Министерство образования Республики Беларусь
Белорусский национальный технический университет
Кафедра «Программное обеспечение вычислительной техники
и автоматизированных систем»

ФУНКЦИОНАЛЬНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Лабораторный практикум по дисциплине “**Функциональное и логическое программирование**” для студентов специальностей 1-40 01 01 “Программное обеспечение информационных технологий” и 1-40 01 02 «Информационные системы и технологии (по направлениям)»

Учебное электронное издание

Минск 2012

УДК 681.3

Предлагаемые лабораторные работы ставят своей целью закрепление теоретического материала и приобретение студентами практических навыков программирования на языке логического программирования Пролог (в среде Visual Prolog версии 5.2) и языке функционального программирования Лисп (в системе программирования muLisp).

В каждой лабораторной работе указывается цель работы, краткие справочные данные по теме работы, содержание задания, варианты заданий и содержание отчета по работе. В конце лабораторной работы даются контрольные вопросы по выполняемой теме. Некоторые вопросы могут предполагать знания, не изложенные в справочных данных к работе. В этом случае требуется использование конспекта лекций, дополнительной литературы и, возможно, сведений, почерпнутых из Internet.

Составители: А.Т.Ковальков, И.А.Ковалькова

Белорусский национальный технический университет
пр-т Независимости, 65, г. Минск, Республика Беларусь
Тел.(017) 292-77-52 факс (017) 292-91-37
Регистрационный № БНТУ/ФИТР49 –54.2012

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА 1.

ОСНОВЫ РАБОТЫ С Visual Prolog В РЕЖИМЕ Test Goal

1. Цель работы: ознакомление с режимом Test Goal системы программирования Visual Prolog.

2. Краткие справочные данные. При использовании утилиты **Test Goal** требуется определить некоторые (не предопределенные) опции компилятора Visual Prolog. Для этого необходимо выполнить следующие действия:

2.1. Запустить среду визуальной разработки **Visual Prolog** двойным щелчком мыши по пиктограмме.

2.2. Создать новый проект (выбрать команду **Project – New Project**, активизируется диалоговое окно **Application Expert**).

2.3.Определить имя проекта и рабочий каталог.

Имя проекта набирается в поле **Project Name**. После щелчка в поле **Name of .VPR File** появляется имя проекта с расширением **.vpr**.

На вкладке **Target** выбрать параметры: **Windows32** в списке **Platform**, **Easywin** – в списке **UI Strategy**, **exe** – в списке **Target Type**, **Prolog** – в списке **Main Program**.

В поле **Base Directory** на вкладке **General** указать директорий для проекта (можно создать новый директорий, используя кнопку **Browse…**).

Нажать кнопку **Create** для того, чтобы создать файлы проекта по умолчанию.

2.4. Открыть окно редактора (использовать команду меню **File –New**). Появится новое окно редактирования с именем **noname.pro**. Редактор среды визуальной разработки – стандартный текстовый редактор. Можно использовать клавиши управления курсором и мышь так же, как и в других редакторах. Он поддерживает команды **Cut, Copy, Delete, Undo** и **Redo**, которые находятся в меню **Edit**. В меню **Edit** также показаны комбинации «горячих» клавиш для этих действий.

2.5. Набрать текст программы (см. в п.4).

2.6. Выполнить программу (команда **Project – Test Goal** или комбинация клавиш **Ctrl-G**).

3. Обработка ошибок. При наличии ошибок в программе отобразится окно **Errors (Warnings)**, которое будет содержать список обнаруженных ошибок. Дважды щелкнув на одной из этих ошибок, попадаем на место ошибки в исходном тексте. Можно воспользоваться клавишей **F1** для вывода на экран интерактивной справочной системы Visual Prolog. Когда окно помощи откроется, следует щелкнуть по кнопке **Search**, набрать номер ошибки, и на экране появится соответствующее окно помощи с более полной информацией об ошибке.

4. Проверка правильности настройки системы. Для проверки того, что система настроена должным образом, можно набрать в окне редактора, например, следующий текст:

GOAL write (“Добро пожаловать”),nl.

Этого достаточно для программы, чтобы она могла быть выполнена. Результат выполнения программы

Добро пожаловать

yes

будет расположен в отдельном окне, которое необходимо закрыть перед тем, как выполнять другую программу.

5. Установка шрифта. Если имеются проблемы с использованием в программе кириллицы, следует щелкнуть правой клавишей мыши в любом месте поля редактора, в

появившемся контекстном меню выбрать команду **Font…**, затем в открывшемся диалоговом окне выбрать шрифт **System** или **Tahoma**.

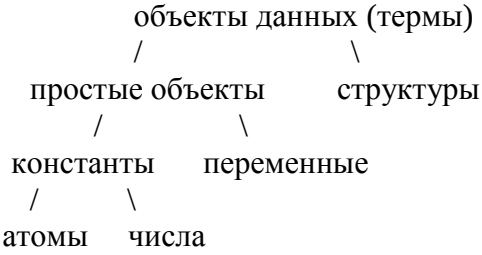
ЛАБОРАТОРНАЯ РАБОТА 2.

СОСТАВЛЕНИЕ ПРОСТЕЙШЕЙ ПРОЛОГ-ПРОГРАММЫ

1. Цель работы: приобретение практических навыков составления, отладки и выполнения простейшей программы в системе программирования Visual Prolog (режим Test Goal).

2. Краткие справочные сведения.

Объекты данных в Прологе называются термами. **Терм** может быть: константой, переменной, структурой (составной терм). Классификация объектов данных приведена на схеме:



Атом - это синтаксически неделимый терм. В качестве атома могут быть: **имя** Visual Prolog (имя – это начинающаяся с буквы последовательность букв, цифр и символа подчеркивания), строчное представление числа (например, “1048”, “25.6”), отдельный символ (кроме пробела).

Числа в Visual Prolog записываются точно так же, как и в других языках программирования.

Константы относятся к одному из стандартных доменов (типов), приведенных в таблице 1 (кроме приведенных в таблице Visual Prolog имеет и другие стандартные домены: byte, short, ushort, word, unsigned, long, ulong, dword).

Таблица 1. Стандартные домены

Тип	Ключевое слово	Диапазон значений	Примеры
Символы	char	все возможные одиночные символы	’a’,’B’,’?’
Целые числа	integer	-2147483648 … 2147483647	-15, 1235, 9
Действ. числа	real	1E-307 ... 1E308	4.8, 2.45E-8
Строки	string	последовательность символов (до 250)	“Минск” “a_min”
Символические имена	symbol	1. Последовательность букв, цифр, знака подчеркивания (первый символ - строчная буква)	read_data1
		2. Последовательность заключенных в кавычки символов	“Delete f1” “Турбо-Си”
Файлы	file	допустимое в Windows имя файла	a.txt progr.pro

Переменная - имя, начинающееся с большой (прописной) буквы или знака подчеркивания. Когда значение переменной неважно, то в качестве имени переменной используется знак подчеркивания. Такая переменная называется **анонимной**.

Структуры (сложные термы) - это объекты, которые состоят из нескольких компонент. Структура записывается с помощью указания ее **функтора** и **компонент**. Компоненты заключаются в круглые скобки и разделяются запятыми. Число компонент в структуре называется **арностью** структуры.

Пример структуры: data_r(12,mart,1962). Здесь data_r – функтор, 12, mart, 1962 – компоненты. Арность приведенной структуры равна трем.

Программа на Visual Prolog состоит из нескольких разделов, каждому из которых предшествует ключевое слово. Типичная простейшая структура программы представлена ниже:

```
/*    комментарии    */
domains
   <описание доменов (типов данных)>
predicates
   <описание предикатов>
clauses
   <предложения или утверждения>
goal
   <целевое утверждение>
```

В программе не обязательно наличие всех разделов. Обычно в программе должны быть по крайней мере разделы predicates и clauses.

В разделе domains объявляются любые нестандартные домены, используемые для аргументов предикатов. Домены на Прологе являются аналогами типов в других языках.

Наиболее простым способом описания доменов в разделе domains является следующий:

name=d, где name – нестандартные домены, определенные пользователем; d - один из стандартных доменов (integer, real, char, string, symbol и др.).

Предикат (отношение) в общем случае - это структура вида:

predname(komp1,komp2,...),

где predname - имя предиката,

komp1,... - **типы** компонентов, описанные в разделе domains или стандартные типы.

Например,

```
domains
   fio=string
   den,god=integer
   mes=symbol
predicates
   anketa(fio,den,mes,god)
```

Если в предикатах используются только стандартные типы данных, то раздел domains может отсутствовать:

```
predicates
   anketa(string,integer,symbol,integer)
```

Предикат может состоять только из одного имени, напр.,

```
predicates
   result
```

В разделе clauses размещаются **предложения** (утверждения). Предложение представляет собой факт или правило, соответствующее одному из объявленных предикатов.

Факт - простейший вид утверждения, которое устанавливает отношение между объектами. Пример факта:

anketa(“Иванов”,5,august,1950).

Этот факт содержит атом anketa, который является именем предиката, и в скобках после него - список термов, соответствующих компонентам этого предиката. **Факт всегда заканчивается точкой**. Факты содержат утверждения, которые всегда являются безусловно верными. Компонентами факта могут быть только константы.

Все предложения раздела clauses, описывающие один и тот же предикат, должны записываться друг за другом.

В разделе goal записывается третий тип предложения – **вопрос (цель)**, состоящий из одного или нескольких целевых утверждений (подцелей), разделенных запятыми и оканчивающихся точкой. Пролог-система рассматривает вопрос как цель, к достижению которой нужно стремиться. При запуске программы Visual Prolog вызывает goal и пытается разрешить все подцели вопроса. Если все подцели в разделе goal истинны, то программа завершается успешно и будут выведены все возможные решения, а также значения всех переменных, используемых в goal. Если же какая-то подцель ложна, то считается, что программа завершается неуспешно.

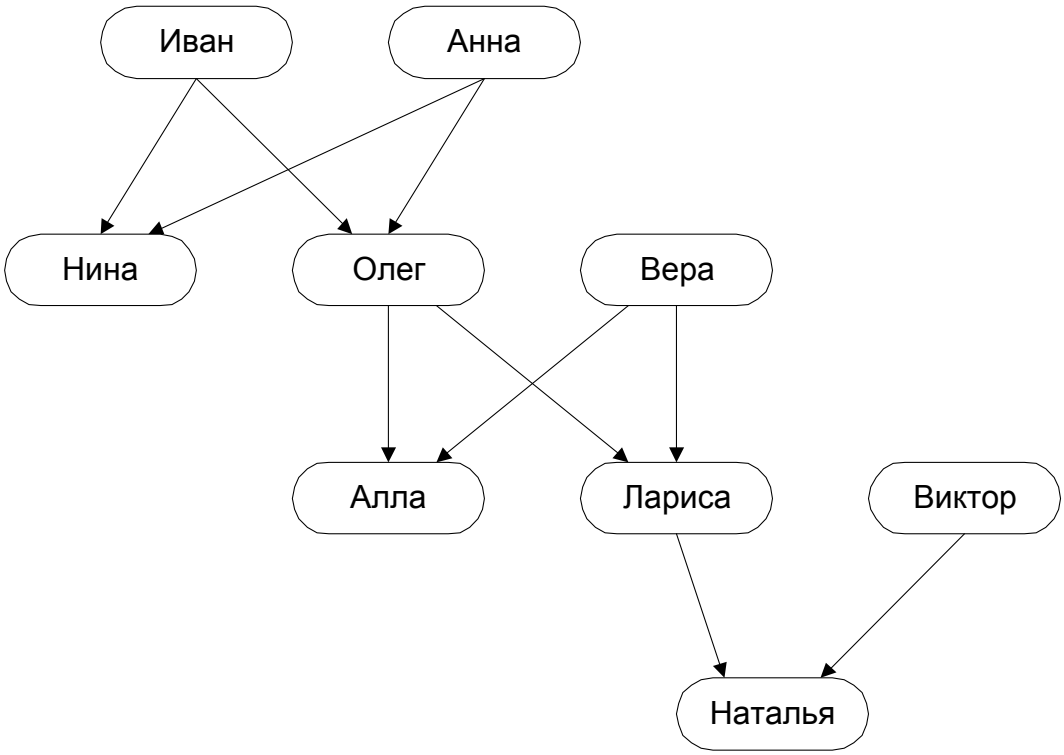


Рис 1. Дерево родственных отношений

3. Содержание задания по лабораторной работе.

Описать средствами Visual Prolog (с помощью фактов) дерево родственных отношений (рис. 1), используя предикат roditel с двумя параметрами: имя родителя и имя ребенка.

В окне диалога сформировать следующие вопросы:

1. Является ли Иван родителем Нины?
2. Является ли Иван родителем Аллы?
3. Кто родители Ларисы?
4. Как зовут детей Олега?
5. Кто родитель родителя Натальи?

6. Кто чей родитель?
7. Есть ли у Нины и Олега общий родитель?
8. Как зовут жену Ивана?
9. Кто у Анны внуки?
10. Есть ли у Ларисы брат или сестра?

4. Содержание отчета.

Отчет по лабораторной работе должен содержать: цель работы, постановка задачи, текст Пролог-программы, вопросы к программе и ответы системы, выводы.

Контрольные вопросы:

1. Где был разработан Пролог?
2. К какой группе языков относится Пролог?
3. Что является теоретической основой Пролога?
4. Что такое терм? Какие объекты данных Пролога могут использоваться в качестве термов?
5. Что может быть в качестве атома?
6. Как записываются в Visual Prolog числа?
7. Чем отличаются записи констант и переменных?
8. Что такое анонимная переменная, когда она используется?
9. Форма записи структур (сложных термов).
10. Из каких основных разделов состоит программа Visual Prolog, назначение этих разделов?
11. Может ли быть в программе Visual Prolog несколько одинаковых разделов?
12. С какого раздела начинается выполнение программы Visual Prolog?
13. В каком разделе программы описываются пользовательские (нестандартные) типы данных?
14. Что такое предикат и в каких разделах программы его имя может использоваться?
15. Что является аналогом предиката в процедурных языках?
16. Какие объекты могут быть в качестве компонент факта?
17. Какой тип предложения записывается в разделе goal?
18. Способы записи комментариев в программе Visual Prolog.

ЛАБОРАТОРНАЯ РАБОТА 3.

ФОРМИРОВАНИЕ ПРАВИЛ

1. Цель работы: приобретение практических навыков формирования правил и использования их в Пролог-программах.

2. Краткие справочные данные.

Правило - это утверждение, истинность которого зависит от некоторых условий. Правило состоит из **заголовка** и **тела**, соединенных символом :-, который читается “если”. Правило, как и факт, заканчивается точкой.

Заголовок является одним из ранее описанных предикатов, в котором в качестве компонентов могут быть переменные. Заголовок правила описывает факт, для определения которого предназначено это правило.

Тело правила описывает подцели, которые должны быть последовательно согласованы с фактами, для того чтобы заголовок правила был истинным. Тело содержит список термов, разделенных запятыми или точкой с запятой (;). При этом:

вместо :- можно использовать if;

вместо , может использоваться and;

вместо ; может использоваться or.

Пример правила:

внук(X,Y):- отец(Y,Z),отец(Z,X),мужчина(X).

Это правило можно прочитать следующим образом: X является внуком Y, если Y является отцом Z и этот Z является отцом X и X - мужского пола.

Важно отметить, что переменная обозначает один и тот же объект только в пределах одного предложения.

3. Содержание задания по лабораторной работе.

Программу лабораторной работы 2 дополнить новыми фактами, позволяющими построить правила для определения следующих целей: отец, мать, сын, дочь, брат, сестра, дядя, тетя, дедушка, бабушка, внук, внучка. Предикат для каждого правила содержит две компоненты, смысл которых аналогичен смыслу компонент правила из примера п.2.

4. Содержание отчета.

В отчете по лабораторной работе должны быть отражены следующие пункты: цель работы, постановка задачи, текст Пролог-программы, вопросы к программе и ответы системы, выводы.

Контрольные вопросы:

1. Что такое правило и из каких частей состоит правило?
2. Различия между фактом и правилом.
3. В каком разделе программы реализуются правила?
4. Обязательно ли описание предиката, реализованного в программе, и где это описание производится?
5. В каких разделах программы могут быть переменные?
6. Область действия переменной в программе Visual Prolog.
7. Какой символ должен быть в качестве первого в имени переменной?
8. С какого символа моет начинаться имя предиката?
9. Что означают символы “:-“, “,” и “;” в программе Visual Prolog?
10. Может ли имя константы в программе Visual Prolog начинаться с прописной буквы?
11. Различия между свободной и связанной переменными.
12. Можно ли изменить значение связанной переменной?
13. Есть ли в Прологе оператор присваивания?
14. Можно ли между предложениями с одним и тем же предикатом вставить новое предложение с другим предикатом?

ЛАБОРАТОРНАЯ РАБОТА 4.

РЕКУРСИЯ

1. Цель работы: приобретение практических навыков составления и отладки программ с использованием рекурсии.

2. Краткие справочные данные.

Мощным средством программирования в Прологе является рекурсия. **Рекурсия** - это определение некоторого отношения через самого себя. Так как в Прологе отсутствуют операторы цикла, то рекурсия служит основным средством программирования циклических процессов.

Классическим примером рекурсии является вычисление факториала F=N! = 1*2*3*...*(N-1)*N. Это выражение можно записать так: N!=(N-1)!*N. Последнее выражение - пример рекурсивных вычислений, т.е. каждый последующий результат получается путем использования предыдущего результата. Для того чтобы рекурсивный метод решения был результативным, он должен в конце концов привести к задаче, решаемой непосредственно. Решить ее позволяют утверждения, называемые граничными условиями (для нашего случая граничным условием является 0!=1). На Прологе процедура вычисления факториала может быть описана следующим образом:

/* Граничное условие */

```
factorial(0,1).   % 0!=1
/*  Рекурсивное правило вычисления F=N!  */
factorial(N,F):-                    % F=N!
                N>0,N1=N-1,
                factorial(N1,F1),    % F1=(N-1)!
                F=F1*N.              % N!=(N-1)!*N
```

При использовании стандартных предикатов ввода-вывода, позволяющих запрограммировать удобный интерфейс пользователя с программой, программа вычисления факториала на Visual Prolog может быть записана в следующем виде:

```
/*  Вычисление факториала F=N!  */
domains
  n=integer
  f=real
predicates
  factorial(n,f)
  result
clauses
  factorial(0,1).
  factorial(N,F):-N>0,N1=N-1, factorial(N1,F1),F=F1*N.
  result:-
    write("Введите число N"),nl,
    write("N="),readint(N),
    factorial(N,F),
    write(N,"!=",F),nl.
goal
  result.
```

В приведенной процедуре вычисления факториала тело правила начинается с рекурсивного вызова определяемого предиката. Такая рекурсия называется **левосторонней (нисходящей).**

Следующая программа демонстрирует другой вариант определения факториала. Здесь **рекурсивный** вызов заменен на **итеративный (восходящая рекурсия).** Итеративной процедурой называется такая процедура, которая вызывает себя только один раз, причем этот вызов расположен в самом конце процедуры. Это так называемая **правосторонняя рекурсия**.

```
/* Программа итеративного вычисления факториала */
predicates
  factorial (integer, real)
  factorial_aux(integer, real, integer, real)
clauses
  factorial(N,F):-factorial_aux(N,F,0,1).
  factorial_aux(N,F,I,P):-
     I<N, NewI=I+1, NewP=P*NewI,
     factorial_aux(N,F,NewI,NewP).
  factorial_aux(N,F,I,F):-I>=N.

goal
  write(“N=”), readinf(N),
  factorial(N,F),
  write(“F=”, F),nl.
```

Для обеспечения итеративного характера процедуры возникла необходимость определить вспомогательный предикат factorial_aux с двумя дополнительными параметрами: первый из них служит для проверки условия окончания цикла, второй – для возврата вычисленного значения через всю порожденную итеративную последовательность вызовов.

3. Содержание задания по лабораторной работе.

Возвести вещественное число a в целую степень n (степень n может быть положительной, нулевой или отрицательной). Составить два варианта программы:

а) используя рекурсивное выражение a^n=a^(n-1)*a;

б) используя проверку четности степени: для нечетной степени используется рекурсивное выражение варианта а), для четной степени - a^n = a^(n/2)*a^(n/2).

4. Задачи на рекурсию.

Используя рекурсию, запрограммировать решение следующих задач:

4.1. Найти сумму целых последовательных чисел в интервале от M до N (M<=N).

4.2. Просуммировать целые последовательные нечетные числа, т.е. 1+3+5+7+…, от 1 до N (учесть, что при вводе значения N оно может быть четным).

4.3. Вычислить значения следующих функций, используя разложение в ряд. Разработать два варианта программы – рекурсивный и итеративный, в итеративном варианте задавать точность вычисления функции. Полученный результат сверить со значением соответствующей стандартной функции для вычисляемого аргумента.

a) e = 1 + 1/1! + 1/2! + …+ 1/k! + … (k=0,1,2,…);
b) ln2 = 1 – 1/2 +1/3 - … + (-1)^(k-1)/k + … (k=1,2,3,…);
c) π/4 = 1 – 1/3 + 1/5 - … + (-1)^(k-1)/(2*k-1) + … (k=1,2,3,…);
d) π^2/6 = 1 + 1/2^2 + 1/3^2 + … + 1/k^2 + … (k=1,2,3,…);
e) π^2/12 = 1 – 1/2^2 + 1/3^2 - … + (-1)^(k-1)/k^2 + … (k=1,2,3,…);
f) π^2/8 = 1 + 1/3^2 + 1/5^2 + … + 1/(2*k-1)^2 + … (k=1,2,3,…);
g) e^x = 1 + x/1! + x^2/2! + … + x^n/n! + … (n=0,1,2,…);
h) sin(x) = x – x^3/3! + x^5/5! - … + (-1)^(m-1)*x^(2*m-1)/(2*m-1)! + … (m=1,2,3,…);
i) cos(x) = 1 – x^2/2! + x^4/4! - … + (-1)^m*x^(2*m)/(2*m)! + … (m=0,1,2,…);
j) arctg(x) = x – x^3/3 + x^5/5 - … + (-1)^(m-1)*x^(2*m-1)/(2*m-1) + … (m=1,2,3,…).

5. Содержание отчета.

В отчете должны быть следующие пункты: цель работы, постановка задачи, исходный текст программы, результаты тестирования, выводы.

Контрольные вопросы:

1. Что такое рекурсия?
2. Основная цель использования рекурсии в программах Visual Prolog.
3. Назначение граничного условия в рекурсивном правиле.
4. Принцип построения граничного условия.
5. Может ли быть в рекурсивном правиле более одного граничного условия?
6. Как программируются разветвления в программах Visual Prolog?
7. Левосторонняя (нисходящая) и правосторонняя (восходящая) рекурсии.
8. Принципиальная разница между рекурсивной и итеративной процедурами.
9. Правила сопоставимости двух термов.
10. Поиск с возвратом в программах Visual Prolog.
11. Поточный шаблон предиката.
12. Стандартные предикаты ввода-вывода Visual Prolog.

ЛАБОРАТОРНАЯ РАБОТА 5.
ИСПОЛЬЗОВАНИЕ ОТСЕЧЕНИЯ В ПРОЛОГ-ПРОГРАММАХ

1. Цель работы: приобретение практических навыков использования отсечения в программах на Турбо-Прологе.

2. Краткие справочные данные.

В процессе достижения цели Пролог-система осуществляет автоматический перебор вариантов, делая возврат при неуспехе какого-либо из них. Такой перебор – полезный программный механизм, поскольку он освобождает пользователя от необходимости программировать этот перебор самому. С другой стороны, ничем не ограниченный перебор может быть источником неэффективности программы. Поэтому иногда требуется его ограничить или исключить вовсе. Для этого в Прологе предусмотрено специальное целевое утверждение **!**, называемое **отсечением** (cut).

Отсечение реализуется следующим образом: после согласования целевого утверждения, стоящего перед отсечением, все предложения с тем же предикатом, расположенные после отсечения, не рассматриваются.

Можно выделить три основные случая использования отсечения:

1) **для устранения бесконечного цикла**;

Пример: вычисление суммы 1 + 2 + … + N.

```
сумма( 1, 1 ) :- !.
сумма( N, R ) :- N1=N-1, сумма( N1, R1 ), R = R1 + N.
```

Если граничное условие будет записано в виде

```
сумма( 1, 1 ).
```

без отсечения, то сопоставление головы правила с запросом в разделе goal будет происходить успешно и при N <= 0, т.е. делается попытка доказать цель сумма(0, R), что в свою очередь приводит к цели сумма(-1, R) и т.д., т.е. образуется бесконечный цикл.

2) **при программировании взаимоисключающих утверждений**;

Пример: нахождение большего числа среди двух чисел X и Y можно запрограммировать в виде отношения

max(X, Y, Max), где Max = X, если X >= Y и Max = Y, если Y > X.

Это соответствует двум предложениям программы:

```
max1( X, Y, X ) :- X >= Y.
max1( X, Y, Y ) :- X < Y.
```

Эти предложения являются взаимоисключающими, т.е. если выполняется первое, второе обязательно терпит неудачу, и наоборот. Поэтому возможна более экономная запись при использовании отсечения:

```
max2( X, Y, X ) :- X >= Y, !.
max2( _, Y, Y ).
```

3) **при необходимости неудачного завершения доказательства цели**;

Пример: присвоение какому-либо объекту категории в зависимости от величины заданного критерия в соответствии с таблицей 2.

Таблица 2. Присвоение категории

Критерий	Категория
свыше 80 до 100	A
свыше 40 до 80	B
от 0 до 40	C

Возможный вариант программы:

```
категория( Кр, _ ) :- Кр > 100, !, fail; Кр < 0, !, fail.
категория( Кр, ‘A’) :- Кр > 80, !.
категория( Кр, ‘B’) :- Кр > 40, !.
категория( _, ‘C’).
```

При запросе категория(200, X) произойдет сопоставление запроса с головой первого утверждения. Цель Кр > 100 будет достигнута. Затем будет доказана цель-отсечение. Но когда встретится предикат fail, который всегда вызывает состояние неудачи, то стоящий перед ним предикат отсечения остановит работу механизма возврата и в результате ответом на запрос будет No Solution (Нет решения).

Комбинация !, fail вводит возникновение неудачи.

3. Содержание задания по лабораторной работе.

Определить возрастной статус человека по известному году рождения в соответствии с таблицей 3. Разработать два варианта программы: без отсечения и с использованием отсечения.

Таблица 3. Возрастной статус человека

Возраст, лет	Статус
< 2	младенец
2 – 12	ребенок
12 – 16	подросток
16 – 25	юноша
25 – 70	мужчина
70 – 100	старик
> 100	долгожитель

4. Содержание отчета.

В отчете по лабораторной работе должны быть отражены следующие пункты: цель работы, постановка задачи, тексты Пролог-программ, результаты тестирования программ, выводы.

Контрольные вопросы:

1. Для чего используется отсечение?
2. Как обозначается отсечение в программе?
3. Механизм работы отсечения.
4. «Красное» и «зеленое» отсечения.
5. Какое отсечение, «красное» или «зеленое» , в процедуре max2 ?
6. Повлияет ли на результат удаление отсечения в процедуре max2 ?
7. Что происходит после выполнения комбинации !, fail?
8. Могут ли располагаться подцели после предиката fail ?
9. Декларативный и процедурный смысл Пролог-программы.
10. Важен ли порядок следования предложений в процедурах max1 и max2 ?

ЛАБОРАТОРНАЯ РАБОТА 6.

РАБОТА СО СПИСКАМИ

1. **Цель работы**: приобретение навыков работы со списками в программах на Visual Prolog.

2. **Краткие справочные данные**.

Список - это специальный вид сложного терма, состоящий из последовательности термов, заключенных в квадратные скобки и разделенных

запятой. Например, [1,2,-5,4,0]. Такой список в программе описывается следующим образом:

```
domains
   list=integer*
```

Списки являются основной структурой данных в программах на Прологе. Для удобства обработки списков введены два понятия: голова (head) и хвост (tail). Так, для списка [1,2,3] элемент 1 является головой списка, а список [2,3], включающий остальные элементы, является хвостом.

В таблице 4 приведены основные примеры обработки списков:

Таблица 4. Примеры обработки списков

Список	Голова	Хвост
[’a’,’b’,’c’]	‘a’	[’b’,’c’]
[1]	1	[ ]
[ ]	не определено	не определено
[[1,2,3],[3,4],[]]	[1,2,3]	[[3,4],[]]

Для отделения головы списка от хвоста используется символ |. Например, для списка [X|Y] X - голова списка, Y - хвост.

Примеры правил для работы со списками:

а) ввод_списка1(Кол_эл, Сп) (integer, integer*) : (i,o) – ввод списка Сп, содержащего Кол_эл элементов.

```
ввод_списка1(0, []).
ввод_списка1(N, [H|T]):-readint(H), N1=N-1,
                        ввод_списка1(N1,T).
```

б) ввод_списка2(Сп) (integer*) : (o) – ввод списка Сп, содержащего произвольное количество элементов.

```
ввод_списка2([H|T]):-readint(H), ввод_списка2(T).
ввод_списка2([]).
```

в) объедин_списков(Сп1, Сп2, Сп3) (integer*, integer*, integer*) : (i,i,o) – список Сп3 получен путем добавления элементов списка Сп2 в конец списка Сп1.

```
объедин_списков([], L, L).
объедин_списков([H|T], L2, [H|T3]):- объедин_списков(T1, L2, T3).
```

г) принадл(Эл, Сп) (integer, integer*) : (i, i) – проверка принадлежности элемента Эл списку Сп.

```
принадл(E, [H| _ ]):-E=H.
принадл(E, [H|T]):-E<>H, принадл(E, T).
```

д) колич_эл(Сп, Кол_эл) (integer*, integer) : (i, o) – подсчет количества элементов Кол_эл в списке Сп.

```
колич_эл([], 0).
колич_эл([ _ |T], N):- колич_эл(T, N1), N=N1+1.
```

е) удал_перв_эл(Сп1, Сп2) (integer*, integer*) : (i, o) – список Сп2 получен путем удаления первого элемента из списка Сп1.

```
удал_перв_эл([ _ | T], T).
```

ж) удал_посл_эл(Сп1, Сп2) (integer*, integer*) : (i, o) – список Сп2 получен путем удаления последнего элемента из списка Сп1.

```
удал_посл_эл([ _ ], []).
удал_посл_эл([H|T], [H|T1]):- удал_посл_эл(T, T1).
```

з) выдел_перв_эл(Сп, Эл) (integer*, integer) : (i, o) – Эл – это первый элемент списка Сп.

```
выдел_перв_эл([H| _ ], H).
```

и) выдел_посл_эл(Сп, Эл) (integer*, integer) : (i, o) – Эл – это последний элемент списка Сп.

```
выдел_посл_эл([X], X).
```

выдел_посл_эл([_ |T], X):- выдел_посл_эл(T, X).

к) доб_эл_в_начало(Эл, Сп, Сп1) (integer, integer*, integer*) : (i, i, o) – список Сп1 получен путем добавления элемента Эл в начало списка Сп.

доб_эл_в_начало(E, [H|T], [E,H|T]).

л) доб_эл_в_конец (Эл, Сп, Сп1) (integer, integer*, integer*) : (i, i, o) – список Сп1 получен путем добавления элемента Эл в конец списка Сп.

доб_эл_в_конец(E, [], [E]).

доб_эл_в_конец(E, [H|T], [H|T1]):- доб_эл_в_конец(E, T, T1).

Приведем полный текст Пролог-программы для следующей задачи. Из исходного списка, содержащего заданное количество элементов, получить новый список, каждый элемент которого равен увеличенному в два раза соответствующему элементу исходного списка.

```
/* Программа увеличения всех элементов списка вдвое */
domains
   list=real*
predicates
  read_list(integer,list)
  new_list(list,list)
  result
clauses
/* Ввод списка из N элементов */
  read_list(0,[]).
  read_list(N,[H|T]):-readreal(H),N1=N-1, read_list(N1,T).
/*  Преобразование исходного списка  */
  new_list([],[]).
  new_list([H1|T1],[H2|T2]):-H2=H1*2,new_list(T1,T2).
/*  Целевое утверждение  */
  result:-write("Введите число элементов списка"),nl,
       write("N="),readint(N),nl,
       write("Вводите элементы списка")nl,
       read_list(N,L),
       new_list(L,L1),
       write("Исходный список L="),
       write (L),nl,nl,
       write("Новый список L1="),
       write (L1),nl.
goal
  result.
```

3. **Содержание задания по лабораторной работе**.

Из списка L1 получить список L2, очередной элемент которого равен среднему арифметическому очередной тройки элементов списка L1. Если число элементов списка L1 не кратно 3, то последний элемент списка L2 получается делением на 3 одного или суммы двух последних элементов списка L1. Список L1 вводится по подсказке с экрана. В результате выполнения программы должны выдаваться исходный L1 и результирующий L2 списки.

4. **Задачи по работе со списками**.

4.1. Переместить первый списка элемент в конец списка.

4.2. Переместить последний элемент списка в начало списка.

4.3. Удалить два последних элемента списка.

4.4. Удалить первое вхождение заданного элемента в списке.

4.5. Удалить все вхождения заданного элемента в списке.

4.6. Удалить из списка повторяющиеся элементы.

4.7. Определить среднее арифметическое элементов списка.

4.8. Из заданного списка образовать полиндром. Полиндром - это список, который читается одинаково как слева направо, так и в обратном направлении. Пример полиндрома - [1,2,3,2,1].

4.9. Произвести циклический сдвиг элементов списка:

- на одну позицию влево (вправо),
- на N позиций влево (вправо).

4.10. Произвести сдвиг элементов списка (не циклический):

- на одну позицию влево (вправо),
- на N позиций влево (вправо).

4.11. Разбить исходный список на два списка примерно одинаковой длины.

4.12. Разбить список на два: в одном содержатся элементы, меньшие или равные заданному числу, в другом - большие заданного числа.

4.13. Вычислить сумму элементов, стоящих в списке на нечетных местах, и сумму элементов, стоящих на четных местах.

4.14. Вывести первые N положительных элементов списка.

4.15. Подсчитать количество положительных и отрицательных элементов списка.

4.16. Определить разность между максимальным и минимальным элементами списка.

4.17. Все элементы списка уменьшить на величину среднего арифметического этих элементов.

4.18. Удалить начало списка до заданного элемента X (включительно).

4.19. Удалить конец списка, начиная с заданного элемента X.

4.20. Удалить конец списка после заданного элемента X.

4.21. Заменить в списке все элементы со значением X на значения Y.

5. Содержание отчета.

В отчете должны быть отражены следующие пункты: цель работы, постановка задачи, текст программы, результаты тестирования, выводы.

Контрольные вопросы:

1. Что такое список ?
2. Голова и хвост списка. Изображение списка в виде головы и хвоста.
3. Как обозначается пустой список ?
4. Можно ли изобразить пустой список в виде головы и хвоста ?
5. Что означают следующие виды изображения списка: L, [X], [_ , A], [_ |T], [H| _], [_], [_ ,B|C] ?
6. Может ли быть в голове списка более одного элемента ?
7. Принадлежит ли элемент 5 списку [[1, 2, 3], [4, 5, 6]] ?
8. Сопоставимы ли два списка: [5, 6, 7] и [H|T] ?
9. Можно ли в процедурах ввода списка **ввод_списка1** и **ввод_списка2** поменять местами первое и второе предложения ?
10. Как нужно изменить процедуру **удал_перв_эл**, чтобы можно было удалить из списка: а) два первых элемента; б) второй элемент ?
11. Как нужно изменить процедуру **удал_посл_эл**, чтобы из списка были удалены: а) два последние элементы; б) предпоследний элемент ?
12. Как нужно изменить процедуру **выдел_перв_эл**, чтобы можно было выделить из списка: а) два первых элемента; б) второй элемент ?
13. Как нужно изменить процедуру **выдел_посл_эл**, чтобы из списка можно было выделить: а) два последние элементы; б) предпоследний элемент ?

14. Как нужно изменить процедуру **доб_эл_в_начало**, чтобы можно было вставить заданный элемент между первым и вторым элементами списка ?
15. Как нужно изменить процедуру **доб_эл_в_конец**, чтобы заданный элемент вставился перед последним элементом списка ?
16. Можно ли в программе со списками обойтись без раздела domains ?

ЛАБОРАТОРНАЯ РАБОТА 7.

РАБОТА СО СТРОКАМИ

1. **Цель работы**: приобретение практических навыков работы со строками в программах на Visual Prolog.

2. **Краткие справочные сведения**.

Для работы со строками имеется набор стандартных предикатов:

а) Определение длины строки

str_len(Строка,Длина)(string,integer):(i,o),(i,i).

б) Объединение строк

concat(Стр1,Стр2,Стр3)(string,string,string): (i,i,o)(o,i,i)(i,o,i)(i,i,i).

Строки Стр1 и Стр2 объединяются в строку Стр3.

в) Создание подстрок

frontstr(КолСим,ВхСтр,ВыхСтр,Остаток)(integer,string,string,string): (i,i,o,o).

Выходная строка ВыхСтр получается из входной строки ВхСтр отделением начальных символов, количество которых определяется параметром КолСим, остаток строки связывается с переменной Остаток.

г) Разделение строки на две части - первый символ и оставшаяся часть строки

frontchar(Строка,ПервСимв,Остаток)(string,char,string):(i,o,o)(i,i,o)
(i,o,i)(i,i,i)(o,i,i).

Предикат frontchar используется для преобразования строки в список символов. Правило, которое выполняет эту процедуру, следующее:

```
convert("",[]).
convert(Str,[H|T]):-frontchar(Str,H,Ost),convert(Ost,T).
```

д) Проверка, является ли строка именем Visual Prolog

isname(Строка)(string):(i).

Строка, которая является именем, должна удовлетворять следующим требованиям:

- строка может содержать буквы, цифры и символ подчеркивания;
- строка начинается с буквы;
- между символами не должно быть пробелов.

е) Формирование атомов из строк

fronttoken(Строка,Атом,Остаток)(string,string,string):(i,o,o)(i,i,o)(i,o,i)(i,i,i)
(o,i,i).

Атомом могут быть:

- имя Турбо-Пролога, (напр., summa, a_12);
- строчное представление числа, (напр., "1265", "1.25", "1e-6");
- отдельный символ (кроме пробела), (напр., 'A', 'x', '%').

Предикат fronttoken может быть использован для преобразования строки в список атомов (по аналогии с предикатом frontchar).

При работе со строками могут использоваться также предикаты преобразования данных из одного типа в другой:

а) Преобразование прописных букв в строчные или наоборот

upper_lower(Стр1,Стр2)(string,string):(i,i)(i,o)(o,i).

б) Преобразование строки в символ или наоборот

str_char(Строка,Символ)(string,char):(i,o)(o,i)(i,i).

в) Преобразование строки в целое число или наоборот

str_int(Строка,ЦелЧисло)(string,integer):(i,o)(o.i)(i,i).

г) Преобразование строки в действительное число или наоборот
str_real(Строка,ДействЧисло)(string,real):(i,o)(o,i)(i,i).

д) Преобразование символа в число (код ASCII) или наоборот
char_int(Символ,Число)(char,integer):(i,o)(o,i)(i,i).

е) Преобразование терма заданного домена в его строковое представление
term_str(Домен, Терм, Строка)(<домен>,<терм>,string):(i,i,o).

3. **Задание по лабораторной работе**.

Введенную с клавиатуры строку вывести на экран дисплея наоборот.

4. **Задачи по работе со строками**.

4.1. Подсчитать, сколько раз во введенной строке встречается заданная буква.

4.2. Проверить, имеются ли в заданной строке два подряд идущие одинаковые символы.

4.3. В заданном предложении подсчитать количество слов.

4.4. В заданной последовательности символов удалить каждый символ * и повторить каждый символ, отличный от *.

4.5. В словах “килограм”, “граматика”, “грамофон” исправить грамматические ошибки (с помощью составленной программы).

4.6. В последовательности слов, разделенных пробелами, определить количество слов, начинающихся с заданной буквы.

4.7. Из последовательности слов, разделенных пробелами, выбрать слова, у которых первый и последний символы одинаковые.

4.8. В последовательности слов, разделенных пробелами, после каждого слова перед пробелом вставить запятую, кроме последнего слова.

4.9. Дана строка символов. Если в ней нет символа *, то строку оставить без изменения, иначе заменить символ, стоящий после первой *, на символ +.

4.10. Дана последовательность слов, разделенных пробелами. Вывести на экран слова, внутри которых имеется хотя бы одна буква, с которой слово начинается.

4.11. В строке найти самое длинное слово.

4.12. Определить, есть ли в строке цифровые символы; если есть, то вывести их на экран.

4.13. Вывести на экран строчные латинские буквы, которых в строке нет.

4.14. Удалить из строки гласные буквы.

4.15. Подсчитать в строке количество слов, длина которых меньше заданной.

4.16. Заменить в сроке все цифровые символы нулями.

4.17. Разбить строку на две строки примерно одинаковой длины по границе слов.

4.18. Удалить из строки лишние пробелы (начальные, в середине строки, в конце строки).

4.19. Проверить, есть ли в строке заданное ключевое слово.

4.20. Найти в строке слова с числовыми символами.

5. **Содержание отчета**.

В отчете должны быть отражены следующие пункты: цель работы, постановка задачи, тексты программ, результаты тестирования, выводы.

Контрольные вопросы:

1. Что будет получено в качестве результата при выполнении предиката **str_len** с поточным шаблоном **(i,i)** ?
2. Можно ли использовать предикат **concat** для получения остатка строки при задании строки и ее подстроки ?
3. Возможно ли применение предиката **frontchar** со свободрыми второй и третьей переменными ?

4. Можно ли с помощью предиката **frontchar** подсоединить заданный символ к началу строки ?
5. Сколько имеется вариантов поточного шаблона предиката **frontstr** ?
6. Имеется строка, содержащая последовательность имен, разделенных пробелами. Какой предикат можно использовать для преобразования этой строки в список имен ?
7. Напишите правило преобразования строки в список атомов. Какой получится список, если в качестве входного параметра этому правилу будет передана следующая строка: **“abc f17 38de, a5b”**?
8. Назначение предиката **isname**.
9. Сколько вариантов поточного шаблона имеют предикаты преобразования типов (за исключением **term_str**) ? Смысл поточного шаблона **(i,i)** для этих предикатов.
10. С помощью какого предиката преобразования типов можно определить десятичный код символа ?
11. Есть ли стандартный предикат для преобразования целого числа в вещественное и наоборот ?

ЛАБОРАТОРНАЯ РАБОТА 8.

ИСПОЛЬЗОВАНИЕ СОСТАВНЫХ ОБЪЕКТОВ

1. **Цель работы**: приобретение практических навыков в составлении и отладке Пролог-программ с составными объектами.

2. **Краткие справочные данные**.

Объекты утверждений (компоненты) представляют собой данные. Тип простых данных ограничен стандартными типами. Рассмотрим следующее утверждение:

любит (петр,музыка).

Оба объекта **(петр, музыка)** имеют простую структуру, они представляют самих себя. Любой объект, представляющий сам себя, называется **простым объектом**. Аналогично структура, состоящая из простых объектов, называется **простой структурой**.

Объект, представляющий другой объект или совокупность объектов, называется **составным объектом**, а использующие такие составные объекты структуры - **составными структурами.**

Возьмем утверждение

```
коллекция(“Иванов”,книга(“Использование Турбо-Пролога”,
                ”Ин,Соломон”,издание(“Мир”,1993))).
```

Предикат **коллекция** содержит составной объект **книга**, который в свою очередь содержит составной объект **издание**.

Описание указанных структур в программе может быть таким:

```
domains
   личная_библиотека=книга(название,автор,издание)
   издание=издание(издательство,год)
   владелец,название,автор,издательство=symbol
   год=integer
predicates
   коллекция(владелец,личная_библиотека)
clauses
   коллекция(“Иванов”,книга(“ИспользованиеТурбо-
                Пролога”,”Ин,Соломон”,издание(“Мир”,1993))).
```

Представление базы данных в структурированном виде удобно для извлечения из нее необходимой информации. Здесь можно ссылаться на объекты, не указывая в деталях всех их компонент. Можно задавать только структуру интересующих нас объектов и оставлять конкретные компоненты без точного описания или лишь с частичным описанием.

Примеры возможных запросов к приведенной программе:

Какие книги есть в коллекции Иванова?

коллекция(“Иванов”,X).

Кто автор книги “Использование Турбо-Пролога”?

коллекция(_,книга(“Использование Турбо-Пролога”,X,_)).

Можно создать набор правил, которые делали бы взаимодействие с базой данных более удобным, например:

```
книга(Название,Автор,Издание):-
                коллекция(_,книга(Название,Автор,Издание)).
издание(Издательство,Год):-книга(_,_,издание(Издательство, Год)).
год_издания(Год):-издание(_,Год).
```

Эти правила можно использовать, например, в следующих запросах:

Кто автор книги “Использование Турбо-Пролога”?

книга(“Использование Турбо-Пролога”,X,_).

Есть ли в коллекции Иванова книги 1990 г. издания?

коллекция("Иванов",X),издание(_,1990).

Примечание: соответствующие предикаты для созданных правил должны быть определены в разделе predicates.

3. **Содержание задания по лабораторной работе**.

Сформировать базу данных о семьях (можно использовать файл semja.pro, приведенный в приложении). Каждая семья описывается одним предложением. Информация о семье представлена в виде структуры. Каждая семья состоит из трех компонент: мужа, жены и детей. Детей представить в виде списка, состоящего из произвольного числа элементов. Каждого члена семьи представить структурой, состоящей из четырех компонент: имени, фамилии, даты рождения, работы. Информация о работе - указание должности и оклада. Пример одного из предложений базы данных:

```
семья(член_семьи(“Николай”,”Иванов”,дата(12,май,1948), работа(инженер,210)),
      член_семьи(“Анна”,”Иванова”,дата(5,январь,1952), работа(врач,190)),
   [член_семьи(“Инна”,”Иванова”,дата(20,март,1971), работа(студент,45)),
      член_семьи(“Олег”,”Иванов”,дата(25,июнь,1978),работа(ученик,0))]).
```

Для упрощения запросов можно сформировать правила для следующих предикатов: муж, жена, дети, член_семьи, ребенок, дата_рождения, работа (примеры некоторых правил есть в файле semja.pro).

Из базы данных (файл **семья.pro**) получить следующую информацию:

1. Определить имена всех членов семьи Ивановых.
2. Найти всех жен, родившихся в январе.
3. Найти всех детей моложе 15 лет.
4. Найти фамилии семей, имеющих не менее двух детей.
5. Найти семьи, в которых жена не носит фамилию мужа.
6. Найти семьи, в которых жена не работает.
7. Найти семьи без отца.
8. В какой семье есть дети-близнецы?
9. Найти людей, родившихся до 1950 г. и имеющих оклад менее 150.
10. Найти семьи без детей.
11. Найти семьи, где жена работает, а муж не работает.

12. Найти детей, разница в возрасте родителей которых составляет не менее 15 лет.

13. Найти общий доход семьи Ивановых.

4. **Содержание отчета**: цель работы, постановка задачи, текст программы, результаты работы программы, выводы.

Контрольные вопросы:

1. Что относится к простым данным ?
2. Что называется простым объектом, простой структурой ?
3. Что такое составной объект и составная структура ?
4. Функтор составного объекта.
5. Могут ли аргументы составного объекта быть также составными объектами ?
6. Объявление составных доменов в разделе программы domains.
7. Сопоставление составных объектов.
8. Могут ли быть элементы списка составными объектами, как такой список объявляется в разделе domains ?
9. Могут ли аргументами составного объекта быть списки ?
10. Как задать цель в разделе goal, чтобы просматривалась вся база данных, состоящая из составных объектов, и в конце просмотра выводилось сообщение Yes ?
11. Объявление альтернативных доменов.

ЛАБОРАТОРНАЯ РАБОТА 9.

РАБОТА С ФАЙЛАМИ

1. **Цель работы**: приобретение практических навыков в составлении и отладке Пролог-программ, использующих файлы.

2. **Краткие справочные данные**.

При использовании в программе файлов ее необходимо снабдить описанием файлового домена, которое выглядит так:

file=datafile,

где file - стандартный тип домена (файловый), datafile - логическое имя файла.

В файловом описании можно указывать несколько логических имен, но само описание должно быть единственным, например,

file=datafile1;datafile2;datafile3

При работе с файлами могут использоваться следующие встроенные предикаты Турбо-Пролога:

openread(СимволичИмяФайла,ИмяФайлаОС) (file,string) - (i,i) – открытие файла для чтения;

openwrite(СимволичИмяФайла, ИмяФайлаОС) (file,string) - (i,i) – открытие файла для записи;

openappend(СимволичИмяФайла, ИмяФайлаОС) (file,string) - (i,i) – открытие файла для дозаписи в конец существующего файла;

openmodify(СимволичИмяФайла, ИмяФайлаОС) (file,string) - (i,i) – открытие файла для модификации (чтения/записи);

readdevice(СимволичИмяФайла) (file) - (i) (o) – переадресация устройства чтения;

writedevice(СимволичИмяФайла) (file) - (i) (o) – переадресация устройства записи;

closefile(СимволичИмяФайла) (file) - (i) – закрытие файла;

filepos(СимволичИмяФайла,Позиция,Режим) (file,real,integer) - (i,i,i) (i,o,i) – установка или определение позиции указателя файла; Режим: 0 – относительно начала файла, 1– относительно текущей позиции файла, 2 – относительно конца файла; для шаблона (i,o,i) Режим=0;

eof(СимволичИмяФайла) (file) - (i) – проверка на конец файла;

flush(СимволичИмяФайла) (file) - (i) – сброс данных из внутреннего файлового буфера в заданный файл;
existfile(ИмяФайлаОС) (string) - (i) – проверка наличия файла;
filemode(СимволичИмяФайла,Режим)(file,integer) - (i,i) – установка указанного файла в текстовый или двоичный режим: Режим=0 – двоичный, Режим=1 –текстовй;
deletefile(ИмяФайлаОС) (string) - (i) – удаление файла ;
renamefile(СтароеИмяФайлаОС, НовоеИмяФайлаДОС) (string,string) - (i,i) – переименование файла;
disk(ПутьОС) (string) - (i) (o) – установление пути к файлу;
dir(ПутьОС,ШаблонФайла,ИмяФайлаОС) (string,string,string) - (i,i,o) – выбор файла из указанного директория;
file_str(ИмяФайлаОС,Строка) (string,string) – (i,o) (i,i) – запись содержимого текстового файла в строку или наоборот.

Например, при записи данных в файл последовательность действий следующая:

- открытие файла с помощью предиката openwrite;
- назначение файла в качестве устройства записи предикатом writedevice;
- сама запись в файл, например, с помощью предикатов write или writef;
- использование любых других предикатов и правил, отвечающих назначению программы;
- закрытие файла предикатом closefile.

Все эти действия в программе могут быть описаны следующим образом:

```
openwrite(datafile1,"File1.dat"),
writedevice(datafile1),
< запись в файл >
< любые другие правила или предикаты >
closefile(datafile1).
```

По такой же схеме могут быть описаны действия чтения данных из файла и дозаписи данных в файл.

Во встроенных предикатах для работы с файлами могут использоваться логические имена стандартных устройств компьютера: **keyboard** (клавиатура), **screen** (экран дисплея).

Программа, осуществляющая считывание данных с клавиатуры и запись их в файл, может выглядеть так:

```
domains
  file=datafile
  kstr,fstr=string
predicates
  readin(kstr,fstr) /* чтение-запись */
  create_a_file     /* создание файла */
goal
  create_a_file.
clauses
  create_a_file:-
    nl,nl,write("Введите имя файла"),nl,nl,
    readln(Filename),
    openwrite(datafile,Filename),
      writedevice(datafile),
      readln(Kstr),/* ввод первой строки с клавиатуры */
      concat(Kstr,"\13\10",Fstr),
      readin(Kstr,Fstr),
      closefile(datafile).
  /* Ввод последующих строк и запись их в файл */
  readin("stop",_):-!.
```

```
readin(_,Fstr):- write(Fstr),readln(Kstr1),
                 concat(Kstr1,"\13\10",Fstr1),
                 readin(Kstr1,Fstr1).
```

В приведенной программе

Kstr - строка, вводимая с клавиатуры,

Fstr - строка, выводимая в файл.

Содержимым файла Filename будут дополненные символами "\13\10" вводимые пользователем строки. Дополнение строки символами "\13\10" необходимо для того, чтобы предикат readln мог различать строки при считывании их из файла в дальнейшем. Работа программы заканчивается, если введена строка "stop". Вместо символов "\13\10" можно использовать символ "\n".

3. **Содержание задания по лабораторной работе**.

Из входной строки, содержащей последовательность слов (например, женские имена), разделенных пробелами, получить предложения с заданным именем функтора (например, женщина), аргументами которого являются слова входной строки. Каждое предложение должно располагаться в отдельной строке и заканчиваться точкой. Предложения записать в файл, а затем из файла вывести на экран дисплея.

4. **Задачи по работе с файлами**.

4.1. Разделение текстового файла на два примерно одинаковой длины по границе строки.
4.2. Объединение трех текстовых файлов в один.
4.3. Нахождение в текстовом файле двух самых коротких строк.
4.4. Нахождение в текстовом файле трех самых длинных строк.
4.5. Определение средней длины строк текстового файла.
4.6. Извлечение из текстового файла строк, длина которых меньше заданной.
4.7. Извлечение из текстового файла строк, длина которых больше заданной.
4.8. Сортировка строк текстового файла в порядке уменьшения их длины.
4.9. Сортировка строк текстового файла в порядке увеличения их длины.
4.10. Извлечение из текстового файла строк, в которых есть числовые данные.
4.11. Подсчет в текстовом файле количества строк, длина которых меньше заданной.
4.12. Подсчет в текстовом файле количества строк, длина которых больше заданной.
4.13. Подсчет количества цифровых символов в текстовом файле.
4.14. Удаление из файла, содержащего текст Пролог-программы, комментариев.
4.15. Удаление из текстового файла строк, в которых нет заглавных букв.
4.16. Дублирование каждой второй строки текстового файла.
4.17. Усечение всех строк текстового файла до указанной длины.
4.18. Извлечение из текстового файла строк, закачивающихся точкой.

5. **Содержание отчета**: цель работы, постановка задачи, текст программы, результаты работы программы, выводы.

Контрольные вопросы:

1. Объявление файловых доменов.
2. Какие из устройств являются стандартными устройствами ввода и вывода по умолчанию.
3. Встроенные логические имена стандартных устройств ввода-вывода.
4. Что такое логическое (символическое) имя файла и где оно объявляется ?
5. Может ли быть записано логическое имя файла в программе с прописной буквы ?
6. Сохраняется ли содержимое файла, открытого для записи предикатом openwrite ?
7. Как можно дополнить содержимое файла новыми данными ?

8. Какой предикат открывает файл одновременно на чтение и запись ?
9. Что произойдет, если файл, открытый для записи, не закрыть предикатом closefile ?
10. Какие предикаты служат для переадресации устройств чтения-записи ?
11. Как можно записать содержимое всего текстового файла в строку ?
12. Нужно ли открывать и закрывать файл при использовании предиката file_str?
13. Как наиболее просто вывести содержимое всего текстового файла на экран ?
14. Что используется в качестве аргументов в предикатах eof и existfile ?
15. Какой предикат служит для чтения строки из файла ?
16. Можно ли использовать предикат filemode для установления режима работы с файлом (бинарный или текстовый) ?
17. Назначение третьего аргумента предиката filepos.

ЛАБОРАТОРНАЯ РАБОТА 10.

ПОСТРОЕНИЕ МЕНЮ

1. **Цель работы**: умение создавать в программе удобный для пользователя интерфейс.

2. **Краткие справочные данные**.

На рис. 2 дается структурная диаграмма программы, использующей упрощенное меню.

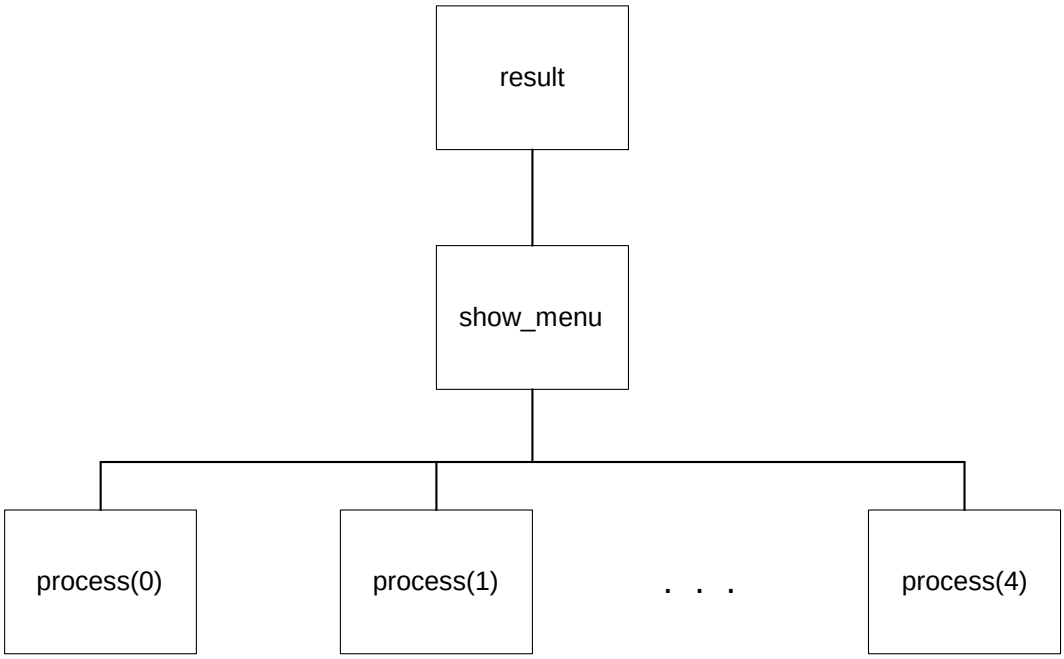


Рис. 2. Структурная диаграмма программы

Целевое утверждение result создает главное меню show_menu. В главном меню можно выбрать одну из пяти опций, каждая из которых реализует определенную операцию (process).

Возможный вариант программы:

```
predicates

  nondeterm repeat

  process(integer)

  show_menu

  result
```

```
goal
  result.
clauses
  result:-
    show_menu,nl,write("Нажмите любую клавишу"),
    readchar(_),exit.
  repeat.
  repeat:-repeat.
  show_menu:-
    repeat,
    write("************************************************"),nl,
    write("   0   Выход"),nl,
    write("   1   Создание файла"),nl,
    write("************************************************"),nl,nl,
/* аналогичные предложения для опций 2-4 */
    write("Наберите одну из цифр (0-4): "),
    readint(N),N<5,process(N),N=0,!.
  process(0):-
    nl,write("    Выход из меню"),nl.
  process(1):-
/* реализация процесса  создания файла */
    write("   Для возврата в главное меню нажмите любую клавишу"),nl,nl,
    readchar(_).
 /* аналогично пишутся остальные правила process */
```

Правило повтора **repeat**, являясь одной из компонент некоторого правила, обеспечивает циклическое выполнение расположенных после repeat подцелей данного правила.

3. **Содержание задания по лабораторной работе**.

Создать программу, использующую меню, в которой предусмотрено выполнение следующих действий: создание файла, дополнение имеющегося файла, редактирование данных файла, просмотр данных файла, удаление файла.

4. **Содержание отчета**: цель работы, постановка задачи, изображение структурной диаграммы программы, текст программы, результаты работы программы, выводы.

Контрольные вопросы:

1. Назначение и механизм работы правила repeat.
2. Можно ли вместо repeat использовать в правиле повтора другое имя ?
3. Что отображается на структурной диаграмме программы ?
4. Как можно расширить структурную диаграмму, приведенную в данной работе ?
5. Можно ли отождествить структурную диаграмму со схемой алгоритма программы ?

6. Как можно изменить приведенный вариант программы, чтобы при реализации каждого пункта меню не нужно было вводить имя файла ?
7. Как будет работать приведенный вариант программы, если при вводе номера пункта меню набрать, например, цифру 7?
8. Что произойдет, если в конце правила process убрать подцель readchar(_)?
9. Объяснить, каким образом и в каких случаях после выполнения отдельного пункта меню производится опять возврат в главное меню ?

ЛАБОРАТОРНАЯ РАБОТА 11.

ДИНАМИЧЕСКИЕ БАЗЫ ДАННЫХ

1. **Цель работы**: приобретение практических навыков составления и отладки программ, работающих с динамической базой данных.

2. **Краткие справочные данные**.

В Visual Prolog имеются специальные средства для организации реляционных баз данных (БД). Предикаты базы данных описываются в разделе **facts**, который должен располагаться в программе перед разделом predicates. Все утверждения с предикатами, описанными в разделе facts, составляют динамическую БД, которая, в отличие от неизменяемой статической БД, являющейся частью кода программы, в процессе работы программы может меняться, т.е. из нее можно удалять любые содержащиеся в ней утверждения, а также добавлять новые.

Динамическая база данных располагается в оперативной памяти (резидентная БД).

Например, в динамической БД содержатся сведения об игроках-футболистах с предикатом dplayer, тогда требуется следующее описание:

```
database
   dplayer(name,team,number),
```

где name, team, number - типы данных, описанные в domains.

Для работы с динамической БД могут использоваться встроенные предикаты:

asserta(<факт>) (dbasedom) : (i) - занесение нового факта в начало динамической БД;

assertz(<факт>) (dbasedom) : (i) - занесение нового факта в конец динамической БД;

retract(<факт>) (dbasedom) : (i) - - удаление утверждения (одного) из динамической БД;

retractall(<факт>) (dbasedom) : (i) - - удаление всех утверждений для указанного предиката (параметр <факт>) из динамической БД.

Домен динамической базы данных dbasedom объявляется автоматически для каждого предиката из раздела facts, поэтому описывать его в разделе domains не нужно.

save(ИмяФайла) (string) : (i) - запись на диск динамической БД;

consult(ИмяФайла) (string) : (i) - добавление текстового файла, сохраненного ранее предикатом save, к динамической базе данных.

findall(Переменная,Терм,Список) : (i,i,o) – сбор из БД объектов Переменная в список Список. Параметр Переменная должна являться одним из аргументов предиката Терм. Список должен быть описан в разделе domains. Например, подцель findall(Name,dplayer(Name,_,_),L) сформирует список L фамилий всех футболистов, описанных в динамической базе данных предикатом dplayer.

Иногда бывает предпочтительно иметь часть информации БД в виде утверждений статической БД. Сразу после активизации программы эти данные заносятся в динамическую БД (с помощью предикатов asserta и assertz). В общем предикаты статической БД имеют другое имя, но ту же форму представления данных, что и предикаты динамической БД.

Предикат статической БД, соответствующий предикату dplayer динамической БД, есть

```
predicates
  player(name,team,number)
clauses
  player("Адамов","Динамо",4).
  player("Сидоров","Торпедо",8).
```

Правилом для занесения в динамическую БД информации из статической БД является:

```
assert_database:-player(Name,Team,Number),
        assertz(dplayer(Name,Team,Number)),
            fail.
assert_database:-!.
```

Используемый в этом правиле предикат fail позволяет перебрать все утверждения предиката player.

Подобным образом можно записать правило для очистки динамической БД:

```
clear_database:-
   retract(dplayer(_,_,_)),fail.
clear_database:-!.
```

Вместо обращения к этому правилу можно записать подцель retractall(_).

3. **Содержание задания по лабораторной работе**.

Разработать самообучающийся англо-русский словарь, сущность которого в следующем. В программе имеется некоторый набор пар английских и соответствующих им русских слов. По введенному слову мы должны получить его перевод (с английского на русский или наоборот). Если нужного слова в словаре нет, должна быть предусмотрена возможность пополнения словаря этим словом с его переводом. В конце работы программы дополненная БД должна сохраняться в файле, а при новом запуске программы информация из файла заносится в динамическую БД, которая может опять пополняться. В программе предусмотреть возможность редактирования и удаления отдельных предложений БД. Выбор отдельных опций должен производиться из меню.

Дополнительные задачи к работе:

1. Выбрать из словаря русские слова (вывод на экран по одному в строке), в которых есть заданная буква.
2. Вывести на экран всю динамическую базу фактов, отсортированную в алфавитном порядке по русским словам.
3. Выбрать из словаря английские слова (вывод на экран по одному в строке), начинающиеся с заданной буквы.
4. Выбрать из словаря русские слова (вывод на экран по одному в строке), заканчивающиеся на заданную букву.
5. Выбрать из словаря английские слова (вывод на экран по одному в строке), длина которых меньше заданной.
6. Выбрать из словаря те пары слов (вывод каждой пары слов в отдельной строке), в которых длина русского слова меньше длины английского слова.
7. Выбрать из словаря и русские и английские слова (вывод на экран по одному слову в строке), в которых первая и последняя буквы одинаковые.
8. Выбрать из словаря и русские и английские слова (вывод на экран по одному слову в строке), в которых есть хотя бы одна повторяющаяся буква.

4. **Содержание отчета**: цель работы, постановка задачи, структурная диаграмма программы, текст программы, результаты работы, выводы.

Контрольные вопросы:

1. Где располагается динамическая база данных ?
2. Отличие статической базы данных от динамической базы данных.
3. Как можно статическую базу данных переслать в динамическую базу данных ?
4. Где в программе объявляются предикаты статической и динамической баз данных ?
5. Чем отличаются факты статической и динамической баз данных для одного и того же объекта ?
6. Можно ли в разделе facts объявить несколько предикатов динамической базы данных ?
7. Может ли быть в программе несколько разделов facts ?
8. Могут ли факты динамической базы данных содержать переменные ?
9. Можно ли в динамической базе данных разместить правила ?
10. Как сохранить факты, размещенные в динамической базе данных, на диске ?
11. Как можно удалить из динамической базы данных сразу несколько фактов ?
12. Назначение стандартного предиката findall.
13. Способы занесения фактов в динамическую базу данных.
14. Можно ли изменить сохраненную в файл динамическую базу данных средстваит обычного текстового редактора ?
15. Требования к формату файла, в котором сохранена динамическая база данных.

ЛАБОРАТОРНАЯ РАБОТА 12.

ВНЕШНИЕ БАЗЫ ДАННЫХ

1. **Цель работы**: приобретение практических навыков составления и отладки программ, работающих с внешней БД.

2. **Краткие справочные данные**.

Visual Prolog наряду с внутренними базами данных (лаб. раб. 11) поддерживает внешние БД. Внешняя БД обычно создается в случае, если объем данных больше свободной части ОЗУ или предполагается значительное расширение БД. В отличие от внутренней БД во внешней БД термы могут храниться не только в виде структуры, похожей на факт, но и в виде списков, сложных структур, чисел и т.д.

Внешняя БД состоит из цепочек (chain), в которых хранятся термы. Количество термов в цепочке не ограничено, в БД может быть любое число цепочек. Каждая цепочка имеет свое имя (типа symbol) и может быть обработана как отдельная группа данных. Вся внешняя БД в целом также легко управляема как отдельный модуль.

Объявление внешней БД:

```
domains
  db_selector=<ИмяБД1>;<ИмяБД2>…
```

Предикаты для работы со всей внешней БД

a) Создание новой (пустой) БД
db_create (ИмяБД, ИмяФайла, Место)(db_selector, string, symbol):(i,i,i)

Например: db_create (db_name, “f1.dat”, in_file)
db_create (mydba, “f2.dat”, in_memory)

Третий параметр предиката определяет место расположения внешней БД: in_file (для внешнего файла), in_memory (для RAM), in_EMS (для EMS-памяти).

Внешняя база данных, созданная предикатом db_create, считается **открытой**.

b) Открытие ранее созданной БД
db_open (ИмяБД, ИмяФайла, Место) (db_selector, string, symbol):(i,i,i)

c) Копирование БД в другое место (напр., из файла в ОЗУ или наоборот)
db_copy (ИмяБД, ИмяФайла, Новое_место)(db_selector, string, symbol):(i,i,i)

d) Закрытие БД (в конце работы)
db_close (ИмяБД)(db_selector):(i)

e) Удаление БД
db_delete (ИмяФайла, Место)(string, symbol):(i,i)

f) Открытие БД, помеченной как ошибочная
db_openinvalid(ИмяБД, ИмяФайла, Место)(db_selector, string, symbol):(i,i,i)

g) Получение информации о БД
db_statistics(ИмяБД, Количество_термов, Объем_ОП_под_таблицами_БД, Объем_памяти_занятой_термами_и_указателями,Свободная_память)(db_selector,real, real,real,real):(:(i,o,o,o,o)

h) Определение имен цепочек во внешней БД
db_ chains(ИмяБД, Цепь)(db_selector, symbol):(i,o)

i) Повышение надежности БД на диске (таблицы БД записываются на диск)
db_flush(ИмяБД)(db_selector):(i)

j) Определение имен B+деревьев внешней БД
db_btrees(ИмяБД, Имя БДер)(db_selector, string):(i,o)

Предикаты для работы с цепочками

Цепочки термов запоминаются последовательно и одновременно доступна только одна цепочка. Каждая цепочка имеет свое имя, которое нигде специально не объявляется. Цепочка создается, если ее имя впервые встретилось в предикатах записи термов в БД.

Для записи термов во внешнюю БД служат следующие три предиката:

chain_inserta(ИмяБД, Цепь, Дом, Терм, Указ)(db_selector, symbol,<domain>,<term>, ref):(i,i,i,i,o)

chain_insertz(ИмяБД, Цепь, Дом, Терм, Указ)(db_selector, symbol,<domain>,<term>, ref):(i,i,i,i,o)

chain_insertafter(ИмяБД, Цепь, Дом, Указ, Терм, НовУказ))(db_selector, symbol, <domain>, ref, <term>, ref)::(i,i,i,i,i,o)

ИмяБД – имя, данное внешней БД в предикате db_create;

Цепь – имя цепочки (тип symbol);

Дом – тип (домен) вставляемого терма;

Терм – вставляемый терм;

Указ – указатель на вставляемый терм (тип ref), в третьем предикате – указатель терма, после которого вставляется Терм с указателем НовУказ.

Указатели относятся к специальному, определенному в системе типу данных ref.

Предикаты chain_inserta и chain_insertz аналогичны по функциям встроенным предикатам asserta и assertz. Предикат chain_insertafter служит для вставки терма между термами, уже занесенными в БД.

Удаление цепочки: chain_delete(ИмяБД, Цепь) (db_selector, symbol):(i,i)

Предикаты

chain_first(ИмяБД, Цепь, Указ_на_первый_терм)(db_selector, symbol, ref):(i,i,o)

chain_last(ИмяБД, Цепь, Ука_на_последний_терм)(db_selector, symbol, ref):(i,i,o)

возвращают указатель соответственно на первый и последний термы в заданной цепочке.

Извлечение термов из внешней БД осуществляется предикатом

chain_terms(ИмяБД, Цепь, Дом, Терм, Указ)(db_selector, symbol, <domain>, <term>, ref):(i,i,i,o,o) (i,i,i,i,o)

Параметры предиката такие же, как и у chain_inserta и chain_insertz. Отличие в том, что четвертый параметр может быть выходным. При шаблоне (i,i,i,i,o) параметр Терм может быть задан частично, т.е. будут извлекаться из связанного списка не все термы, а только с нужными свойствами.

Предикаты

chain_next(ИмяБД, Цепь, Указ_на_следующий_терм)(db_selector, symbol, ref):(i,i,o)

chain_prev(ИмяБД, Цепь, Указ_на_предыдущий_терм)(db_selector, symbol, ref): (i,i,o)

возвращают указатель соответственно на следующий и предыдущий термы относительно заданного.

Предикаты для работы с термами

Замена терма

term_replace(ИмяБД, Дом, Указ, Терм) (db_selector,<domain>,ref,<term>): (i,i,i,i)

Заменяет терм с указателем Указ новым термом Терм типа Дом.

Удаление терма

term_delete(ИмяяБД, Цепь, Указ) (db_selector, symbol, ref):(i,i,i)

Извлечение терма из внешней БД

ref_term(ИмяБД, Дом, Указ, Терм) (db_selector,<domain>,ref,<term>)::(i,i,i,o)

Пример программы, читающей термы из внешней БД:

```
domains
    db_selector = mydba          % объявление имени внешней БД
    dbdom = city(cityname, code);
            person(family, name, street, code)
    cityname, code, family, name, street = string
predicates
    rd( ref )        % извлечение термов из БД
clauses
    rd( Ref) :- ref_term( mydba, dbdom, Ref, Term),
                write(Term), nl, fail.
    rd( Ref) :- chain_next( mydba, Ref, Next), rd( Next).
    Rd(_).
goal
    db_open( mydba, “dd.dat”, in_file),
    db_chains( mydba, Chain),
    chain_first( mydba, Chain, Ref),
    rd( Ref ).
```

В приведенном примере извлечение термов из внешней БД происходит в пределах только одной цепочки.

Чтобы извлечь термы всей БД (из всех цепочек), нужно модифицировать правило rd(Ref), удалив из него последнее предложение rd(_). В этом случае после извлечения из первой цепочки всех термов происходит возврат к подцели db_chains раздела goal и осуществляется переход к следующей цепочке, из которой будут извлекаться все термы и т.д. до конца внешней БД.

3. Содержание задания по лабораторной работе.

Создать внешнюю базу данных на диске, имеющую два связанных списка (две цепочки chain) с именами, например, gr215 и gr315. Структура термов внешней БД:

student(Фамилия, Имя, Дата_рождения, Пол).

Извлечь из первой цепочки фамилии и имена студентов указанного пола и из обеих цепочек – фамилии и имена студентов, родившихся в заданном месяце.

В меню предусмотреть операции создания внешней БД, извлечения из БД нужной информации в соответствии с заданием, дополнения БД новыми термами, удаления термов из БД, просмотра содержимого отдельных цепочек внешней БД.

Дополнительные задачи к работе:

1. Вывести на экран всю внешнюю БД, отсортированную в алфавитном порядке по фамилиям.
2. Выбрать из всей БД фамилии (вывод на экран по одной в строке), в которых есть заданная буква (независимо от регистра).
3. Выбрать из всей БД фамилии (вывод на экран по одной в строке), заканчивающиеся на заданную букву.
4. Выбрать из всей БД фамилии (вывод на экран по одной в строке), начинающиеся с заданной буквы.
5. Выбрать из всей БД фамилии и имена (вывод каждой пары в отдельной строке), где длина фамилии меньше длины имени.
6. Выбрать из всей БД фамилии (вывод на экран по одной в строке), длина которых меньше заданной.
7. Выбрать из всей БД фамилии (вывод на экран по одной в строке), в которых есть хотя бы одна повторяющаяся буква (независимо от регистра).
8. Выбрать из всей БД фамилии (вывод на экран по одной в строке), в которых первая и последняя буквы одинаковые (независимо от регистра).

4. **Содержание отчета:**

Цель работы, постановка задачи, текст программы, результаты тестирования программы, выводы.

Контрольные вопросы:

1. Отличия внешней базы данных от динамической базы данных.
2. Какие объекты данных могут быть использованы в качестве термов в динамической и внешней базах данных ?
3. Где и как производится объявление внешней базы данных в программе ?
4. Где может располагаться внешняя база данных ?
5. Соглашения об именовании стандартных предикатов управления внешними базами данных.
6. Что такое цепочка внешней базы данных ?
7. Где впервые при работе с программой, использующей внешнюю базу данных, встречается имя цепочки ?
8. Что произойдет, если при вводе данных неправильно набрано имя цепочки ?
9. Что содержится во внешней базе данных, созданной предикатом db_create ?
10. Как можно определить имена цепочек внешней базы данных ?
11. Какие стандартные предикаты используются для занесения термов во внешнюю базу данных ?
12. Какие стандартные предикаты используются для движения по термам внешней базы данных ?
13. Какие стандартные предикаты используются для извлечения термов из внешней базы данных, чем они отличаются ?
14. Что такое указатель терма, его тип.
15. Объяснить, как с помощью правила rd(Ref), используемого в программе, приведенной в разделе 2, извлекаются термы из одной цепочки (из всех цепочек) ?
16. Как можно открыть внешнюю базу данных, помеченную как ошибочная (invalid) ?
17. Будет ли отличаться скопированная внешняя база данных с помощью предиката db_copy от оригинала ?

ЛАБОРАТОРНАЯ РАБОТА 13.

B+ДЕРЕВЬЯ

1. **Цель работы**: приобретение практических навыков составления и отладки программ, работающих с B+деревьями.

2. **Краткие справочные данные**.

B+дерево является структурой данных, которую можно применять для очень эффективного метода сортировки большого количества данных.

B+деревья находятся во внешней базе данных. Каждый вход в B+дерево – это пара величин: ключ (информационное поле записи, по которому построено B+дерево) и связанный с этим ключом указатель базы данных. При вводе новой записи во внешнюю БД Пролог включает ключ и указатель, соответствующие этой записи, в B+дерево.

B+дерево разбито на отдельные страницы или узлы. Каждый узел B+дерева является либо последним (лист), либо порождает два нижележащих узла. Каждый узел содержит группу ключей из заданного диапазона. По значению ключа можно быстро проверить, не находится ли искомый ключ внутри диапазона конкретного узла. Если да, то быстро находится указатель записи; если значение ключа меньше диапазона ключей узла, то поиск ведется по левой ветви B+дерева, если больше – по правой.

B+дерево **создается** с помощью предиката

bt_create(ИмяБД, ИмяB+дер, ПерB+дер, ДлКл, Пор)(db_selector, string, bt_selector, integer, integer): (i,i,o,i,i)

ИмяБД – имя внешней БД, в которой создается B+дерево;

ИмяB+дер – имя, которое дает B+дереву пользователь;

ПерB+дер – переключатель (ссылка), используемый другими предикатами при выполнении операций с B+деревом;

ДлКл – длина ключа, в пределах которой будет производиться упорядочивание ключей при построении дерева;

Пор – порядок дерева, определяющий, сколько ключей запоминается в каждом узле B+дерева. Для баз данных среднего размера рекомендуется Пор=4.

Для **открытия, закрытия и удаления** B+дерева используются предикаты:

bt_open(ИмяБД, ИмяB+дер, ПерB+дер)(db_selector,string,bt_selector):(i,i,o)

bt_close(ИмяБД, ПерB+дер)(db_selector, bt_selector):(i,i)

bt_delete(ИмяБД, ИмяB+дер)(db_selector,bt_selector):(i,i)

Ниже приведен пример правила, проверяющего, существует ли открываемое B+дерево; если его нет, то оно создается:

```
external_open(Bt_sel):-
   existfile("parts.dba"),!,
   db_open(parts, "parts.dba",in_file),
   bt_open(parts,"parts.tree",Bt_sel).
external_open(Bt_sel):-
   db_create(parts,"parts.dba",in_file),
   bt_create(parts,"parts.tree",Bt_sel,10,4).
```

Изменения в B+дереве относительно ключа осуществляются двумя предикатами:

1) key_insert(ИмяБД, ПерB+дер, Ключ, Ссылка)(db_selector, bt_selector, string, ref):(i,i,i,i)

- **вставляет** новый ключ (информационное поле) в B+дерево. Параметр Ключ является новым ключом, а Ссылка является номером ссылки БД, принадлежащим этому ключу.

Пример:

```
domains
  db_selector=mydba
  dbdom=person(family,name,sex)
```

```
  family, name=string
  sex=char
goal
  db_open(mydba,"dd.bin",in_file),
  bt_open(mydba,"personnames",Bt_sel),
  chain_inserta(mydba,namechain,,dbdom,person("Hoffman","Artur",'m',),Ref),
  key_insert(mydba,Bt_sel,"Artur",Ref),
  db_close(mydba).
```

2) key_delete(ИмяБД, ПерВ+дер, Ключ, Ссылка)(db_selector, bt_selector, string, ref):(i,i,i,i)
   - **удаляет** ключ из В+дерева.

Для осуществления **поиска** в В+дереве используются следующие предикаты:

key_search(ИмяБД, ПерВ+дер, Ключ, Ссылка)(db_selector,bt_selector,string,ref):(i,i,i,o)

- находит ключ в В+дереве. Если ключ найден, номер ссылки БД, принадлежащий этому ключу, будет возвращен через параметр Ссылка. Если ключ не найден, предикат будет несогласован, однако внутренний указатель В+дерева будет указывать на ключ, следующий непосредственно за тем, на который указатель указывал до поиска ключа.

key_current(ИмяБД, ПерВ+дер, Ключ, Ссылка)(db_selector,bt_selector,string,ref):(i,i,o,o)

- возвращает текущий ключ и номер ссылки БД для текущего внутреннего указателя В+дерева.

key_first(ИмяБД, ПерВ+дер, Ссылка) (db_selector, bt_selector, ref):(i,i,o)

key_last(ИмяБД, ПерВ+дер, Ссылка) (db_selector, bt_selector, ref):(i,i,o)

key_next(ИмяБД, ПерВ+дер, Ссылка) (db_selector, bt_selector, ref):(i,i,o)

key_prev(ИмяБД, ПерВ+дер, Ссылка) (db_selector, bt_selector, ref):(i,i,o)

Эти предикаты, имеющие одинаковый формат, перемещают внутренний указатель В+дерева соответственно на первый, последний, следующий и предыдущий ключи в В+дереве и возвращают номер ссылки БД, заданный для соответствующего ключа.

Пример:

/* Создание БД типа группа(Фамилия, Вес). Создание В+дерева по полю «Вес». Вывод фамилий членов группы в порядке убывания веса */

```
domains
    db_selector = mydba              % объявление имени внешней БД
    dbdom = gruppa(string,string)    % объявление домена терма
predicates
    insert_term(bt_selector)         % занесение термов в БД
     show_up(bt_selector)            % просмотр В+дерева в обратном порядке
clauses
    insert_term(Bt_sel1):-
       write("Фамилия:"),readln(Fam),Fam<>"aaa",
       write("Вес   :"),readln(Ves),
       chain_inserta(mydba,chain,dbdom,gruppa(Fam,Ves),Ref),
       key_insert(mydba,Bt_sel1,Ves,Ref),
       insert_term(Bt_sel1).
    insert_term(_):-!.
    show_up(Bt_sel):-
        key_current(mydba,Bt_sel,_,Ref),
        ref_term(mydba,dbdom,Ref,gruppa(Fam,Ves)),
        write(Ves,"  ",Fam),nl,fail.
     show_up(Bt_sel):-
        key_prev(mydba,Bt_sel,_),
        show_up(Bt_sel).
     show_up(_).
```

```
goal
    db_create(mydba,”gruppa.bin”,in_file),
    bt_create(mydba,”ves”,Bt_sel1,3,3),
    insert_term(Bt_sel1),
    key_last(mydba,Bt_sel1,_),
    show_up(Bt_sel1),
    db_close(mydba).
```

Замечания к программе:

а) Если упорядочивание производится по нескольким полям, то для каждого из этих полей создается свое двоичное дерево.

б) При вставке нового ключа в базу данных для одного из полей тип данных этого поля при ссылке на него в предикате key_insert должен быть string или приведен к string с помощью предикатов преобразования типов.

в) Если вывод по ключу производится в прямом порядке, то строится соответствующее правило (в goal используется предикат key_first вместо key_last, а в правиле show_up вместо key_prev используется key-next).

3. **Содержание задания по лабораторной работе.**

Создать внешнюю базу данных на диске следующего формата:

student(Номер_зачетки, Фамилия, Средний_балл).

В этой БД по каждому из полей сформировать B+дерево.

Требуется:

а) вывести фамилии студентов в порядке возрастания номеров зачетной книжки;

б) вывести фамилии студентов в алфавитном порядке;

в) вывести фамилии студентов в порядке убывания среднего балла.

В меню предусмотреть операции создания внешней БД, извлечения из БД нужной информации в соответствии с заданием, дополнения БД новыми термами, удаления термов из БД, просмотра содержимого отдельных B+деревьев.

Дополнительные задачи к работе:

1. Выбрать из базы данных фамилии (вывод на экран по одной в строке), в которых есть хотя бы одна повторяющаяся буква.
2. Выбрать из базы данных фамилии (вывод на экран по одной в строке), в которых первая и последняя буквы одинаковые.
3. Выбрать из базы данных фамилии (вывод каждой в отдельной строке), длина которых меньше длины номера зачетки.
4. Выбрать из базы данных фамилии (вывод на экран по одной в строке), длина которых меньше заданной.
5. Выбрать из базы данных фамилии (вывод на экран по одной в строке), заканчивающиеся на заданную букву.
6. Выбрать из базы данных фамилии (вывод на экран по одной в строке), начинающиеся с заданной буквы.
7. Выбрать из базы данных номера зачеток (вывод на экран по одному в строке), сумма цифровых символов которых меньше заданного числа.
8. Выбрать из базы данных фамилии (вывод на экран по одной в строке), в которых есть заданная буква.

4. **Содержание отчета:**

Цель работы, постановка задачи, текст программы, результаты тестирования программы, выводы.

Контрольные вопросы:

1. Что такое B+деревья, где и для чего они используются ?
2. Отличие B+дерева от двоичного (бинарного) дерева.
3. Механизм упорядочивания данных в двоичном дереве.

4. Что представляет собой ключ B+дерева ?
5. Страница, длина и порядок B+дерева.
6. Как в программе создается B+дерево ?
7. Какие стандартные предикаты используются для открытия, закрытия и удаления B+дерева ?
8. Как можно вставить новый ключ в B+дерево ?
9. Как можно удалить ключ из B+дерева ?
10. Какие стандартные предикаты используются для установки указателя ключа B+дерева на первый, последний, текущий ключ ?
11. Какие стандартные предикаты используются для перемещения указателя на ключ B+дерева вперед и назад ?
12. Какой тип данных должны иметь ключи B+дерева ?
13. Как можно определить имена B+деревьев внешней базы данных (см. лабораторную работу № 12) ?.
14. Как можно получить статистическую информацию о B+дереве ?
15. В каком порядке разместятся в B+дереве следующие значения ключа: 1, 2, 5, 12,43?

ЛАБОРАТОРНАЯ РАБОТА 14.

РЕКУРСИВНОЕ ОБЪЯВЛЕНИЕ ДОМЕНОВ

1. **Цель работы**: приобретение практических навыков составления и отладки программ, использующих рекурсию при объявлении доменов.

2. **Краткие справочные данные.**

При объявлении доменов Visual Prolog позволяет использовать рекурсию. Это бывает полезным при формализации знаний о некоторых объектах. В такого рода объявлениях, как правило, используются альтернативные домены, представленные в виде структур.

В качестве примера рассмотрим, каким образом формализуются знания при расчете сопротивления последовательно-параллельной электрической схемы, приведенной на рис. 3.

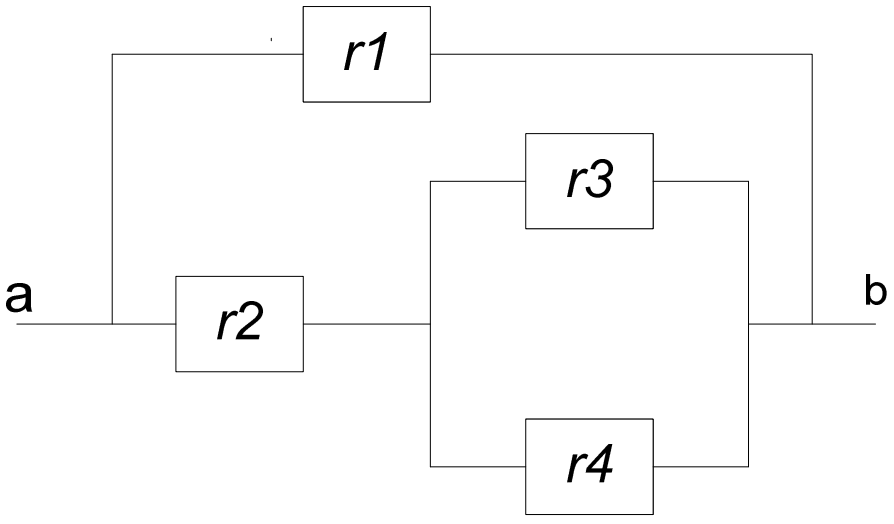


Рис. 3. Расчетная электрическая схема

Введем домен, определяющий представление схемы: отдельным резистором, последовательным соединением резисторов, параллельным соединением резисторов.

```
typ=res(symbol);
    posl(typ,typ);
    par(typ,typ)
```

В структуре res(symbol) параметр в скобках определяет символическое имя отдельного резистора или некоторой схемы из резисторов.

Назначение вводимых предикатов:

resist(симв_имя, Сопр) – описание резистора, обозначенного на схеме именем **симв_имя** и имеющего сопротивление **Сопр** (или схемы с именем симв_имя и сопротивлением Сопр);

schema(симв_имя, опис_схемы) – описание схемы соединения резисторов; первый параметр – символическое имя схемы, второй – описание последовательности соединения элементов схемы;

sopr_el(опис_схемы, Сопр) – определение сопротивления **Сопр** отдельного варианта схемы **опис_схемы**: резистора (или схемы), последовательной ветви, параллельной ветви;

result(симв_имя, Сопр) – целевое утверждение для определения общего сопротивления **Сопр** рассчитываемой схемы с именем **симв_имя**.

Текст Пролог-программы

```
domains
   typ=res(symbol);
        posl(typ,typ);
        par(typ,typ)
predicates
   resist(symbol, real)
   schema(symbol, typ)
   sopr_el(typ, real)
   result(symbol, real)
goal
   result(ab, R).
clauses
   resist(r1,3).
   resist(r2,5).
   resist(r3,4).
   resist(r4,8).
   schema(ab, par(res(r1),posl(res(r2),par(res(r3),res(r4))))).
   sopr_el(res(E),R):-resist(E,R).
   sopr_el(posl(C1,C2),R):- sopr_el(C1,R1), sopr_el(C2,R2),R=R1+R2.
   sopr_el(par(C1,C2),R):- sopr_el(C1,R1), sopr_el(C2,R2),R=(R1*R2)/(R1+R2).
   result(N,R):-schema(N,C),sopr_el(C,R).
```

3. Содержание задания по лабораторной работе.

Рассчитать общее сопротивление схемы, предложенной преподавателем.

4. Содержание отчета:

Цель работы, постановка задачи, текст программы, результаты тестирования программы, выводы.

Контрольные вопросы:

1. Почему домен typ, объявленный в программе раздела 2, является рекурсивным ?
2. Что означают символы «;» в объявлении домена typ ?
3. Объяснить, как производится описание электрической схемы в программе ?
4. Можно ли записать в разделе goal целевое утверждение следующим образом: result(AB, R) ?
5. Какие изменения нужно произвести, чтобы в результате выполнения программы определялось общее сопротивление параллельной ветви, составленной из резисторов r3 и r4 ?
6. Как можно описать схему, представленную тремя параллельными ветвями ?

7. При работе с какими другими объектами, кроме электрической схемы, могут использоваться рекурсивные домены ?

ЛАБОРАТОРНАЯ РАБОТА 15.

ОСНОВЫ РАБОТЫ В СРЕДЕ ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ

VISUAL PROLOG

1. Цель работы: приобретение начальных навыков создания графического интерфейса пользователя средствами визуальной среды разработки Visual Prolog.

2. Краткие справочные данные.

2.1. Создание нового приложения.

При помощи пункта меню **Project | New Project** вызывается эксперт приложений **Application Expert**.

В поле **Project Name** задается имя проекта, которое экспертом приложений используется для автоматической генерации файла проекта с расширением **vpr** и файла с расширением **prj** (если отмечен флажок **Multiprogrammer Mode**, разрешающий работу нескольких программистов над проектом).

В поле **Base Directory** указывается каталог, в котором будет находиться проект. При помощи кнопки **Browse** можно открыть браузер каталогов и указать тот, который необходим. Используя поле **Subdirectory**, можно создать для проекта свой подкаталог в одном из родительских каталогов.

На вкладке **Target** определяются значения четырех параметров: платформа (**DOS, DOS Extended, Windows16, Windows32**), пользовательский интерфейс (**VPI, EasyWin, WINBIND**, **Textmode, Other**), тип целевого объекта (**exe, dll**), основная платформа (**Prolog, MSVC++32bit**) (подчеркнуты установки по умолчанию, для нашего проекта их оставить без изменения).

На вкладке **VPI Options** можно определить, какие **VPI**-пакеты будет использовать проект (оставляем отмеченные установки по умолчанию).

После выполнения необходимых установок нажимается кнопка **Create** для генерации всех необходимых файлов и создания приложения по умолчанию.

В окне проекта среды визуальной разработки сейчас должны быть два исходных модуля: **<имя_проекта>.pro** и **VPITools.pro**.

После того, как проект создан, команда меню **Project | Run** откомпилирует и выполнит его. Должно появиться небольшое приложение.

2.2. Изменение меню.

В окне проекта нажимам кнопку **Menu** на левой панели инструментов. В центральной части окна проекта появится список меню, зарегистрированных в проекте. Дважды щелкаем мышью на пункте **Task Menu** (или нажимаем кнопку **Edit**) для активизации редактора меню. В окне редактора выбираем пункт меню, после которого будет добавляться новый пункт меню верхнего уровня, и нажимаем кнопку **New**. В появившемся окне **Menu Item Attributes** в поле **Text** набираем имя нового пункта меню. Имя-константу для пункта меню вводить не нужно: он автоматически получит константу с префиксом **id_** по умолчанию.

Перед именем пункта меню может быть символ **&**, который определяет, что при отображении меню символ, следующий за **&**, будет подчеркнут. Это используется для определения “горячих” клавиш. “Горячую” клавишу можно определить, напечатав символ в поле **Accelerator** и выбрав соответствующую комбинацию клавиш **Shift, Alt** или **Ctrl**.

Добавление пункта меню заканчивается нажатием кнопки **OK.**

С помощью редактора меню можно создать подменю для выбранного пункта меню путем нажатия кнопок **Submenu** и **New**.

Теперь можно протестировать меню. Для этого нажимается кнопка **Back**, пока она не станет неактивной. Затем выбирается режим **Test** для предварительного просмотра меню.

В режиме **Test** тестируемое меню появляется вместо меню задачи (т.е. взамен обычного меню) среды визуальной разработки; можно открыть и просмотреть его подменю.

Для выхода из режима **Test** нужно щелкнуть мышью в любом месте окна среды визуальной разработки за пределами меню или снова нажать кнопку **Test**.

Закрывается редактор меню при помощи кнопки **Close**.

2.3. Создание нового модуля.

В Visual Prolog возможно обрабатывать множество окон и диалоговых окон в одном и том же исходном модуле. Но часто код Пролога для нового окна помещается в отдельный (новый) модуль. Как организовать проект – это решение программиста.

Для создания нового модуля в окне проекта нажимается кнопка **Module** (на левой стороне), а затем кнопка **New**, в результате чего появится окно **Add file**. После набора в этом окне (в поле **Имя файла**) имени нового модуля нажимается кнопка **Open(Открыть)**, появится диалоговое окно **File Inclusions for Module**, в котором можно определить, какие файлы нужно генерировать для модуля. Далее нажимается кнопка **OK**. Будет сгенерирован новый файл для модуля с расширением **pro** и помещен в каталог проекта.

2.4. Создание окна.

Создание нового окна начинается с регистрации его в проекте. Нажимается кнопка **Window** на левой стороне окна проекта и затем нажимается кнопка **New** (наверху справа).

Появится диалоговое окно **Window Attributes**. В этом окне нужно набрать имя создаваемого окна в поле **Name** и нажать кнопку **OK.** Когда диалоговое окно **Window Attributes** закроется, автоматически появится редактор окон, который используется для размещения средств управления в окнах или изменения размера окна и атрибутов.

Далее для окна нужно сгенерировать его стандартный исходный код, генерируемый по умолчанию. Это делается с помощью эксперта окон и диалоговых окон, который активизируется кнопкой **Code Expert**, когда имя создаваемого окна выбрано в окне проекта.

После активизации эксперта окон и диалоговых окон в поле **Module** нужно раскрыть список и выбрать исходный модуль, в котором будет размещен код для окна. В окне **Event Type** выделяем строку **Window**, а в окне **Event or Item – e_Create**. Затем нажимается кнопка **Default Code**. Когда заданный по умолчанию код будет сгенерирован, станет доступной кнопка **Edit Code**. Нажатие кнопки **Edit Code** вызовет редактор, позиционированный на сгенерированном коде.

2.5. Активизация окна.

Теперь, когда мы закончили создавать окно, необходимо активизировать его, например, через один из пунктов меню. Выполняемые функции для выбранного пункта меню могут быть добавлены следующим образом.

Выбираем кнопку **Window** на левой стороне окна проекта, отмечаем в списке окон **Task Window** и нажимаем кнопку **Code Expert.** Когда появится эксперт окон и диалоговых окон, в списке **Event Type** выбираем строку **Menu**, в списке **Event or Item** выбираем идентификатор пункта меню и нажимаем кнопку **Add Clause**. Далее нажимаем кнопку **Edit Clause**, чтобы войти в редактор к добавленному предложению к обработчику события основного окна (**Task**) приложения. В начале второй строки добавленного предложения (после первого отсечения **!**) щелкаем правой кнопкой мыши и выбираем команду **Insert | Predicate Call | Window, Dialog or Toolbar**. В появившемся диалоговом окне выбираем переключатель **User Defined Window**. После нажатия кнопки **OK** будет вставлен вызов предиката для создания окна.

Если теперь запустить приложение, то из нашего пункта меню будет активизироваться созданное ранее окно.

2.6. Создание диалогового окна.

2.6.1. Создание стандартного диалогового окна.

Стандартное диалоговое окно (dlg_Ask, dlg_Error, dlg_Note и др.) может быть вызвано в окно приложения одним из пунктов меню (подменю) способом, описанным в п. 2.5 «Активизация окна», при этом после команды **Insert | Predicate Call | Window, Dialog or Toolbar** в появившемся диалоговом окне **Insert Call** выбирается команда **Common Dialog.**

2.6.2. Создание диалогового окна, определенного пользователем.

Диалоговое окно создается нажатием кнопок **Dialog** и **New** в окне проекта. Дальнейшие действия выполняются в соответствии с п. 2.4 «Создание окна». Активизация диалогового окна производится так же, как и активизация окна (см. п. 2.5).

Для упрощения работы с элементами управления, размещаемыми в диалоговом окне, можно использовать эксперт пакета диалоговых окон – инструмент, который отражает все опции, доступные в диалоговом пакете. Пакет диалоговых окон упрощает инициализацию и получение значений для диалоговых окон и имеет множество вариантов обработки и проверки достоверности значений элементов управления.

Эксперт пакета диалоговых окон вызывается из эксперта окон и диалоговых окон при помощи кнопки **Dialog Pack**.

Экран эксперта диалогового пакета разделен на две части. Левая область окна всегда содержит список доступных элементов управления. Некоторые из них снабжены префиксом **+.** Двойной щелчок на + отображает более детальную информацию. Правая сторона используется для отображения и изменения различных настроек выбранного элемента управления.

3. Содержание задания по лабораторной работе.

Создать приложение с именем, составленным из номера группы и фамилии одного из членов бригады (например, **107215_Kotov**). После пункта меню **Edit** родительского окна проекта вставить пункт меню **New**, в котором имеется три подменю: **Submenu1, Submenu2** и **Submenu3. Submenu1** вызывает общее диалоговое окно **dlg_Note**, имеющее заголовок **“Window1”** и текст внутри окна **“Hello World”. Submenu2** вызывает окно **Window2**, внутри которого имеется текст с фамилиями членов бригады, обрамленными рамкой. **Submenu3** вызывает диалоговое окно с элементами управления. Коды для окон **Window2** и **Window3** генерируются в модуле **Lab15.pro**.

Примерный порядок выполнения задания:

1. Создать новое приложение (п. 2.1).
2. Создать новый пункт меню New с тремя подменю (п. 2.2).
3. Создать модуль Lab15.pro (п. 2.3).
4. Подменю Submenu1 связать с общим диалоговым окном dlg_Note (п. 2.6.1).
5. Создать окно Window2 и связать его с подменю Submenu2 (пп. 2.4, 2.5).
6. Создать диалоговое окно Window3 и связать его с подменю Submenu3 (пп. 2.6.2, 2.5).

4. **Содержание отчета**: цель работы, постановка задачи, тексты исходных модулей программы, результаты работы программы, выводы.

Контрольные вопросы:

1. Чем отличается процесс создания нового проекта в среде визуального программирования Visual Prolog от аналогичного процесса в режиме Test Goal ?
2. Одинаково ли производится запуск на выполнение проекта в среде визуального программирования и в режиме Test Goal ?
3. Когда появляется и что представляет собой окно проекта ?
4. Как можно узнать имена всех исходных модулей проекта ?
5. Порядок создания нового модуля проекта.
6. Как можно открыть отдельный текстовый модуль проекта ?

7. Для чего предназначен браузер кода (Code Browser) ?
8. Что понимается под термином «ресурсы» проекта ?
9. Как можно узнать имя (идентификатор) ресурса проекта ?
10. Что представляет собой окно Task ?
11. Отличие окна от диалогового окна.
12. Модульные и немодульные диалоговые окна.
13. Как можно отредактировать отдельное диалоговое окно проекта ?
14. Назначение эксперта окон и диалоговых окон (Dialog and Window Expert).
15. Добавление предложения для предиката обработчика событий окна с помощью элемента группирования Event Handling и окон Event Type и Event or Item.
16. Порядок добавления нового пункта меню.
17. Порядок создания и активизации окна.
18. Порядок создания диалогового окна.
19. Основные элементы управления, используемые при визуальном программировании.

ЛАБОРАТОРНАЯ РАБОТА 16.

СИСТЕМА ПРОГРАММИРОВАНИЯ MULISP

1. Цель работы: приобретение практических навыков работы в системе программирования muLisp.

2. Краткие справочные данные.

2.1. **Запуск интерпретатора muLisp**.

Запуск интерпретатора осуществляется путем инициализации файла **mulisp.com** (в нашем случае через bat-файл **lisp.bat**).

После выдачи начальных сообщений система высвечивает приглашение (знак доллара $), которое означает, что muLisp готов к вводу с клавиатуры.

Далее пользователь набирает некоторое выражение и нажимает клавишу Enter. Интерпретатор оценивает это выражение и печатает результирующее значение в начале новой строки, затем снова высвечивается приглашение $ и т.д. Этот цикл взаимодействия интерпретатора с пользователем повторяется до тех пор, пока не будет введена системная команда (SYSTEM), которая завершает работу и возвращает управление операционной системе.

2.2. **Механизм работы Лисп-системы**.

Так как Лисп является функциональным языком, то любая программа на Лиспе – тоже функция. Программа на Лиспе записывается в виде s-выражения (символьного выражения) как обращение к функции.

В общем случае обращение к функции имеет следующий вид:

(F Arg1 Arg2 … ArgN).

Здесь F – первый элемент списка, интерпретируется как имя функции, которую необходимо выполнить, взяв в качестве аргументов оставшиеся элементы списка Arg1, Arg2, …, ArgN.

Механизм работы Лисп-системы состоит из трех последовательных шагов (read-eval-print):

- считывание s-выражений (READ);
- интерпретация s-выражений (EVAL);
- печать s-выражения (PRINT).

Интерпретация s-выражений – это единственная и главная задача Лисп-интерпретатора. Ее выполняет функция Лисп-системы EVAL, которая берет одно s-выражение, интерпретирует его, если это возможно, и возвращает другое s-выражение как результат интерпретации первого s-выражения.

2.3. **Прерывания при работе интерпретатора muLisp**.

При работе интерпретатора могут возникнуть внутренние и внешние прерывания (остановы). В этом случае на экране дисплея высвечивается подсказка в виде опций:

Continue, Break, Abort, Top-level, Restart, System?

Затем система ждет, пока пользователь выберет одну из опций по ее имени (C, B, A, T, R или S соответственно).

Действия опций:

Continue (продолжить): возвращает управление программе, которая вызвала прерывание.

Break (останов): временно приостанавливает выполнение программы и выходит на следующий нижний уровень цикла read-eval-print (чтение – вычисление – печать). Выход из останова – команда (RETURN).

Abort (прерывание): прерывает выполнение программы, присваивает формальным параметрам, размещенным в стеке переменных, первоначальные значения и возвращает управление на текущий уровень цикла read-eval-print.

Top-level (верхний уровень): прерывает выполнение программы, присваивает первоначальные значения формальным параметрам, которые размещены в стеке переменных, выводит текущие входные и выходные данные и возвращает управление верхнему уровню цикла read-eval-print.

Restart (повторный старт): перезагрузка системы.

System (система): выход из системы Лисп.

В любое время в ходе выполнения программы пользователь может прервать ее работу, нажав клавишу Esc на клавиатуре.

2.4. **Редактор muLisp**.

2.4.1. **Загрузка редактора и опции главного меню**.

Загрузка редактора в существующую среду muLisp выполняется командой:

(RDS Edit), а затем (RETURN).

Как только редактор начинает работать, он чистит экран, рисует рамку и выдает головное меню: Edit, Print, Screen, Lisp, Quit:.

Затем система ждет, пока пользователь не наберет одну из опций путем ввода первой буквы ее имени (E, P, S, L или Q).

- Edit: выдает на экран подсказку Edit file: и ждет, пока не будет введено имя файла для редактирования. После ввода имени файла (напр., D:\107215\aaa) нажимается Enter. По умолчанию используется расширение .lsp и текущий каталог.
- Print: выдает на экран подсказку Print file: и ждет, пока будет введено имя файла для печати.
- Screen: выдает на экран подсказку Full, Vertical, Horizontal: и ждет выбора одной из опций (F, V или H). Опция Full screen использует для редактирования полный экран. Опция Vertical split screen использует правую половину экрана для редактирования, а левую – для отладки. Опция Horizontal split screen использует верхнюю половину экрана для редактирования, а нижнюю – для отладки.
- Lisp: завершает работу редактора; на экране появляется знак доллара. Из среды muLisp можно вернуться в редактор по команде (RETURN).
- Quit: завершает работу редактора и системы muLisp.

2.4.2. **Команды управления курсором**.

Основные команды перемещения курсора:

Ctrl-Q, D – в правый конец текущей строки,

Ctrl-Q, S – в левый конец текущей строки,

Ctrl-Q, X – в нижний конец экрана,

Ctrl-Q, E – в верхний конец экрана,

Ctrl-Q, I – влево в предыдущую позицию табуляции.

2.4.3. **Команды скроллинга экрана**.

Ctrl-Z – сдвигает текст вверх на одну строку,

Ctrl-W – сдвигает текст вниз на одну строку,

Ctrl-R – сдвигает текст на высоту экрана вниз,

Ctrl-C - сдвигает текст на высоту экрана вверх,

Ctrl-Q, C – перемещает курсор и окно в конец файла,

Ctrl-Q, R - перемещает курсор и окно в начало файла.

2.4.4. **Изменение режима ввода текста**.

Для переключения из режима вставки (Insert) в режим замещения (Replace) и наоборот используется команда Ctrl-V.

2.4.5. **Команды удаления и восстановления текста**.

Ctrl-Y – удаляет строку, на которой располагается курсор,

Ctrl-Q, Y – удаляет правый конец строки, начиная с курсора,

Ctrl-U – восстанавливает текст, уничтоженный последней командой удаления.

2.4.6. **Команды блочных операций**.

Ctrl-O, B – отмечает начало блока,

Ctrl-O, E – отмечает конец блока,

Ctrl-O, C – копирует отмеченный блок в место, указанное курсором,

Ctrl-O, M – перемещает отмеченный блок в положение курсора,

Ctrl-O, R – читает текст из указанного файла и вставляет в место, указанное курсором,

Ctrl-O, W – запись отмеченного блока в указанный файл.

2.4.7. **Команды списковых структур**.

Команды списковых структур позволяют перемещать, копировать, уничтожать, считывать из файла и записывать в файл s-выражения. Двухсимвольные команды, начинающиеся с Esc, инициируют команды списковых структур.

Esc-D – перемещает курсор вперед на одно s-выражение,

Esc-S – перемещает курсор назад на одно s-выражение,

Esc-F – перемещает курсор вперед, устанавливая его после конца списка,

Esc-A - перемещает курсор назад, устанавливая его перед началом списка,

Esc-T – уничтожает s-выражение под курсором и справа от курсора,

Esc-Y – уничтожает определение, где располагается курсор,

Esc-L – закрывает редактор и передает управление в окно вычисления Lisp. Возврат в редактор – команда (RETURN).

Esc-! – выдает в окно вычисления Lisp результат вычисления s-выражения под курсором и справа от него. Данная команда дает возможность выборочно переопределять определения редактируемых функций; это плезно при их тестировании.

2.4.8. **Команды сохранения файлов** (начинаются с Ctrl-K).

Через несколько секунд после ввода Ctrl-K выдается подсказка, если пользователь не успел ввести второй символ команды.

Ctrl-K, D – сохраняет текст в виде файла и возвращает управление в головное меню редактора,

Ctrl-K, S – сохраняет текст в виде файла и заново открывает файл для редактирования.

Если в текст вносились изменения, Ctrl-K, A и Ctrl-K, Q высвечивают подсказку:

Abandon <filename>? (Y/N),

где <filename> - имя отредактированного файла. При вводе Y отредактированный текст удаляется, а управление возвращается в головное меню редактора. В противном случае никаких действий не происходит. Если изменения не вносились, текст просто удаляется.

2.4.9. **Команды окна**.

Ctrl-Q, W – позволяет изменить размер окна и его форму на время редактирования файла. Команда выдает подсказку

Full, Vertical, Horizontal

и ждет выбора одной из опций (F, V или H) (см. пункт 2.4.1).

2.5. **Трассировка программы**.

Чтобы выполнить трассировку программы, нужно загрузить отладочный пакет muLisp'a с помощью команды

(RDS 'DEBUG)

Трассировка функций осуществляется с помощью функции TRACE. Она вызывается с одной или более функциями для трассировки в качестве своих аргументов. Например, команда

(TRACE 'APP 'REV)

трассирует функции APP и REV.

Если после этого вызвать функцию APP (объединение двух списков) следующим образом

(APP '(A B) '(C D E)),

то будет выведена трасса для этой функции.

Если вывод трассы осуществляется слишком быстро, то для временной остановки вывода можно набрать на клавиатуре Ctrl-S.

Трасса также может быть выведена командой (HISTORY).

Для запрещения трассировки используется команда CLEAR. Например, команда (CLEAR 'APP 'REV) запрещает трассировку функций APP и REV.

2.6. **Примитивы Лиспа**:

- функция (**car L**) – возвращает голову списка L, напр., (car '((a b) c d)) возвращает (A B);
- функция (**cdr L**) – возвращает хвост списка L, напр., (cdr '((a b) c d)) возвращает (C D);
- функция (**cons S1 S2**) – включает элемент S1 в начало списка S2, напр., (cons 'a '(b c)) возвращает (A B C);
- предикат (**atom S**) – проверяет, является ли аргумент S атомом, напр., (atom '()) возвратит T, а (atom '(nil)) возвратит NIL;
- предикат (**eq S1 S2**) – проверяет тождественность двух символов S1 и S2, напр., (eq 'a (car '(a b c))) возвратит T, а (eq 'a '(a)) возвратит NIL;
- функция (**quote S**) – блокирует вычисление выражения S, значением функции является символьное выражение S, представленное в вызове, напр., (quote (+ 2 3)) возвращает (+ 2 3). Синонимом для (quote (+ 2 3)) является запись '(+ 2 3).

3. Содержание задания по лабораторной работе.

3.1. Загрузить muLisp.

3.2. Ознакомиться с возможностями редактора muLisp.

3.3. Проделать следующие вычисления с помощью интерпретатора Лиспа:

a) 3.234*(4.56+21.45)
b) 5.67+6.342+12.97
c) (454-214-657)/(456+678*3)

3.4. Написать выражение, вычисляющее среднее арифметическое чисел 23, 5, 43 и 17.

3.5. Определить значение следующих выражений:

a) '(+ 2 (* 3 4))
b) (+ 2 '(* 3 4))
c) (+ 2 ('* 3 4))
d) (+ 2 (* 3 '4))
e) (quote 'quote)
f) (quote 2)
g) '(quote nil)

3.6. Записать последовательность вызовов CAR и CDR, выделяющие из приведенных ниже списков символ «цель». Упростить эти вызовы с помощью функций C…R.

a) (1 2 цель 3 4)
b) ((1) (2 цель) (3 (4)))
c) ((1 (2 (3 4 цель))))

3.7. Вычислить значения следующих выражений. Проверить результат на компьютере.
a) (cons nil ‘(суть пустой список))
b) (cons nil nil)
c) (cons ‘(nil) ‘(nil))
d) (cons (car ‘(a b)) (cdr ‘(a b)))
e) (car ‘(car (a b c)))
f) (cdr (car (cdr ‘(a b c))))

3.8. Проверить, какие из следующих вызовов возвращают значение T ?
a) (atom ‘(cdr nil))
b) (atom (* 2 (+ 2 3)))
c) (atom (atom (+ 2 3)))
d) (eq nil ( ))
e) (eq t (atom (car ‘(мышь))))

3.9. Выполнить трассировку функции APP, объединяющей два списка.
Определение функции:

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

4. Содержание отчета: цель работы, постановка задачи, диалог пользователя с интерпретатором muLisp, выводы. (Для формирования отчета использовать редактор muLisp).

Контрольные вопросы:

1. Почему Lisp называют языком функционального программирования ?
2. Чем отличается запись списка на языке Lisp от аналогичной записи на языке Пролог ?
3. Инфиксная и префиксная формы записи выражений; какая из этих форм принята в Lisp ?
4. Пять базовых функций (примитивов) Lisp.
5. Для чего используется функция QUOTE, чем ее можно заменить ?
6. Что означает символ $ в начале строки при работе с интерпретатором muLisp ?
7. Как производится выход из системы muLisp ?
8. Как будет интерпретироваться системой muLisp s-выражение (a b c d e) ?
9. Как загружается редактор в среду muLisp ?
10. Как можно запустить на выполнение функцию, набранную в редакторе muLisp?
11. По какой команде можно перейти из редактора в окно вычислений muLisp ?
12. По какой команде можно вернуться из окна вычислений в редактор ?
13. Как можно сохранить отредактированный в редакторе текст в виде файла ?
14. Как производится трассировка программы в среде muLisp ?

ЛАБОРАТОРНАЯ РАБОТА 17.

ОПРЕДЕЛЕНИЕ ФУНКЦИЙ В LISP

1. Цель работы: приобретение практических навыков определения функций на языке Lisp.

2. Краткие справочные данные.

2.1. **Представление данных в Лисп.**

Данные в Лисп представляются **атомами, списками, консами, символьными выражениями**. Взаимосвязь понятий иллюстрируется рис. 4.

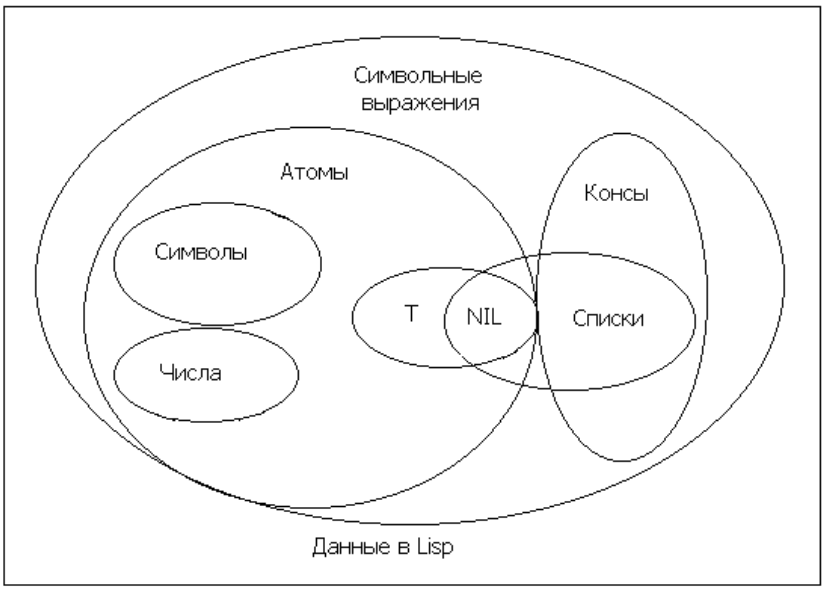


Рис. 4. Данные в Lisp

Атомы, простейшие объекты в Lisp, делятся на символьные и числовые. **Символьный атом** – это последовательность букв, цифр и специальных символов, например, X7, Month8, A-1. **Числовой атом** – это последовательность цифр, включающая возможно символы ‘+’, ‘-‘, ‘.’, ‘/’, например, 12, -8, 3.14, 2/7. Числовые атомы интерпретируются как константы, символьные – как константы и как переменные. Атом T означает логическое значение “истина”, атом NIL – логическое значение “ложь” или пустой список.

Списки – это последовательность элементов, разделенных пробелами и заключенных в круглые скобки. Элементами списка могут быть любые объекты Лиспа: атомы, списки, консы, например, (1 a 2 b 3 c), ((x1 0) (x2 1)), ((y.blue) (z.yellow)). Пустой список не содержит элементов и обозначается ( ) или NIL.

Консы – это пара элементов, разделенных точкой и заключенных в круглые скобки. Список (e1 e2 … eN) представляется суперпозицией консов (e1. (e2. (… (eN.NIL) …))).

Символьным выражением в Лисп является один из следующих объектов:

1) атом,
2) список (s1 … sn),
3) конс (s1.s2),

где s1, s2, …, sn – символьные выражения.

Например, (2 (T.NIL).(y z)) является символьным выражением.

2.2. **Префиксная форма записи выражений**.

В языке Лисп как для вызова функции, так и для записи выражений принята **единообразная префиксная форма** записи, при которой как имя функции или действия, так и сами аргументы записываются внутри скобок.

Например, в математике вызов функции записывается следующим образом:

f(x), g(x,y), h(x, g(y,z)) и т.д.

На Лиспе эти же записи выглядят следующим образом:

(f x), (g x y), (h x (g y z)) и т.д.

Соответственно арифметические выражения x+y, x*(y+z), x*x-y*y на Лиспе будут записаны в виде: (+ x y), (* x (+ y z)), (- (* x x) (* y y)).

2.3. **Примитивы Лиспа, используемые при определении функций**:

- ***функции*** **car, cdr, cons, quote**, ***предикаты*** **atom, eq** (см. лаб. работу 16);

- ***математические функции***:

(+ x1 x2 … xN) – выполняет сложение всех аргументов;

(- x1 x2 … xN) – возвращает разность между первым аргументом и всеми остальными;

(* x1 x2 … xN) – выполняет произведение всех аргументов;

(/ x1 x2 … xN) – возвращает результат деления первого аргумента на все остальные;

(**min x1 x2 … xN**) – возвращает наименьшее xi;

(**max x1 x2 … xN**) – возвращает наибольшее xi;

- ***функции сравнения чисел***:

(= x1 x2 … xN) – числа равны;

(/= x1 x2 … xN) – числа не равны;

(< x1 x2 … xN) – числа возрастают:

(> x1 x2 … xN) – числа убывают;

(<= x1 x2 … xN) – числа возрастают или равны;

(>= x1 x2 … xN) – числа убывают или равны;

- ***логические функции***:

(**and S1,S2,…,Sn**) – логическое И;

(**or S1,S2,…,Sn**) – логическое ИЛИ;

(**not S**) – логическое НЕ;

- ***функции проверки знака числа***:

(plusp число) – число положительное;

(minusp число) – число отрицательное;

(zerop число) – нуль;

- ***функции назначения***:

(setq x Sv) – переменной x приписывается значение выражения Sv;

(set Sv1 Sv2) – присваивает значение выражения Sv2 значению выражения Sv2;

- предикат (**null Sv**) – проверяет, является ли значением аргумента Sv пустой список, напр., (null (cdr ‘(a))) возвращает T;

- предикат **(equal S1 S2)** – проверяет идентичность записей S1 и S2, позволяет сравнивать однотипные числа, а также проверяет одинаковость двух списков, напр., (equal 5 (+ 2 3)) возвращает T; (equal ‘(a b c) (cons ‘a ‘(b c))) возвращает T;

- функция **(list x1 x2 … xN)** – возвращает в качестве своего значения список из значений аргументов, напр., (list ‘a ‘b (+ 2 3)) возвращает (A B 5).

2.4. **Управляющие структуры**.

В Лиспе имеются управляющие структуры для организации последовательных вычислений, разветвлений и циклов. Управляющие структуры (будем называть их предложениями) выглядят внешне как вызовы функций. Предложение записывается в виде скобочного выражения, первый элемент которого действует как имя управляющей

структуры, а остальные элементы – как аргументы. В качестве аргументов используются формы. Под **формой** понимается такое символьное выражение, значение которого может быть найдено интерпретатором.

Для **последовательных** вычислений используются предложения **prog1** и **progn**.

(prog1 форма1 форма2 … формаN)

(progn форма1 форма2 … формаN)

У этих предложений переменное число аргументов, которые они последовательно вычисляют и возвращают в качестве значения значение первого (prog1) или последнего (progn) аргумента.

Prog1 вычисляет форму1, затем оставшиеся формы и возвращает результат вычисления формы1. Prog1 часто используется для того, чтобы не вводить временные переменные для хранения результатов в процессе вычисления других выражений.

(setq FOO ‘(a b c d)) -------→ (A B C D)

(prog1 (car FOO) (setq FOO (cdr FOO))) --------→ A

FOO ------→ (B C D)

Progn последовательно вычисляет формы, начиная с формы1 и выдает значение последней формы.

```
(progn (setq num1 (+ 2 5))
       (setq num2 (* 3 4))
       ((< num1 num2)
          ((minusp num1)
             (* 3 num2))
          (+ num1 num2))
        (- num1 num2))          ----------→ 19
```

Основным средством **разветвления** вычислений является предложение **cond**.

(cond (p1 a1) (p2 a2) … (pN aN))

Здесь p1, p2, …, pN – предикаты, a1, a2, …, aN – произвольные выражения.

Cond последовательно вычисляет предикаты pi до появления значения, отличного от nil, или до исчерпания всех выражений в cond-аргументах. В первом случае значением cond будет значение выражения ai, соответствующее первому, отличному от nil, предикату pi. Если значения всех предикатов есть nil, то значением cond будет nil.

Пример. Сколько элементов в списке ? (Два варианта – 1 или больше1).

```
(cond ((cdr L) ‘больше 1)
      ( t   1))
```

В простом случае для разветвлений может использоваться предложение **if**:

(if условие то-форма иначе-форма)

Пример. (if (atom t) ‘атом ‘список) --------→ атом

Для программирования циклических вычислений обычно используется рекурсия.

2.5. **Определение функций**.

Определение функций в среде Лисп-системы осуществляется с помощью примитива defun.

(defun name (arg1 arg2 … argN)

(Ev1) (Ev2) … (EvN)),

где name – имя определяемой функции,

arg1, arg2, …, argN – список формальных параметров в виде переменных,

Ev1, Ev2, …, EvN – s-выражения, связывающие переменные определяемой функции и вычисляющие ее значение.

Пример 1. Определим функцию, выделяющую второй элемент списка:

```
(defun  Второй  (Lst)
  (car  (cdr  Lst)))
```

Пример 2. Определим логическую функцию И:

```
(defun  И  (x  y)
  (cond  (x  y)
         (t  nil)))
```

Пример 3. Определим рекурсивную функцию mem, проверяющую, является ли атом x элементом некоторого списка y.

```
(defun  mem  (x  y)
  (cond  ((null  y)  nil)
         ((eq  x  (car  y))  t)
         ( t   (mem  x  (cdr  y)))))
```

Пример 4. Определим функцию ! вычисления факториала:

```
(defun  !  (N)
  (cond  ((zerop  N)  1)
         (t  (*  N  (!  (-  N  1))))))
```

3. Содержание задания по лабораторной работе.

3.1.Определить функцию, вычисляющую x + y – x*y.

3.2.Определить функции (null x), (caddr x) и (list x1 x2 x3) с помощью базовых функций (использовать имена null1, caddr1, list1, чтобы не переопределять одноименные встроенные функции Лисп).

3.3.Определить логическую функцию ИЛИ (OR) (имя OR не использовать).

3.4.Определить логическую функцию НЕ (NOT) (имя NOT не использовать).

3.5.Определить N-ое число Фибоначчи.

3.6.Подсчитать количество элементов списка.

3.7.Все элементы списка увеличить на заданное число.

3.8.Выделить последний элемент списка.

3.9.Добавить заданный элемент в конец списка.

3.10. Удалить из списка последний элемент.

3.11. Удалить из списка первое вхождение заданного элемента на верхнем уровне.

3.12. Удалить из списка все вхождения заданного элемента на верхнем уровне

3.13. Удалить из списка каждый второй элемент.

3.14. Определить среднее арифметическое элементов списка.

3.15. Подсчитать количество отрицательных элементов списка.

3.16. Определить минимальный элемент целочисленного списка (не используя встроенную функцию min).

3.17. Заменить в списке все элементы со значением X на значение Y.

3.18. Удалить начало списка до заданного элемента X (включительно).

3.19. Получить новый отсортированный список путем вставки заданного элемента в исходный отсортированный в порядке возрастания элементов список.

3.20. Определить функцию, разбивающую список (a b c d …) на пары ((a b) (c d) …).

3.21. Определить функцию, которая, чередуя элементы списков (a b …) и (1 2 …), образует новый список (a 1 b 2 …).

3.22. Определить функцию (первый-совпадающий x y), которая возвращает первый элемент, входящий в оба списка x и y, в противном случае nil.

3.23. Проверить, является ли список множеством, т.е. входит ли каждый элемент в список лишь один раз.

3.24. Преобразовать список в множество, т.е. удалить из списка повторяющиеся элементы.

3.25. Определить предикат Равенство-множеств, проверяющий совпадение двух множеств (независимо от порядка следования элементов).

3.26. Определить предикат Подмножество, проверяющий, является ли одно множество подмножеством другого.

3.27. Определить функцию Пересечение, определяющую общие для дух множеств элементы.

3.28. Определить предикат Непересекающиеся, проверяющий, что два множества не имеют общих элементов.

3.29. Определить функцию Объединение, формирующую объединение двух множеств.

3.30. Определить функцию Разность, формирующую разность двух множеств, т.е. удаляющую из первого множества все общие со вторым множеством элементы.

4. Содержание отчета: цель работы, постановка задачи, тексты определяемых функций, результаты тестирования функций, выводы.

Замечания:

- при определении функций использовать только следующие примитивы Lisp: car, cdr, cons, atom, eq, quote, plusp, minusp, zerop, set, setq, equal, null, list, математические функции, функции сравнения чисел и логические функции and , or, not;
- в заданиях 3.22 – 3.30 использовать вспомогательную функцию, проверящую принадлежность заданного элемента списку;
- для формирования отчета использовать редактор muLisp.

Контрольные вопросы:

1. Какие объекты относятся к символьным выражениям Lisp ?
2. Какие объекты относятся к атомам Lisp ?
3. Что означает атом NIL ?
4. Что такое конс ?
5. Префиксная форма записи выражений в Lisp.
6. Какие функции в Lisp называются предикатами ?
7. Управляющие структуры Lisp для последовательных вычислений.
8. Управляющие структуры Lisp для разветвления вычислений.
9. Основной механизм программирования циклических вычислений.
10. Определение функций в Lisp.
11. Можно ли в качестве имени определяемой функции взять имя стандартной функции Lisp.
12. Принципиальное отличие восходящей рекурсии от нисходящей рекурсии.
13. Вспомогательные (промежуточные) функции.
14. Суть двумерной рекурсии.

ЛИТЕРАТУРА

1. Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. - СПб.: БХВ-Петербург, 2003.
2. Братко И. Программирование на языке Пролог для искусственного интеллекта. - М.: Мир, 1990.
3. Городняя Л.В. Основы функционального программирования. М.: ИНТУИТ.РУ «Интернет-университет Информационных Технологий», 2004.
4. Доорс Дж., Рейблейн А.Р., Вадера С. Пролог - язык программирования будущего. - М.: Финансы и статистика, 1990.
5. Ин Ц., Соломон Д. Использование Турбо-Пролога. М.:Мир,1993.
6. Клоксин У., Меллиш К. Программирование на языке Пролог. - М.: Мир, 1987.
7. Ковальков А.Т., Ковалькова И.А. Лабораторный практикум по дисциплине “Функциональное и логическое программирование” для студентов специальностей 1-40 01 01 и 1-40 01 02. – Минск, БГПА, 2005.
8. Макаллистер Дж. Искусственный интеллект и Пролог на микро-ЭВМ. - М.: Машиностроение, 1990.
9. Малпас Дж. Реляционный язык Пролог и его применение. - М.: Наука, 1990.
10. Марселлус Д. Программирование экспертных систем на Турбо-Прологе. - М.: Финансы и статистика, 1994..
11. Набебин А.А. Логика и Пролог в дискретной математике. - М.: Изд. МЭИ, 1996.
12. Прихожий А.А. Функциональное и логическое программирование: Метод. пособие для студентов специальности «Программное обеспечение информационных технологий». Мн.: БГУИР, 1998.
13. Стерлинг Л., Шапиро З. Искусство программирования на языке Пролог. - М.: Мир, 1990.
14. Стобо Дж. Язык программирования Пролог. - М.: Радио и связь, 1993.
15. Хендерсон П. Функциональное программирование. -М.: Мир, 1983.
16. Хювёнен Э., Сеппянен И. Мир ЛИСПа. Том 1, 2. -М.: Мир, 1990.
17. Шрайнер П.А. Основы программирования на языке Пролог. – М.: Интернет – Ун-т Инфом. Технологий, 2005.
18. Янсон А. Турбо-Пролог в сжатом изложении. - М.:Мир, 1991.

ПРИЛОЖЕНИЕ

```
domains
 день,год,оклад=integer
 имя,фамилия,месяц,должность=string
 дата=дат(день,месяц,год)
 работа=раб(должность,оклад)
 член_семьи=чл_с(имя,фамилия,дата,работа)
 список_детей=член_семьи*
 список_имен_детей=string*
predicates
 семья(член_семьи,член_семьи,список_детей) %описание всех членов семьи
 муж(член_семьи)                          %муж
 жена(член_семьи)                         %жена
 список_детей_семьи(список_детей)         %дети
 количество_элементов_списка(список_детей,integer) %подсчет количества
                                                    %элементов в списке
 количество_детей_семьи(string,integer) %количество детей в данной семье
 имена_детей(string,список_имен_детей)  %список имен детей в данной семье
 выбор_имен_детей(список_детей,список_имен_детей) %выбор имен детей
                                                   %из списка детей
clauses
 семья(чл_с("Иван","Иванов",дат(1,"май",1940),раб("инженер",200)),
       чл_с("Анна","Иванова",дат(15,"январь",1945),раб("врач",180)),
      [чл_с("Олег","Иванов",дат(22,"март",1970),раб("студент",40)),
       чл_с("Инна","Иванова",дат(7,"июль",1975),раб("ученица",0))]).
 семья (чл_с("Лев","Сидорик",дат(12,"февраль",1949),раб("хирург",300)),
       чл_с("Раиса","Сидорик",дат(5,"ноябрь",1952),раб("домохозяйка",0)),
      [чл_с("Борис","Сидорик",дат(2,"май",1973),раб("артист",140))]).
 семья (чл_с("Степан","Петров",дат(1,"август",1940),раб("шофер",200)),
       чл_с("Мария","Дрозд",дат(15,"июнь",1945),раб("кассир",180)),[]).
 семья (чл_с("Леонид","Загорский",дат(1,"май",1960),раб("инженер",200)),
       чл_с("Нина","Загорская",дат(5,"апрель",1965),раб("техник",180)),
      [чл_с("Елена","Загорская",дат(2,"январь",1983),раб("ученица",0)),
       чл_с("Ева","Загорская",дат(15,"февраль",1985),раб("ученица",0)),
       чл_с("Софья","Загорская",дат(15,"февраль",1985),раб("ученица",0))]).
 семья (чл_с("Николай","Савич",дат(1,"март",1970),раб("плотник",200)),
       чл_с("Нина","Савич",дат(15,"январь",1965),раб("маляр",180)),
      [чл_с("Игорь","Савич",дат(12,"ноябрь",1990),раб("ученик",0)),
       чл_с("Игнат","Савич",дат(7,"декабрь",1994),раб("дошкольник",0))]).
 семья (чл_с("Кирилл","Казак",дат(11,"май",1970),раб("аспирант",100)),
       чл_с("Жанна","Казак",дат(15,"март",1975),раб("студентка",50)),
      [чл_с("Ольга","Казак",дат(17,"июль",1995),раб("дошкольница",0))]).
 семья (чл_с("","",дат(0,"",0),раб("",0)),
       чл_с("Анна","Лоза",дат(15,"март",1975),раб("студентка",50)),
      [чл_с("Ольга","Лоза",дат(17,"июль",1995),раб("дошкольница",0))]).
 муж(чл_с(Имя,Фам,Дат,Раб)):-семья(чл_с(Имя,Фам,Дат,Раб),_,_).
 жена(чл_с(Имя,Фам,Дат,Раб)):-семья(_,чл_с(Имя,Фам,Дат,Раб),_).
 список_детей(Сп_дет):-семья(_,_,Сп_дет).
 количество_элементов_списка([],0).
 количество_элементов_списка([_|Хвост],Кол):-
      количество_элементов_списка(Хвост,Кол1),Кол=Кол1+1.
 количество_детей_семьи(Фам_мужа,Кол_дет):-
      семья(чл_с(_,Фам_мужа,_,_),_,Сп_дет),
      количество_детей_семьи(Сп_дет,Кол_дет).
 имена_детей(Фам_мужа,Сп_имен_дет):-
      семья(чл_с(_,Фам_мужа,_,_),_,Сп_дет),
      выбор_имен_детей(Сп_дет,Сп_имен_дет).
 выбор_имен_детей([],[]).
 выбор_имен_детей([Ребенок|Хвост_списка_детей],[Имя|Хвост_имен]):-
      Ребенок=чл_с(Имя,_,_,_),
      выбор_имен_детей(Хвост_списка_детей,Хвост_имен).
```