
  (substr_after (cdr input_lst))
)
```

```
; Разделение строки на лексемы
; Строка обрабатывается в распакованном виде - как список символов

(defun convers_lst (input_lst)
    ((null input_lst) nil)
    (cons (pack (substr_before input_lst))
            (convers_lst (substr_after input_lst))
    )
)

; Головная функция разделения строки на лексемы
; Аргумент - строка символов
; Значение - список символьных строк

(defun convers (sentence)
    (convers_lst (unpack sentence))
)
; Запрос строки и запуск головной функции
(defun main ()
    (princ "Введите строку : ")
    (clear-input)
    (convers (read-line)))
```

Один из наиболее распространенных и очевидных способов идентификации ключевых слов основан на частоте применения определенных грамматических конструкций : ключевые слова определяются позицией, которую они занимают в предложении. В настоящей работе рассматривается случай, когда ключевыми являются первое и последнее слово предложения. Первое слово определяет выполняемое программой действие, последнее – интересующий нас объект, информацию о котором мы хотим найти в БД. Работа с первым (командным) ключевым словом может быть организована по аналогии с обработкой пунктов меню (см. *Лабораторную работу №6*). Здесь следует особо отметить необходимость выделения префикса, общего для разных поверхностных форм выражения одной и той же команды во входном ЕЯ-запросе. К примеру, применительно к поиску информации в качестве таких префиксов можно выделить "Най", "Выда", "Выве". Для выхода из программы это будут "Вых", "Вый", "exit", "quit". Выделение префиксов можно организовать с применением встроенной функции SUBSTRING, см. *Лабораторную работу №8*.

9.2. Задание на лабораторную работу.

Дополнить реализованную в лабораторной работе №7 Систему Управления Базой Данных пользовательским интерфейсом на Естественном Языке[1]. Подсистема понимания ЕЯ реализуется на основе описанной выше стратегии с использованием ключевых слов. Для формирования пользователем запроса на Естественном Языке в произвольной форме в состав системы должен быть включен текстовый редактор (см. *Лабораторную работу №8*).

9.3. Содержание отчета по лабораторной работе.

Отчет по лабораторной работе должен содержать :

- Формулировку цели и постановку задач проводимых исследований;
- Анализ задания, выбор метода решения с обоснованием, описание процесса разработки программ, полученные результаты и их анализ (обоснование);
- Текст программы с комментариями и обоснованиями.;
- Выводы по проведенным машинным экспериментам.

СПИСОК ЛИТЕРАТУРЫ

1. Хювенен Э., Сеппянен Й. Мир Лиспа. В 2-х т. Пер. с финск. – М.: Мир, 1990.
2. Вирт Н. Алгоритмы + структуры данных = программы : Пер. с англ. - М.: Мир, 1985. – 406 с., ил.
3. Вирт Н. Алгоритмы и структуры данных : Пер. с англ. – СПб.: Невский диалект, 2001. – 351 с.
4. Ахо А.В., Хопкрофт Д.Э., Ульман Д.Д. Структуры данных и алгоритмы : Пер. с англ.: Уч.пос. – М.: Издательский дом “Вильямс”, 2000. – 382 с., ил.
5. Клоксин У., Меллиш К. Программирование на языке Пролог : Пер. с англ. – М.: Мир, 1987. – 336 с.
6. Матусевич М.И. Введение в общую фонетику. Пособие для студентов ун-тов и пед. ин-тов. – М.: Учпедгиз, 1959. - 135 с., ил.

[1] В настоящей работе рассмотрение составляющих естественно-языкового интерфейса ведется на материале русского языка. Допускаются варианты выполнения данной работы для других широко употребляемых на сегодняшний день Естественных Языков (английского, немецкого, французского и т.д., смотри также [19]).

7. Белоногов Г.Г. и Богатырев В.И. Автоматизированные информационные системы. Под ред. К.В. Тараканова. - М.: Сов. радио, 1973. – 328 с.
8. Мельчук И.А. Опыт теории лингвистических моделей "смысл⇔текст" : Семантика, синтаксис / И.А.Мельчук.-[Переизд.]. // Школа "Языки русской культуры". Москва, 1999. - 345 с.
9. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT : Пер. с англ. – М.: Финансы и статистика, 1991. – 543 с.
10. Попов Э.В. и др. Искусственный интеллект. – В 3-х кн. – М.: Радио и связь, 1990.
11. Машинный фонд русского языка : идеи и суждения : [Сборник] / АН СССР, Институт русского языка, Научный совет по лексикологии и лексикографии; Отв. Ред. Ю.Н. Караулов. – М.: Наука, 1986. - 239 с.
12. Ульман Дж. Основы систем баз данных : Пер. с англ. - М.: Финансы и статистика, 1983. – 334 с., ил.
13. Хомский Н. Аспекты теории синтаксиса : Пер. с англ. – М.: Изд. Моск. Ун-та, 1972. – 260 с.
14. Гладкий А.В. Синтаксические структуры естественного языка в автоматизированных системах общения. – М.: Наука, 1985. – 143 с., ил.
15. Виноград Т. Программа, понимающая естественный язык : Пер. с англ. – М.: Мир, 1976. – 294 с.
16. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов : Пер. с англ. – М.: Мир, 1979. – 654 с.
17. Бек Леланд Л. Введение в системное программирование : Пер. с англ. – М.: Мир, 1988. – 448 с., ил.
18. Белоногов Г.Г., Новоселов А.П. Автоматизация процессов накопления, поиска и обобщения информации. – М.: Наука, 1979. – 256 с.
19. Нариньяни А.С. Автоматическое понимание текста - новая перспектива // Труды международного семинара Диалог-97 по компьютерной лингвистике и ее приложениям. - Москва, 1997, с. 203-208.
20. Ин Ц., Соломон Д. Использование Турбо-Пролога : пер. с англ. - М.:Мир, 1993. - 608 с., ил.