Министерство образования и науки Российской Федерации
Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Кафедра управления и информатики в технических системах

И.И. Стрекалова

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Методические указания к лабораторным работам

Рекомендовано к изданию Редакционно-издательским советом Государственного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет»

Оренбург
ИПК ГОУ ОГУ
2010

УДК 004 .42 (076.5)
ББК 32.973-018я73
С 84

Рецензент – доцент, кандидат технических наук Д.В. Горбачев

Стрекалова, И.И.
С84 Логическое программирование: методические указания к лабораторным работам /И.И. Стрекалова; Оренбургский гос. ун-т. – Оренбург: ОГУ, 2010. – 94 с.

Методические указания предназначены для выполнения лабораторных работ, обеспечивающих учебный процесс по дисциплине «Логическое программирование» для студентов 2 курса специальности 230100 очной формы обучения.

Методические указания содержат 10 лабораторных работ, рекомендуемую литературу и являются основным учебным руководством при выполнении работ студентами.

УДК 004 .42 (076.5)
ББК 32.973-018я73

Содержание

Введение

Предмет «Логическое программирование» является дисциплиной по выбору для специальности 230100 «Информатика и вычислительная техника». Она базируется на следующих дисциплинах: «Математическая логика и теория алгоритмов», «Дискретная математика», «Структуры и алгоритмы обработки данных», «Программирование на языке высокого уровня».

Курс рассчитан на 54 часа лабораторно-практических занятий. Формой итогового контроля является экзамен, который проводится в пятом семестре.

В данном издании содержатся 10 лабораторных работ по предмету «Логическое программирование». В начале каждой лабораторной работы даны краткие теоретические сведения, необходимые для решения всех последующих примеров и задач. Приведенные примеры типовых практических задач даются с подробными объяснениями. Также в каждой лабораторной работе представлены содержание отчета и примерный перечень вопросов, которые необходимы для защиты.

В учебном процессе дисциплины «Логическое программирование» используется язык Turbo Prolog. Название Prolog происходит от Programming in logic (программирование в логике). Turbo Prolog является компиляторно-ориентированным языком программирования высокого уровня. Он предназначен для программирования задач из области искусственного интеллекта (ИИ). Как язык программирования ИИ Turbo Prolog особенно хорош для создания экспертных систем, динамических баз данных, программ с применением естественно-языковых конструкций. Он также может быть использован и для других задач общего характера. Turbo Prolog имеет окна, цветную графику и интерактивные средства ввода-вывода, что свидетельствует о его максимальном удобстве для пользователя прикладных программ. Turbo Prolog имеет великолепный полноэкранный редактор, множество рабочих окон и интерактивный диалоговый отладчик. Он поддерживает цветную графику IBM PC, снабженного цветным графическим адаптером (CGA) и расширенным графическим адаптером (EGA). Предикаты графики и система с

графическим экранным пером являются составной частью Turbo Prolog. Он также снабжен средствами работы с последовательными файлами, файлами прямого доступа и двоичными файлами. Поэтому, язык Turbo Prolog был выбран специально для проведения лабораторно-практических занятий по дисциплине «Логическое программирование».

1 Лабораторная работа №1. Знакомство со средой программирования Turbo Prolog. Отладка и трассировка программ

Лабораторная работа №1 предназначена для приобретения практических навыков в работе со средой Turbo Prolog фирмы Borland, а также в составлении, отладке и выполнения простейших программ в среде программирования Turbo Prolog.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пункту 1.2 и 1.3.

На выполнение и защиту лабораторной работы №1 отводится 4 академических часа.

1.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);
2) выполнить последовательно задания, приведенные в пункте 1.2;
3) защитить лабораторную работу.

1.2 Краткие теоретические сведения

Запуск среды программирования Turbo Prolog.

Для работы в среде Turbo Prolog нужно сначала запустить файл prolog.exe и мы попадем в среду Turbo Prolog, в соответствии с рисунком 1.

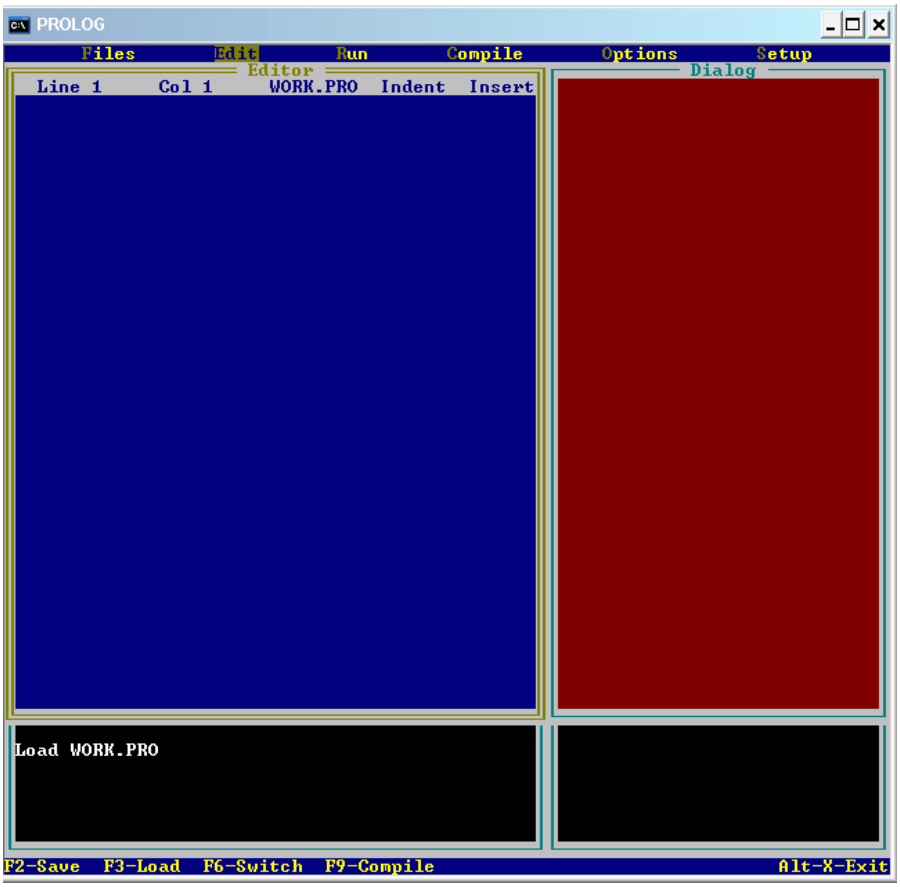


Рисунок 1– Среда Turbo Prolog

Описание среды Turbo Prolog.

Главное меню Турбо-Пролога высвечивает 7 доступных пользователю опций (команд) в верхней части экрана. Первая буква названия каждой из команд выделена при помощи увеличенной яркости. Выделение имеет целью напоминать, что для задания команды достаточно нажать лишь первую букву ее названия.

Команды определяются 7 функциями Турбо-Пролога, каковыми являются:

– запуск программы на счет (Run);

– трансляция программы (Compile);

– редактирование текста программы (Edit);

– заданий опций компилятора (Options);

– работа с файлами (Files);

– настройка системы в соответствии с индивидуальными потребностями (Setup);

– выход из системы (Quit).

В приложении А подробно приведены назначения пунктов меню.

Переход от одной команды к другой прост и удобен. Существует простой способ задания команд. Он требует нажатия клавиши, соответствующей первой букве названия выбранной команды. Так, для выбора команды Edit необходимо нажать E. (Нет никакой разницы, какая буква была введена большая или маленькая, т. е. использование Shift не обязательно.) Для окончания работы с командой используется клавиша Esc.

Главное меню содержит четыре окна. В левом верхнем углу располагается окно редактора Турбо-Пролога (Editor), в правом верхнем углу - окно диалога (Dialog), в левом нижнем - окно сообщений (Message), в правом нижнем - окно трассировки (Trace). По умолчанию для окна редактора задается голубой цвет, для окна диалога - красный и черный - для окон сообщений и трассировки. Верхняя строка окна редактора содержит информацию о высвечиваемом в этом окне файле Line 1 и Col 1 свидетельствуют о том, что курсор в настоящий момент располагается в первой позиции первой строки. Значения этих индикаторов строки и позиции меняются вслед за изменением положения курсора. Надпись Indent сигнализирует о том, что включен режим автоматического выравнивания строк, а надпись Insert - о том, что задан режим вставки. WORK.PRO является заданным по умолчанию именем рабочего файла; .PRO есть заданное по умолчанию расширение для файлов, содержащих программы на Турбо-Прологе. Если вы набьете в редакторе какой-либо текст и запишите его на диск без изменения имени файла, то файл с вашим текстом получит имя WORK.PRO.

Размеры окон можно менять при помощи команд меню. Нужно «добраться» до пункта меню Setup (Alt-S) в соответствии с рисунком 2. Выбрав нужное окно в меню и нажав Enter, можно нажимать клавиши управления курсором, при этом будут изменяться размеры выбранного окна. Зафиксировать установленный размер окна можно нажатием Enter.

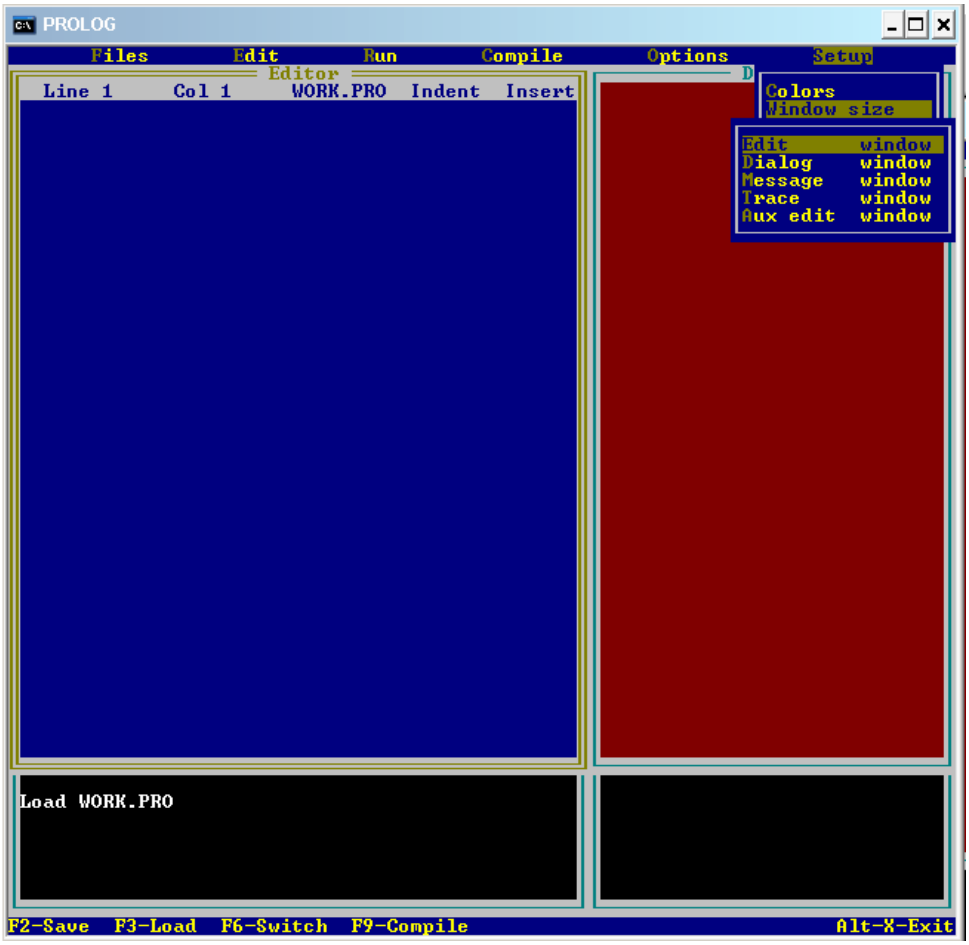


Рисунок 2 – Выбор меню в среде Turbo Prolog

Начало работы в Турбо-Пролог.

Программа, работу с которой мы сейчас начинаем, имеет целью дать необходимые навыки в использовании меню системы и основных команд редактора. Создайте новый файл (Files-New File) и переименуйте его как WELCOME.PRO (Files-Write to).

Наберите текст программы WELCOME.PRO:

```
predicates
hello
goal
hello.
clauses
```

```
hello :-
write("Hello, students! "), nl.
```

Примечание – Здесь и далее машинописным шрифтом Courier New: 0, 1, 2, A, B, C, d … – обозначается текст программы, вводимы и выводимый в среде Turbo Prolog.

Когда вы доходите до конца очередной строки, нажимайте клавишу Enter для перехода на следующую. Для удаления неверного символа нужно прибегнуть к помощи клавиши BackSpace. После запуска программы на экране появится следующая строка:

```
Hello, students!
```

Вы ввели в компьютер вашу первую программу на Турбо-Прологе. Для того чтобы запустить ее, сначала требуется покинуть редактор системы; для этого нужно нажать клавишу Esc. Курсор редактора при этом исчезнет, а курсор главного меню станет указывать на команду Edit. Задайте теперь команду Run и пронаблюдайте за двумя появившимися во время трансляции программы строками в окне сообщений Message и за результатом работы программы в окне диалога Dialog.

Первая строка в окне сообщений (Compiling WELCOME.PRO) указывает на то, что началась трансляция программы WELCOME.PRO. Трансляция задается автоматически при задании команды Run, т. е. нет необходимости прибегать к помощи специальной команды - Compile.

Турбо-Пролог позволяет адресовать результат трансляции либо на диск, либо в оперативную память. При задании Run программа транслируется в оперативную память. Вторая строка в окне сообщений сигнализирует о трансляции предиката hello.

Сохранение программного файла.

Для того, чтобы записать на диск программу и таким образом сохранить ее, необходимо выйти из редактора (если вы находитесь в режиме редактирования),

нажав клавишу Esc, а затем выбрать команду Files и подкоманду Save во вновь появившемся меню (либо нажав S, либо используя стрелки и клавишу Enter). В результате этих действий на экране возникнет небольшое окно, в котором будет высвечено либо заданное по умолчанию имя файла (как, например, WORK.PRO), либо то имя, которое вы присвоили файлу сами. Имя файла можно оставить без изменений, а можно и отредактировать. В нашем случае следует ввести имя WELCOME.PRO, а затем нажать Enter.

Просмотр каталога директории.

Для того чтобы просмотреть каталог файлов какой-либо директории, необходимо выбрать в главном меню команду Files и подкоманду Directory в появившемся меню команды. На экране возникнет окно, в котором перечисляются все файлы текущей директории .PRO. Если вы хотите увидеть каталог другой директории, то следует ввести путь доступа к этой директории, а затем нажать клавишу Enter. В ответ система попросит задать маску интересующих вас файлов (File mask). По умолчанию стоит маска *.PRO.

Загрузка и редактирование программного файла.

Турбо-Пролог обладает очень мощным экранным редактором, оснащенным большим количеством средств, облегчающих работу программиста. В этом состоит важное отличие Турбо-Пролога от других реализаций Пролога: некоторые из них вообще не обладают встроенными редакторами и приходится выходить из них каждый раз, как только возникает необходимость внести в программу хоть малейшее изменение.

При помощи команд работы с фрагментами можно проделать несколько полезных операций редактирования. Эти команды выделяют кусок текста программы (фрагмент) с целью выполнения одного из трех действий:

- перемещения фрагмента в другое место программы;
- копирования фрагмента;
- удаления фрагмента.

Для того чтобы попрактиковаться в использовании этих команд, войдите в режим редактирования Edit главного меню. Убедитесь в том, что программа, с

которой вы до этого работали, сохранена на внешнем носителе. Затем, чтобы иметь некий текст, с которым вы будете работать, напечатайте на экране программу «Приглашение пользователя»:

```
predicates
test
goal
test.
clauses
test:-clearwindow,
write("please write your name"),
nl,nl,nl,
    readln(ThisName),nl,
    write ("welcome ",ThisName, "!").
```

Теперь посмотрите, как легко пометить этот текст как выделенный фрагмент. Поместите курсор в начало строки, содержащей clauses, и нажмите комбинацию Ctrl-KB (для этого необходимо нажать К при нажатой клавише Ctrl, затем В, причем при нажатии В уже не важно, остается нажатой Ctrl или нет). Поместите курсор в конец строки с write ("welcome ",ThisName, "!") и отметьте конец фрагмента нажатием комбинации Ctrl-KK. Теперь весь отмеченный Вами фрагмент будет выделен на экране. В зависимости от типа используемого Вами монитора выделение будет осуществляться либо при помощи контрастного цвета, либо при помощи повышенной яркости.

Выделенный фрагмент можно скопировать в другое место файла. Прежде всего нажмите клавишу Enter несколько раз подряд, чтобы иметь рабочее пространство из нескольких пустых строк (в противном случае копия фрагмента очутится за пределами окна редактора). Затем поместите курсор двумя строками ниже выделенного фрагмента и нажмите комбинацию Ctrl-KC - фрагмент целиком скопируется в то место, куда был помещен курсор. Повторите комбинацию для

получения по крайней мере еще одного экземпляра текста. Заметьте, что при копировании выделение фрагмента перемещается вместе с самим фрагментом.

Выделение исчезнет, если нажать комбинацию Ctrl-KH. Действие этой команды не зависит от расположения курсора; нет необходимости, чтобы он находился в пределах отмеченного фрагмента.

Теперь можно попытаться удалить одну из копий текста. Подгоните курсор в начало строки goal и отметьте начало фрагмента при помощи Ctrl-KB. Поместите курсор в начало строки, следующей за строкой test , и отметьте конец фрагмента при помощи Ctrl-KK. Удалите фрагмент при помощи Ctrl-KY.

Трассировка программы.

Если программа компилируется, то есть возможность ее отрассировать, т.е. пошагово выполнить. Это можно сделать следующим образом:

a) выбрать Compiler directives;

b) далее выбрать пункт Trace;

c) в окошке появятся пункты меню:

   1) Trace обычная трассировка;

   2) Short Trace укороченная трассировка;

   3) Off выключить трассировку;

d) далее возвратиться в основное меню;

e) выбрать пункт Run, запустить программу;

f) указать цель в окне «Цель», если это требуется;

g) нажатием клавиши F10 осуществить пошаговое выполнение программы (процесс трассировки отображается в окне Trace);

h) Прервать трассировку можно нажатием клавиши Esc.

По окончании выполнения программы в окне «Trace» появится результат выполнения.

Чтобы отключить трассировку, нужно повторить шаги a-e, выбрав в пятом пункте Off.

Оттрассируйте программу «Приглашение пользователя» и проследите за окном Trace.

1.3 Примерный перечень вопросов

1) Какие пункты меню отображаются в среде Turbo Prolog?
2) Какие окна отображаются в среде Turbo Prolog?
3) Как происходит сохранение программного файла?
4) Как просмотреть каталог директорий?
5) Как происходит загрузка программного файла?
6) Как осуществляется копирование, перемещение, удаление и выделение текста в программном файле?
7) Этапы трассировки программного файла.

2 Лабораторная работа №2. Основные понятия языка Turbo Prolog. Организация запросов. Использование баз знаний

Лабораторная работа №2 предназначена для изучения основных понятий языка Turbo Prolog, представлений баз знаний и организации простых и сложных запросов.

Разработать программы, в соответствии с заданиями в пункте 2.4.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 2.3 и 2.4, а также составление отчета, в соответствии с пунктом 2.2. Примерный перечень вопросов приведен в пункте 2.5.

На выполнение и защиту лабораторной работы отводится 4 академических часа.

2.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);
2) написать программу для каждого задания в среде программирования Turbo Prolog;
3) распечатать текст и результат программы в отчет;
4) оформить отчет по лабораторной работе;
5) защитить лабораторную работу.

2.2 Содержание отчета

1) титульный лист;
2) цель работы;
3) формулировка каждого задания;
4) текст программы для каждого задания с комментариями;
5) результат выполнения программы.

2.3 Краткие теоретические сведения

2.3.1 Теоретические принципы Пролога

Математическую основу Пролога составляет исчисления предикатов первого порядка (ИППП). Предикат – это логическая формула от одного или нескольких аргументов. Можно сказать, что предикат – это функция, отображающая множество произвольной природы в множество {ложно, истинно}. Обозначаются предикаты F(x), F(x, y), F(x, y, z) и т. д.

Одноместный предикат F(x), определенный на предметной области M, является истинным, если у объекта x есть свойство F, и ложным – если этого

свойства нет. Двухместный предикат F(x, y) является истинным, если объекты x и y находятся в отношении F.

Многоместный предикат F($x_1, x_2, x_3 \ldots x_n$) задает отношение между элементами $x_1, x_2, x_3 \ldots x_n$ и интерпретируется как обозначение высказывания: «Элементы $x_1, x_2, x_3 \ldots x_n$ находятся между собой в отношении F».

Пример –

хищник (X)

супруги (X,Y)

Одноместный предикат при подстановке в него значения переменной становится «нульместным», т.е. высказыванием-предложением, которое является истинным или ложным:

хищник (тигр).

В этом примере приведено высказывание, которое является истинным предложением.

2.3.2 Основные синтаксические конструкции

Пролог – программы состоят из термов.

Терм - это либо *константа*, либо *переменная*, либо *структура*.

Примечание – Отметим, что понятие терма в Прологе отличается от этого понятия в ИППП.

Константа - это либо *атом*, либо *число*.

Атомы бывают трех типов:

– последовательность букв, цифр и знака «подчёркивание», обязательно начинающаяся со строчной буквы (маленькой буквы);

– последовательность спецзнаков ":-", "?-", "=", ">=", "--" и др.;

– заключённая в апострофы (одинарные кавычки) последовательность любых символов.

В следующем примере приведены несколько атомов

```
abcD, a_gear, '17-256', xyz100
```

Любая система программирования ПРОЛОГ обеспечивает работу с целыми и действительными числами.

Константы представляют собой самоопределённые термы, т.е. их «значением» является их графическое представление (последовательность символов или число). Константы в Прологе используются для обозначения (именования) конкретных объектов предметной области и конкретных отношений между ними.

Переменная – последовательность букв, цифр и знака «подчеркивание», обязательно начинающаяся с прописной (большой) буквы или знака «подчеркивание». Среди переменных выделена переменная "_" (один знак подчеркивания), называемая анонимной. Она используется в ситуациях, когда нас не интересует её значение.

Структура представляется на языке Пролог с помощью указания её функтора и компонент в следующем виде:

Имя структуры = функтор(компонента-1, ..., компонента-N),

где в качестве функтора должен выступать атом, а компонентой может быть любой терм (в том числе и структура).

Если объекты структуры относятся к одному и тому же типу доменов, то такая структура называется однодоменной. Если объекты структуры относятся к разным типам доменов – многодоменная.

Объекты могут быть константами, переменными или структурами. Для того, чтобы объединить объекты в структуры, надо выбрать функтор. Каждый функтор определяется двумя параметрами:

– именем, синтаксис которого совпадает с именем предиката;

– арностью – т. е. числом аргументов.

Рассмотрим в структуру date:

```
date (1, september, 2002),
```

где date – функтор, все аргументы данной многодоменной структуры являются константами.

```
date (Day, september,2002)
```

В этом примере date представляет функтор многодоменной структуры, аргументы которой являются переменные и константы.

```
likes ('Katja', fruits(banana, apples, oranges)),
```

А в этом примере likes – главный функтор, аргументы данной многодоменной структуры – константа и структура.

Рассмотренную структуру в программе можно описать следующим образом:

```
domains
personal_liking = fruits(type1,type2, type3)
type1, type2, type3 = symbol
predicates
likes(symbol, personal_liking)
```

Чтобы легче понять структуры, их обычно представляют в виде дерева, в котором каждому функтору соответствует вершина, а компонентам соответствуют

ветви дерева, в соответствии с рисунками 3 и 4. Каждая ветвь может указывать на другую структуру, так что можем иметь структуры внутри структур.

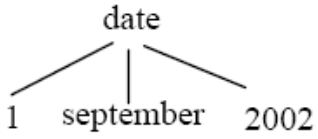


Рисунок 3 – Представление структуры date в виде дерева

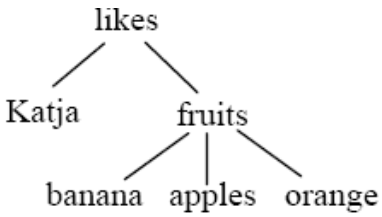


Рисунок 4 – Представление структуры likes в виде дерева

2.3.3 Утверждения

Программа на Прологе есть совокупность утверждений. Утверждения состоят из целей и хранятся в базе данных Пролога. Таким образом, база данных Пролога может рассматриваться как программа на Прологе. В конце утверждения ставится точка ".". Иногда утверждение называется предложением. Основная операция Пролога - доказательство целей, входящих в утверждение.

Предложения бывают двух видов: факты, правила.

Прежде чем приступить к детальному описанию предложений, нам нужно познакомиться нормальной формой Бэкуса-Наура (БНФ), разработанной в 1960 Джоном Бэкусом и Питером Науром и используемой для формального описания синтаксиса языков программирования. Впервые БНФ была применена Питером Науром при записи синтаксиса языка Алгол-60.

При описании синтаксиса конструкций используются следующие обозначения:

Символ «::=» читается как «по определению» («это», «есть»). Слева от разделителя располагается объясняемое понятие, справа – конструкция, разъясняющая его.

Пример –

<Имя> ::= <Идентификатор>

В угловые скобки заключается часть выражения, которая используется для обозначения синтаксической конструкции языка, в частности объясняемое понятие. В приведенном выше примере это <Имя> и <Идентификатор>.

Символ «|» означает в нотации БНФ «или», он применяется для разделения различных альтернативных растолкований определяемого понятия. А теперь перейдем к описанию фактов.

Определение 1. Факт – это утверждение, которое всегда истинно.

Если воспользоваться нормальной формой Бэкуса-Науэра, то предикат можно определить следующим образом:

<Предикат>::=<Имя> | <Имя>(<аргумент>[,<аргумент>]),*

Примечание – математической логике отношения принято называть предикатами.

Из определения видно, что предикат состоит либо только из имени, либо из имени и следующей за ним последовательности аргументов, заключенной в скобки.

Аргументом или параметром предиката может быть константа, переменная или составной объект. Число аргументов предиката называется его арностью или местностью.

Приведем пример двухаргументного факта, который определяет отношение «мама» между двумя объектами «Наташа» и «Даша»:

```
mother("Natasha", "Dasha").
```

Некоторые предикаты уже известны системе, они называются стандартными или встроенными.

В Турбо Прологе предложения с одним и тем же предикатом в заголовке должны идти одно за другим. Такая совокупность предложений называется процедурой.

Определение 2. Правило – это предложение, истинность которого зависит от истинности одного или нескольких предложений. Обычно правило содержит несколько хвостовых целей, которые должны быть истинными для того, чтобы правило было истинным.

В нотации БНФ правило будет иметь вид:

<Правило>::=<предикат>:-<предикат>[,<предикат>].*

В следующем примере приведено правило «бабушка»: Соответствующие правила будут иметь вид:

```
grandmother (X,Y):-
   mother (X,Z), mother (Z,Y).
grandmother (X,Y):-
      mother (X,Z), father (Z,Y).
```

Из этого правила видно, что бабушка человека - это мама его мамы или мама его папы.

Символ «:-» означает «если», и вместо него можно писать if. Символ «,» - это логическая связка «и» или конъюнкция, вместо него можно писать and.

Первое правило сообщает, что X является бабушкой Y, если существует такой Z, что X является мамой Z, а Z - мамой Y. Второе правило сообщает, что X является бабушкой Y, если существует такой Z, что X является мамой Z, а Z - папой Y. В данном примере X, Y и Z - это переменные.

Переменные могут быть свободными или связанными.

Свободная переменная - это переменная, которая еще не получила значения. Она не равняется ни нулю, ни пробелу; у нее вообще нет никакого значения. Такие переменные еще называют неконкретизированными.

Третьим специфическим видом предложений Пролога можно считать вопросы.

Определение 3. Вопрос – это утверждение, которое используется для выполнимости некоторого отношения между объектами.

Примечание – Иногда вопрос называют запросом.

Вопрос состоит только из тела и может быть выражен с помощью БНФ в виде:

*<Вопрос>::=<Предикат>[,<Предикат>]**

Система рассматривает вопрос как цель, к которой надо стремиться. Ответ на вопрос может оказаться положительным или отрицательным, в зависимости от того, может ли быть достигнута соответствующая цель.

Программа на Прологе может содержать вопрос в программе (так называемая внутренняя цель). Если программа содержит внутреннюю цель, то после запуска программы на выполнение система проверяет достижимость заданной цели.

Если внутренней цели в программе нет, то после запуска программы система выдает приглашение вводить вопросы в диалоговом режиме (внешняя цель). Программа, компилируемая в исполняемый файл, обязательно должна иметь внутреннюю цель.

Если цель достигнута, система отвечает, что у нее есть информация, позволяющая сделать вывод об истинности вопроса («Yes»). При этом если в вопросе содержатся переменные, то система либо выдает их значения, приводящие к решению, если решение существует, либо сообщает, что решений нет («No solution»). Если достичь цели не удалось, система ответит, что у нее нет положительного ответа («No»). Следует заметить, что ответ «No» на вопрос не всегда означает, что отношение, о котором был задан вопрос, не выполняется.

Система может дать такой ответ и в том случае, когда у нее просто нет информации, позволяющей положительно ответить на вопрос.

Можно сказать, что утверждение - это правило, а факт или вопрос - это его частный случай.

Пример – Пусть в программе заданы следующие отношения:

```
mother("Natasha","Dasha").
mother("Dasha","Masha").
```

Ниже представлены запросы к нашей базе фактов (левая колонка символов) и ответы на них (правая колонка):

```
mother("Natasha","Dasha").    Yes
mother("Natasha","Masha").    No

mother("Natasha",X).          X=Даша
                              1 Solution

mother (X,Y)                  X=Наташа
                              Y=Даша
                              X=Даша
                              Y=Маша
                              2 solutions

mother (X,_)                  X=Наташа
                              X=Даша
                              2 solutions

mother (_,_)                  Yes
```

В Turbo Prolog существует возможность создавать сложные запросы к базам данным, чтобы получить полную информацию. Ниже приведена база данных:

```
likes (“ Mary”, “Food”).
likes (“ Mary”, “Vine”).
likes (“ Jon”, “ Vine ”).
likes (“ Jon”, “ Mary ”).
```

Задаем вопрос: «Нравится ли Джон Мэри и нравится ли Мэри Джон?». Чтобы задать такой вопрос, мы используем конъюнкцию запросов, как это показано в следующем примере:

```
likes (" Jon", " Mary "), likes ("Mary", " Jon ")
```

2.4 Задание на лабораторную работу

1) Разработать программу, определяющую область семейных отношений. Базовые отношения, такие как отец, мать, пол мужской или женский, описываются в разделе clauses фактами. Более сложные отношения, такие как сестра, брат, бабушка, дедушка, дядя, тетя, племянник, племянница, тесть, теща, свекровь, родитель и т.д. описываются правилами. Задачи формулируются в виде целей: найти всех родственников данного лица, всех бабушек и т.д..

2) Дана база данных «Рождение и хобби друзей»:

рождение (иванова,лена,22,июнь, 1971).

рождение (петров,сергей,25,октябрь,1973).

рождение (сидорова, оля, 1 ,декабрь, 1974).

любит (иванова, лена, книги).

любит (иванова, лена, танцы).

любит (петров, Сергей, видео).

любит (сидорова, оля, кино).

Сформулировать вопросы на Прологе:

- Кто родился в 1971 году?
- Кто родился в октябре?
- Кто любит книги?
- Кто любит и книги и танцы?

3) Дана база данных «Колобок»:

ушел (колобок, дедушка).

ушел (колобок, бабушка).

ушел (колобок, заяц).

ушел (колобок, волк).

ушел (колобок, медведь).

не_ушел (колобок, лиса).

Сформулировать вопросы на Прологе:

- Ушел колобок от бабушки?
- Кто ушел от волка?
- От кого не ушел колобок?
- Кто не ушел от лисы?
- Кто ушел от волка и от бабушки?
- Какой следует задать вопрос, чтобы узнать всех персонажей сказки?

4) Построить базу данных «Важнейшие события Древнего Мира» на основе установленных фактов, произошедших с 31 по 6 век до нашей эры. Каждый факт приводить в виде событие(X,Y,Z), где X — название государства, где произошло событие,Y — в каком веке произошло событие, Z — какое произошло событие.

В 31-м веке до нашей эры возникли первые города-государства. Единое государство в Египте образовалось в 30 веке до нашей эры. В 27 веке до нашей эры в Индии появились первые древнейшие города, а в Египте построена пирамида Хеопса. Первые греческие государства появились в 18 веке до нашей эры. В этом же веке в Египте произошло крупное восстание бедняков и рабов. В 15 веке до нашей эры появились первые государства в Китае. Тутмос III правил в Египте в 15 веке до нашей эры. Греция вела троянскую войну в 13 веке до нашей эры. Вторжение борийских племен в Грецию произошло в 11 веке до нашей эры. В 8 веке до нашей эры был основан город Рим. Олимпийские игры стали проводиться в Греции в 8 веке до нашей эры. В 6 веке до нашей эры в Риме была установлена республика, а в Греции произошли реформы Солона. В этом же веке персы взяли Вавилон в Междуречье и завоевали Египет. Продумать запросы, которые можно задать к этой базе данных.

5) Данные о крупных реках России сведены в таблицу:

Название реки	Длина, км	Годовой сток, км	Площадь бассейна, тыс. км^2	Истоки	Куда впадает
Амур	416	50	1855	Яблоневый хребет	Татарский пролив
Лена	400	88	2490	Байкальский хребет	Море Лаптевых
Обь	4070	400	2990	Предгорья Алтая	Карское море
Иртыш	248	23	1643	Китай	Обь
Енисей	487	600	2580	Восточный Саян	Карское море
Волга	530	55	1360	Валдайская возвышенность	Каспийское море
Колыма	129	44	643	Хребет Черского	Восточно — сибирское море
Урал	428	54	231	Южный Урал	Каспийское море
Дон	200	45	504	Средне-русская возвышенность	Азовское море
Кама	805	30	507	Верхне — Камская возвышенность	Волга
Печора	809	30	322	Северный Урал	Баренцево море
Ангара	779	62	1039	Байкал	Енисей
Селенга	1024	4	447	Монголия	Байкал
Кубань	70	1	58	Кавказ	Азовское море
Нева	4	23	281	Ладожское озеро	Балтийское море

Составить базу данных и ответить на следующие вопросы:

- Определить реки, впадающие в Азовское море.
- Определить реки, исток которых находится на Валдайской возвышенности.
- Какие реки впадают в Каспийское море?
- Какие реки короче Камы?
- Какие реки длиннее Иртыша?
- Как задать вопрос, определяющий все данные о какой-либо реке?

Примечание – В запросах можно использовать сравнения между числовыми константами.

2.5 Примерный перечень вопросов

1) Каковы теоретические принципы языка Пролог?
2) Что такое терм. Какие термы в Прологе используются?
3) Этапы программирования на языке Пролог.
4) Что такое факты и какие особенности их использования в языке Пролог?
5) Что такое вопросы и какие особенности их использования в языке Пролог?
6) Что такое правила и какие особенности их использования в языке Пролог?
7) Как используется конъюнкция в вопросах? Как происходит сопоставление в Пролог?

3 Лабораторная работа №3. Структура программы на языке Prolog. Организация запросов. Составные объекты

Лабораторная работа №3 предназначена для изучения структуры программы на языке Prolog, получения навыков в использовании составных объектов, в составлении правил и в организации сложных и простых запросов.

Разработать программы, в соответствии с заданиями в пункте 3.4.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 3.3 и 3.4, а также составление отчета, в соответствии с пунктом 3.2. Примерный перечень вопросов приведен в пункте 3.5.

На выполнение и защиту лабораторной работы отводится 4 академических часа.

3.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);

2) написать программу для каждого задания в среде программирования Turbo Prolog;

3) распечатать текст и результат программы в отчет;

4) оформить отчет по лабораторной работе;

5) защитить лабораторную работу.

3.2 Содержание отчета

1) титульный лист;

2) цель работы;

3) формулировка каждого задания;

4) текст программы для каждого задания с комментариями;

5) результат выполнения программы.

3.3 Краткие теоретические сведения

Программа на Турбо Прологе состоит из следующих семи разделов:

- *директивы компилятора*;
- **CONSTANTS** - раздел описания *констант*;
- **DOMAINS** - раздел описания *доменов*;
- **DATABASE** - раздел описания предикатов внутренней базы данных;
- **PREDICATES** - раздел описания предикатов;
- **CLAUSES** - раздел описания *предложений*;
- **GOAL** - раздел описания *внутренней цели*.

В программе не обязательно должны быть все эти разделы.

3.3.1 Директивы компилятора

В самом начале программы можно расположить одну или несколько директив компилятора, которые дают компилятору дополнительные инструкции по обработке программы.

Директива trace применяется при отладке программы для трассирования. Трассировка позволяет пользователю наблюдать за ходом выполнения программы. Если после ключевого слова trace указаны имена предикатов через запятую, то трассировка идет только по этим предикатам. В противном случае - по всем предикатам программы:

– во время исполнения программы при включенной трассировке в специальном окне трассировки будет отображаться следующая информация;

– после слова «CALL» будет указано имя выполняемого предиката (текущая подцель) и его параметры;

– после слова «FAIL» будет выводиться имя текущей подцели, которая не была достигнута;

– после слова «RETURN» будет выводиться результат вычисления текущей подцели, в случае успеха. При этом если у подцели есть еще альтернативы, к которым возможен возврат, то перед именем предиката высвечивается звездочка ("*");

– слово «REDO» перед именем предиката указывает на то, что произошел возврат и происходит вычисление альтернативного решения.

Переход от подцели к подцели вызывается нажатием F10.

Директива nowarnings используется для подавления предупреждения системы о том, что какая-то переменная встречается в предложении только один раз. Эту директиву стоит использовать только в хорошо отлаженных программах.

С помощью директивы include при компиляции в исходный текст можно вставить содержимое некоторого файла.

Заметим, что многие директивы компилятора могут быть не только расположены в тексте программы, но и установлены в меню среды разработки

Турбо Пролога (Options→Compiler Directives). Значение директивы компилятора, указанное в тексте программы, имеет более высокий приоритет, чем значение, установленное в меню.

3.3.2 Раздел описания констант

Раздел, озаглавленный зарезервированным словом CONSTANTS, предназначен для описания констант. Объявление константы имеет вид:

<имя константы>=<значение>

Имя константы должно состоять из английских букв, цифр и знака подчеркивания, причем не может начинаться с цифры. Каждое определение константы должно размещаться в отдельной строке.

Пример – Описание констант

```
pi=3.14
bgi_path="c:\\prolog\\bgi"
```

В разделе описания констант можно использовать в качестве первого символа имени константы прописные символы, потому что в этом разделе прописные и строчные символы не различаются. Однако при использовании констант в разделе описания предложений нужно задействовать в качестве первого символа имени константы только строчные символы, чтобы Пролог-система не восприняла константу как переменную. Разделов описания констант может быть несколько, но каждая *константа* должна быть определена до ее первого использования.

3.3.3 Раздел описания доменов

Раздел описания доменов является аналогом раздела описания типов в обычных императивных языках программирования и начинается с ключевого слова DOMAINS.В Турбо Прологе имеются стандартные домены, которые не нужно указывать в разделе описания доменов. Основные стандартные домены - это:

- integer - целое число (из промежутка -32768...32767);
- real - действительное число (лежащее между ±1e-307...±1e308);
- char - символ, заключенный в одиночные апострофы;
- string - последовательность символов, заключенная в двойные кавычки;
- symbol - символическая константа (начинающаяся со строчной буквы последовательность букв латинского алфавита, цифр и знаков подчеркивания или последовательность любых символов, заключенная в кавычки). Этот домен соответствует понятию атома**;**
- file - файл.

В разделе описания доменов объявляются любые нестандартные домены, используемые в качестве аргументов предикатов. Объявление домена имеет следующий вид:

<имя домена>=<определение домена>

или

file=<имя файлового домена1>;...;<имя файлового доменаN>

Из доменов можно конструировать составные или структурные домены (структуры). Структура описывается следующим образом:

<имя структуры>=<имя функтора>(<имя домена первой компоненты>,...,<имя домена последней компоненты>)

3.3.4 Раздел описания предикатов внутренней базы данных

Факты, описанные в разделе clauses, можно рассматривать, как статическую базу данных (БД). Эти факты являются частью кода программы и не могут быть оперативно изменены. Для создания динамической базы данных в ПРОЛОГе предусмотрен специальный раздел database. Предикаты в этом разделе могут иметь такую же форму представления, что и в статической части ПРОЛОГ-программы, но должны иметь другое имя.

Пример – Описание динамической базы данных:

```
DOMAINS
name = symbol
rost, ves = integer
DATABASE
dplayer(name, rost, ves)
PREDICATES
player(name, rost, ves)
CLAUSES
player("Михайлов", 180, 87)
player("Петров", 187, 93)
player("Харламов", 177, 80)
```

В примере динамическая база данных имеет имя dplayer.

3.3.5 Раздел описания предикатов

В разделе, озаглавленном зарезервированным словом PREDICATES, содержатся описания определяемых пользователем предикатов. В традиционных языках программирования подобными разделами являются разделы описания заголовков процедур и функций. Описание n-местного предиката имеет следующий вид:

<имя предиката>(<имя домена первого аргумента>,...,
<имя домена n-го аргумента>).

Домены аргументов должны быть либо стандартными, либо объявленными в разделе описания доменов.

Пример – Описание предиката:

```
PREDICATES
mother(string,string)
```

Это описание означает, что у предиката два аргумента, причем оба строкового типа.

Один предикат может иметь несколько описаний. Это используется, когда нам нужно, чтобы предикат работал с аргументами различной природы.

Кроме того, при описании предиката можно указать, будет он детерминированным или недетерминированным. Детерминированный предикат возвращает только одно решение, а недетерминированный предикат при помощи поиска с возвратом может давать много решений. Детерминированные предикаты менее требовательны к оперативной памяти и выполняются быстрее. Отсечение позволяет превращать недетерминированные предикаты в детерминированные. Для того чтобы указать, что предикат является детерминированным (недетерминированным), нужно перед его именем поместить зарезервированное слово determ (nondeterm). Если ни determ, ни nondeterm при описании предиката не использовались, то, по умолчанию, предикат считается детерминированным.

В Турбо Прологе есть так называемые стандартные (встроенные) предикаты, которые не нужно описывать в разделе описания предикатов PREDICATES.

Примеры встроенных предикатов:

– nl вызывает перевод строки;
– write применяется для вывода информации на экран;

– «=» – операция унификации;

– not логическое отрицание.

Встроенный предикат не может являться головой правила или появляться в факте.

3.3.6 Раздел описания предложений

Этот раздел можно считать основным разделом программы, потому что именно в нем содержатся факты и правила, реализующие пользовательские предикаты. Все предикаты, которые применяются в этом разделе и не являются стандартными предикатами, должны быть описаны в разделе описания предикатов или в разделе описания предикатов базы данных. Начинается этот раздел со служебного слова CLAUSES. Предложения, у которых в заголовке указан один и тот же предикат, должны идти друг за другом. Такой набор предложений называется процедурой.

3.3.7 Раздел описания внутренней цели

С зарезервированного слова GOAL начинается раздел описания внутренней цели программы. Если этот раздел отсутствует, то после запуска программы Пролог-система выдает приглашение вводить вопросы в диалоговом режиме (внешняя цель). При выполнении внешней цели Пролог-система ищет все решения, выводя все возможные значения для переменных, участвующих в вопросе. Если же выполняется внутренняя цель, то осуществляется поиск только первого решения, а для получения всех решений нужно предпринимать дополнительные действия.

Программа, компилируемая в исполняемый файл, который можно запускать независимо от среды разработки, обязательно должна иметь внутреннюю цель. Внешнюю цель обычно применяют на этапе отладки программы.

Для запуска программы нажимаем Alt+R (Run). Так как раздела описания внутренней цели в нашей программе не было, Пролог система выведет приглашение на ввод внешней цели («GOAL:»). Вводим вопросы, наблюдаем результаты. Повтор предыдущей цели F8.

Пролог - не самый лучший инструмент для выполнения большого объема вычислений, в нем имеются стандартные средства для реализации обычных вычислений. Стандартные средства для организаций вычислений приведены в приложении Б.

Для описания любых операций арифметики можно также использовать собственные предикаты.

Пример – Описание арифметических операций:

```
PREDICATES
add(integer,integer)
fadd(real,real)
maximum(real,real,real)
CLAUSES
add(X,Y):-Z=X+Y,write("Sum= ",Z),nl.
fadd(X,Y):-Z=X+Y,write("FSum= ",Z),nl.
maximum(X,X,X).
maximum(X,Y,X):- X>Y.
maximum(X,Y,Y):- X<Y.
```

3.4 Задания на лабораторную работу

1) Для приведенной ниже базы данных определите отношения: выше (X,Y) высокий (X), средний (X), низкий (X).

Условимся считать, что мужчина m, и женщина f, имеют высокий рост, если он превышает 180 и 175 см соответственно, и низкий рост, если он не превышает 170 и 160 см.

рост (Билл, m, 180).

рост (Энди, m, 154).

рост (Джим, m, 187).

рост (Пол, m, 175).

рост (Джон, m, 166).

рост (Алекс, m, 176).

рост (Мери, f, 173).

рост (Анна, f, 161).

рост (Джоан, f, 168).

рост (Кэтрин, f, 178).

2) Сформировать базу знаний «Квартет» из следующих фактов и правил:

Мартышка играет на скрипке. Осел играет на альте. Козел играет на виолончели. Мишка играет на контрабасе. Четверо музыкантов X, Y, Z и W могут образовать квартет, если один из них играет на скрипке, другой — на альте, третий — на виолончели и четвертый — на контрабасе.

Ответить на вопросы:

- Кто играет на альте?
- На чем играет мартышка?
- Образуют ли квартет Мартышка. Осел, Козел и Мишка?
- Кто из музыкантов данной базы знаний может образовать квартет?

3) Рассмотрим следующие таблицы оборудования и его поставщиков на склад запасных частей для автомобилей:

Поставщик	Код	Имя	Род деятельности	Город
	001	Джон	Фабрикант	Афины
	002	Энди	Посредник	Патрас
	010	Джон	Посредник	Салоники
	110	Ник	Импортер	Пирей

Товар	Артикул	Наименование	Модель	Вес
	003	Масло	30	300
	004	Шины	157/75	2000
	005	Лампы	RAAI	10
	013	Масло	60	500

Поставка	Код	Артикул	Количество
	001	005	150
	002	003	200
	010	004	030
	110	013	250

Представьте информацию, содержащуюся в таблицах, в виде базы данных ПРОЛОГа, используя только двухместные предикаты и запишите в виде хорновских дизъюнктов следующие запросы:

– Как зовут поставщика масла?

– В каком городе проживает поставщик шин, и в каком городе проживает поставщик ламп?

– Что поставляет Джон?

– Какой поставщик масла обосновался в Патрасе и поставляет менее 400 тонн продукта?

4) Построить базу знаний «Рабочая смена»:

Мария работает в дневную смену. Сергей работает в вечернюю смену. Борис работает в вечернюю смену. Валентина работает в вечернюю смену. Два служащих знают друг друга, если они работают в одну смену.

Определить:

- Знает ли Сергей Бориса?
- Кого знает Валентина?
- Кого знает Мария?

5) Построить базу знаний «Животные»:

содержит(иванов, животное(васька, кот, черный)).

содержит(петров, животное(бусик, кот, серый)).

содержит(сидоров,животное(резвый,собака, серый)).

содержит(иванов, животное(барс, кот, серый)).

содержит(иванов, животное(белка, кошка, белый)).

Добавить правило:

цвет_животного(X,Y), где X — кличка, Y — цвет животного.

Сформулировать вопросы, позволяющие:

- определить клички всех животных серого цвета;
- найти хозяев, у которых животные не серого цвета;
- определить кличку животного белого цвета;
- каких животных держит Иванов?

6) Составить базу знаний «Знакомства» из следующих фактов и правил:

Мери прелестна. Джон добрый. Джон мужественный. Джон сильный. Некто счастлив, если богатый или нравится женщинам. Мужчина нравится женщине, если женщина нравится мужчине и он добрый, либо мужчина добрый и сильный. Мужчине нравится женщина, если она прелестна.

Сформулировать вопрос: счастлив ли Джон?

Найти мужчин, которые могут нравиться женщинам.

7) Построить базу знаний и сформулировать к ней вопросы, основываясь на следующих утверждениях:

Резвый — это собака. Рекс — это собака. Белка — это кошка. Быстрая — это лошадь. Резвый — черная. Белка — белая. Рекс — рыжая. Быстрая — белая. Домашние животные — это собака или кошка. Животные — это либо лошадь, либо домашние животные. Том владеет тем. кто собака и не черного цвета. Кейт владеет тем, кто либо черного цвета, либо лошадь.

8) Построить базу знаний.

Муська — коричневая кошка, Стрелка — черная кошка, Мурка — рыжая кошка. Рекс. Дружок и Мухтар — собаки. Дружок — рыжая, Мухтар — белая. Все животные, которыми владеют Анатолий и Николай, имееют родословные. Анатолий владеет всеми черными и коричневыми животными, а Николай владеет всеми собаками небелого цвета, которые не являются собственностью Анатолия. Иван владеет Муркой, если Николай не владеет Муськой и если Мухтар не имеет родословной. Рекс — пятнистая собака.

Определить, какие животные не имеют хозяев.

9) Построить базу знаний, отражающую следующие характеристики и родственные связи древнегреческих богов:

Зевс — отец Ареса. Гера — мать Ареса. Арес — отец Гармонии. Афродита — мать Гармонии. Карид — отец Семелы. Зевс — отец Диониса. Семела — мать Диониса. Божества — Зевс, Гера, Арес, Афродита. Гармония — царица. Определить правила: женщина, мужчина, родитель, родители, дедушка, бабушка.

10) Даны результаты сдачи экзаменов для группы студентов математического факультета:

Фамилия	Математический	Геометрия	Алгебра
Антонов	5	5	5
Бобров	5	3	2
Вяткин	5	5	5
Дедов	4	4	4
Емельянова	5	5	5
Кротов	2	3	3
Марьин	5	4	5
Новиков	2	3	2
Подлесный	2	3	3
Полежаев	5	5	5
Соснин	4	4	4

Построить базу знаний о результатах экзаменов, определив в ней следующие правила:

- отличник — это человек, у которого по всем предметам пятерки;
- двоечник — если есть хотя бы одна двойка;-
- математик — если по алгебре, геометрии и матанализу учится на 4 и 5.

Получить ответы на следующие вопросы: Является ли Новиков отличником? Определить всех отличников. Определить всех двоечников. Является ли Соснин математиком? Определить всех неуспевающих по матанализу. Определить всех двоечников.

3.5 Примерный перечень вопросов

1) Какие разделы существуют в программе на языке Пролог?
2) Какие из них обязательны и какие нет?
3) Дайте описание каждого раздела.
4) Какие запросы существуют? Что такое сложные запросы?

4 Лабораторная работа №4. Использование рекурсии в Пролог-программах

Лабораторная работа №4 предназначена для изучения рекурсии в Turbo Prolog и ее разновидностей.

Разработать программы, в соответствии с заданиями в пункте 4.4.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 4.3 и 4.4, а также составление отчета, в соответствии с пунктом 4.2. Примерный перечень вопросов в приведен в пункте 4.5.

На выполнение и защиту лабораторной работы отводится 2 академических часа.

4.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);
2) написать программу для каждого задания в среде программирования Turbo Prolog;
3) распечатать текст и результат программы в отчет;
4) оформить отчет по лабораторной работе;
5) защитить лабораторную работу.

4.2 Содержание отчета

1) титульный лист;
2) цель работы;
3) формулировка каждого задания;
4) текст программы для каждого задания с комментариями;
5) результат выполнения программы.

4.3 Краткие теоретические сведения

Рекурсия – один из фундаментальных приемов программирования на Турбо-Прологе. Она обычно применяется в ситуациях, когда число возможных решений заранее неизвестно, либо, когда обрабатываются структуры данных с произвольным количеством элементов.

Предикат или функция называются рекурсивными, если они ссылаются на самих себя. При этом задача разбивается на части все меньшего и меньшего размера до тех пор, пока они не станут настолько малы, что их решение не будет сводиться к набору из одной или нескольких простейших операций.

Для управления рекурсией используется условие выхода. При рекурсивном вызове новые копии используемых значений переменных помещаются в стек. Стек – область памяти, которая используется для передачи значений переменных между правилами. В стеке данные хранятся до тех пор, пока правило не завершится успешно или нет. В случае отсутствия условия выхода из рекурсии возникает бесконечная рекурсия, непрерывно растет число значений переменных, помещаемых в стек. Возникает переполнение стека и в результате могут быть утерянными необходимые значения переменных.

Обычно рекурсивная программа состоит как минимум из двух частей:

– граничного условия, при котором рекурсия останавливается (базис рекурсии);

– рекурсивного условия, при котором в описание функции или предиката входит и сама функция или предикат (шаг рекурсии).

Базис рекурсии - это предложение, определяющее некую начальную ситуацию или ситуацию в момент прекращения. Как правило, в этом предложении записывается некий простейший случай, при котором ответ получается сразу даже без использования рекурсии. Так, в приведенной выше процедуре, описывающей предикат предок, базисом рекурсии является первое правило, в котором определено, что ближайшими предками человека являются его родители. Это предложение часто содержит условие, при выполнении которого происходит выход из рекурсии или отсечение.

Шаг рекурсии - это правило, в теле которого обязательно содержится, в качестве подцели, вызов определяемого предиката. Если мы хотим избежать зацикливания, определяемый предикат должен вызываться не от тех же параметров, которые указаны в заголовке правила. Параметры должны изменяться на каждом шаге так, чтобы в итоге либо сработал базис рекурсии, либо условие выхода из

рекурсии, размещенное в самом правиле. В общем виде правило, реализующее шаг рекурсии, будет выглядеть так:

<имя определяемого предиката>:-
[<подцели>], (1)
[<условие выхода из рекурсии>], (2)
[<подцели>], (3)
<имя определяемого предиката>, (4)
[<подцели>]. (5)

Данное правило структуры имеет пять компонентов.

Первая компонента – группа предикатов выполняется не влияя на рекурсию (кроме предиката отсечения, который исключает построение стека для хранения значений входных аргументов правила).

Вторая компонента – обычно из одного предиката для проверки условия окончания рекурсии.

Третья компонента – группа предикатов выполняется не влияя на рекурсию.

Четвертая компонента – имя самого рекурсивного правила, вызывая его повторно. При этом системой Турбо-Пролога предусмотрено заполнение стека памяти (если нет отсечения), где хранятся значения переменных каждого нового вхождения рекурсивного правила в само себя (хвост рекурсии).

Пятая компонента – группа предикатов берет значения переменных, помещенные в стек во время выполнения рекурсии, т. е. после завершения рекурсии значения из стека последовательно используются этими предикатами как бы разворачивая ее в обратную сторону.

Так как рекурсивное описание правила содержит в своем теле ссылку на заголовок этого же правила, то связи с этим возможны следующие разновидности рекурсии:

Правая рекурсия:	**Левая рекурсия:**	**Обобщенная рекурсия:**
<имя правила рекурсии>:-	*<имя правила рекурсии>:-*	*<имя правила рекурсии>:-*
<список предикатов>,	*<имя правила рекурсии>,*	*<список предикатов>,*
<предикат условия выхода>,	*<список предикатов>,*	*<предикат условия выхода>,*
<список предикатов>,	*<предикат условия выхода>,*	*<список предикатов>,*
<имя правила рекурсии>.	*<список предикатов>.*	*<имя правила рекурсии>,*
		<список предикатов>.

Создадим предикат, который будет вычислять по натуральному числу его факториал. Эта задача допускает рекурсивное решение на многих языках программирования, а также имеет рекурсивное математическое описание:

1!=1 /*факториал единицы равен единице */

N!=(N-1)!*N /* для того, чтобы вычислить факториал некоторого числа, нужно вычислить факториал числа на единицу меньшего и умножить его на исходное число */

Программа, реализующая функцию факториала числа n! в Прологе, выглядит так:

```
fact(1,1).   /* факториал единицы равен единице */
fact(N,F):-
N>0,
N1=N-1,
fact(N1,F1),/*F1 равен факториалу числа на единицу меньшего исходного числа */
 F=F1*N.      /* факториал исходного числа равен произведению F1 на само число */
```

Программа состоит из двух процедур.

Первая процедура fact(1,1) является базисом рекурсии. Условием выхода из рекурсии является значение переменной N равной 1 (1!=1).

Вторая процедура fact (N,F) называется шагом рекурсии. Подцель N>0 является условием выполнения рекурсии. Переменная F1 является неозначенной, она получает значение равное 1, когда переменная N1 становится равной 1, т.е. успешным становится вызов процедуры fact(1,1). На это рекурсивные вызовы заканчиваются. Начинает выполняться подцель `F=F1*N.` В процессе работы данной процедуры был создан стек, содержащий все промежуточные значения переменной N. Теперь эти значения извлекаются из стека и переменная F становится означенной переменной.

4.4 Задания на лабораторную работу

1) Используя рекурсию, напишите программу, вычисляющую сумму первых N натуральных чисел.

Пример – Цель sum(9,S) должна возвращать S=45, а sum(100,S) — 5050.

Примечание – 55=1 +2+3+4+5+6+7+8+9+10.

2) Измените программу, полученную в задании 1 так, чтобы она вычисляла сумму нечетных натуральных чисел от 1 до указанного нечетного числа.

Пример – Цель sum2(9,S) должна возвращать S=25.

Примечание – 25=1+3+5+7+9.

3) Измените программу, полученную в задании 2 так, чтобы она вычисляла либо сумму нечетных натуральных чисел от 1 до указанного нечетного числа, либо сумму четных натуральных чисел от 2 до указанного четного числа.

Пример – Цель sum3(9,S) должна возвращать S=25, а sum3(10,S) — 30.

Примечание – 25=1+3+5+7+9, 30=2+4+6+8+10.

4) Вычислить произведение двух целых положительных чисел (используя суммирование).

5) Напишите программу, вычисляющую факториал числа N, используя промежуточные значения числа и его факториала.

Пример – Цель fact(5,F) должна возвращать F= 120, а fact(10,F) — 3628800.

6) Вычислить значение n-го члена ряда Фибоначчи:
f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2).

7) Организовать ввод целых положительных чисел и их суммирование до тех пор, пока сумма не превысит некоторого порогового значения. Введенные отрицательные целые числа суммироваться не должны.

4.5 Примерный перечень вопросов

1) Что такое рекурсия?
2) Что такое шаг и базис рекурсии?
3) Из каких компонентов состоит обобщенная рекурсия?
4) Какие виды рекурсии существуют? Как они описываются?
5) Что такое хвостовая рекурсия?
6) Как происходит вычисление рекурсии?

5 Лабораторная работа №5. Реализация списков в Turbo Prolog и выполнение основных операций с ними

Целью лабораторной работы №5 является приобретение навыков в использовании средств Turbo Prolog для работы со списками.

Разработать программы, в соответствии с заданиями в пункте 5.4.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 5.3 и 5.4, а также составление отчета, в соответствии с пунктом 5.2. Примерный перечень вопросов приведен в пункте 5.5.

На выполнение и защиту лабораторной работы отводится 4 академических часа.

5.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);
2) написать программу для каждого задания в среде программирования Turbo Prolog;
3) распечатать текст и результат программы в отчет;
4) оформить отчет по лабораторной работе;
5) защитить лабораторную работу.

5.2 Содержание отчета

1) титульный лист;
2) цель работы;
3) формулировка каждого задания;
4) текст программы для каждого задания с комментариями;
5) результат выполнения программы.

5.3 Краткие теоретические сведения

Список – это упорядоченная последовательность элементов, которая может иметь произвольную длину. Признак упорядоченный указывает на то, что порядок элементов в последовательности является существенным. Элементами списка могут быть любые термы – константы, переменные, структуры, которые включают и другие списки.

Список – это набор объектов одного и того же типа. При записи список заключают в квадратные скобки, а элементы списка разделяют запятыми.

Пример – Представление списков:

[monday, tuesday, wednesday, thursday, friday, saturday, sunday] (1)
["понедельник", "вторник", "среда", "четверг", "пятница", "суббота", "воскресенье"] (2)
[1, 2, 3, 4, 5, 6, 7] (3)
['п', 'в', 'с', 'ч', 'п', 'с', 'в'] (4)
[[1,3,7],[],[5,2,94],[–5,13]] (5)

(1) – список, элементами которого являются английские названия дней недели;

(2) – список, элементами которого являются русские названия дней недели;

(3) – список, элементами которого являются номера дней недели;

(4) – список, элементами которого являются первые символы русских названий дней недели;

(5) – список, элементами которого являются списки целых чисел.

Пустой список записывают как [ ] – открывающая квадратная скобка, за которой следует закрывающая квадратная скобка.

Turbo Prolog позволяет работать как со списком в целом, так и с отдельными элементами списка.

Внутренние унификационные подпрограммы позволяют отделить от списка первый элемент и обрабатывать его отдельно. Данный метод работает вне зависимости от длины списка до тех пор, пока список не будет исчерпан. Этот метод доступа к голове списка называется методом разделения списка на голову и хвост и является основным методом работы со списками. Это позволяет всякий список представить в виде бинарного дерева, в соответствии с рисунком 5.

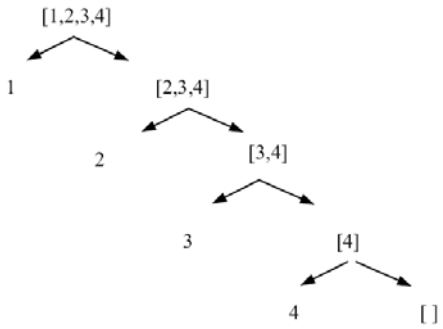


Рисунок 5 – Операция разделения списка на голову и хвост

Дадим рекурсивное определение списка:

Список — это структура данных, определяемая следующим образом:

1. пустой список (`[ ]`) является списком;

2. структура вида `[H|T]` является списком, если `H` — первый элемент списка (или несколько первых элементов списка, перечисленных через запятую), а `T` — список, состоящий из оставшихся элементов исходного списка.

Принято называть `H` головой списка, а `T` — хвостом списка. Заметим, что выбор переменных для обозначения головы и хвоста не случаен. По-английски голова — Head, а хвост — Tail.

В разделе описания доменов списки описываются следующим образом:

*<имя спискового домена>=<имя домена элементов списка>**

Звездочка после имени домена указывает на то, что мы описываем список, состоящий из объектов соответствующего типа.

Пример – Описание списков:

```
listI = integer*   /* список, элементы которого — целые числа */
listR = real*      /* список, состоящий из вещественных чисел */
listC = char*      /* список символов */
lists = string*    /* список, состоящий из строк */
listL = listI*     /* список, элементами которого являются списки целых
                      чисел */
```

В Турбо Прологе, в связи со строгой типизацией, все элементы списка должны принадлежать одному домену. Однако можно разместить в одном списке объекты разной природы, используя домен с соответствующими альтернативами.

Пример – Описание списка с элементами разных доменов:

```
element = i(integer); c(char); s(string)
```

```
listE = element*
```

позволит иметь дело со списками вида:

```
[i(-15), s("Мама"),c('A'),s("мыла"),c('+'),s("раму"),i(48),c('!')]
```

Основной прием построения алгоритмов работы со списками – составление процедуры, состоящей из двух правил: рекурсивного правила работы с головой и хвостом списка и правила работы с пустым списком.

В таблице 1 показан пример, как сопоставляются списки.

Таблица 1 – Сопоставление списков

Список 1	**Список 2**	**Результат присвоения**
[X,Y,Z]	[джону нравится рыба]	X = Джону Y = нравится Z= рыба
[кошка]	[X|Y]	X = кошка Y = []
[X,Y|Z]	[мэри нравится вино]	X = Мэри Y = нравится Z = [вино]
[[этот, Y]|Z]	[[X, заяц]|[находится, здесь]]	X = этот Y = нравится Z = [[находится, здесь]]
[X,Y| Z,W]	{Синтаксически некорректная конструкция списка}	
[золотистый | T]	[золотистый, свет]	T = [свет]
[белая, лошадь]	[лошадь, X]	{сопоставление невозможно}
[белая | Q]	[P | лошадь]	P = белая Q = лошадь

Над списками можно выполнять следующие операции:

- доступ к элементам списка;
- поиск элемента в списке;
- добавление нового элемента в список;
- удаление элемента из списка;
- деление списка на подсписки;
- объединение двух списков.

Сортировка элементов списка по возрастанию и по убыванию.

Рассмотрим на примере операции принадлежности элемента к списку и печать списка.

Пример – Проверка принадлежности элемента к списку:

```
domains
список= integer*
clauses
member( integer, список).
predicates
member(X,[X|_]).
   member(X,[_|Y]):-член(X,Y).
```

Первое определение правила `member(X,[X|_])` читается так: X является элементом списка, если он тождественен с заголовком списка.

Второе определение правила `member(X,[_|Y]):-member (X,Y).` читается так: X является элементом его списка, если он является элементом хвоста Y.

Теперь можно задать вопрос программе:

```
Goal: член(3,[-1,3,4]).
Yes
```

Пример – Печать списка:

```
domains
Элементы_списка = real*
predicates
печать(Элементы_списка)
clauses
печать([]). /*фиксирует достижение конца списка */
печать([Голова|Хвост]):-
write(Голова), nl, печать(Хвост).
```

```
Goal:    печать([-1,3,4])
Ответ:   -1
          3
          4
         Yes
```

Рассмотрим этот процесс подробнее.

Проход I

– Первый вариант правила печать([]) дает неуспех, т. к. `[-1,3,4]` - не пустой список.

– В результате сопоставления цели: печать([-1,5,4]) с головной целью (заголовком) правила:

печать `([Голова|Хвост]):-`

– Переменные Голова и Хвост получают значения:

`Голова=-1`

`Хвост=[3,4]`

– Встроенные предикаты `write(Голова),nl` выводят на печать `-1` и переводят курсор на новую строчку.

– Печать `[Хвост]` активизирует рекурсивный вызов правила печать `([3,4])` для элементов хвоста списка.

Проход II

На втором проходе печатается второй элемент списка 3 и происходит рекурсивный вызов правила печать ([4]).

Проход III

На третьем проходе происходит печать третьего элемента списка: 4 и происходит рекурсивный вызов правила печать ([]). Вариант вызова печать([]) не имеет рекурсивных вызовов, т. к. согласуется с первым определением правила: печать([]) и таким образом рекурсия завершается.

5.4 Задания на лабораторную работу

Реализовать на Прологе следующие операции:

1) создать предикат, который вычисляет длину списка. Предикат будет иметь два аргумента: первый – исходный список, второй – количество символов в списке;

2) создать предикат, который проверяет принадлежность элемента списку. Предикат будет иметь два аргумента: первый — искомое значение, второй — список, в котором производится поиск;

3) создать предикат, который соединяет списки в один. Первые два аргумента предиката будут представлять соединяемые списки, а третий — результат соединения;

4) разработать предикат, позволяющий «обратить» список. Предикат будет иметь два аргумента: первый — исходный список, второй — список, получающийся в результате записи элементов первого аргумента в обратном порядке;

5) создать предикат, который позволит проверить, является ли *список* палиндромом. Палиндромом называется список, который совпадает со своим обращением. Соответственно, у данного предиката будет всего один аргумент (список, который проверяем на «палиндромность»);

6) создать предикат, позволяющий получать элемент списка по его номеру так же, как по номеру можно получать элемент массива в императивных языках программирования. Предикат будет трехаргументный: первый аргумент — исходный список, второй аргумент — номер элемента и третий — элемент списка, указанного в качестве первого аргумента предиката, имеющий номер, указанный в качестве второго аргумента;

7) создать предикат, удаляющий все вхождения заданного значения из списка. Предикат будет зависеть от трех параметров. Первый параметр будет соответствовать удаляемому списку, второй — исходному значению, а третий — результату удаления из первого параметра всех вхождений второго параметра.

5.5 Примерный перечень вопросов

1) Что такое список?
2) Какие существуют объекты списка?
3) Какие операции выполняются над списками?
4) В чем заключается метод разделения списка на голову и хвост?
5) Как происходит компоновка данных в список?

6 Лабораторная работа №6. Реализация строк в Turbo Prolog и выполнение основных операций над ними

Целью лабораторной работы №6 является приобретение навыков в использовании средств Turbo Prolog для работы со строками.

Разработать программы, в соответствии с заданиями в пункте 6.4.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 6.3 и 6.4, а также составление отчета, в соответствии с пунктом 6.2. Примерный перечень вопросов приведен в пункте 6.5.

На выполнение и защиту лабораторной работы отводится 4 академических часа.

6.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);
2) написать программу для каждого задания в среде программирования Turbo Prolog;
3) распечатать текст и результат программы в отчет;
4) оформить отчет по лабораторной работе;
5) защитить лабораторную работу.

6.2 Содержание отчета

1) титульный лист;
2) цель работы;
3) формулировка каждого задания;
4) текст программы для каждого задания с комментариями;
5) результат выполнения программы.

6.3 Краткие теоретические сведения

Под *строкой* в Прологе понимается последовательность символов, заключенная в двойные кавычки.

В Turbo Prolog имеются несколько встроенных предикатов, предназначенных для работы со строками. Рассмотрим их.

Встроенный предикат **str_len**, который предназначен для определения длины строки, т.е. количества символов, входящих в строку. Он имеет два аргумента: первый — строка, второй — количество символов. Имеется три варианта использования данного предиката. Опишем частоупотребляемые варианты его использования.

Первый вариант использования этого предиката, когда первый аргумент связан, а второй свободен. В этом случае во второй аргумент будет помещено количество символов в первом аргументе.

Пример – Использование str_len в первом варианте:

Запрос	Ответ
`str_len ("student", X)`	`X=7`

Второй вариант использования этого предиката, когда оба аргумента связаны. В этом случае предикат будет успешен, если длина первого аргумента будет совпадать со вторым аргументом, и неуспешен в противном случае.

Пример – Использование str_len во втором варианте:

Запрос	Ответ
`str_len ("student", 7)`	`Yes`
`str_len ("anna", 7)`	`No`

Следующий стандартный предикат **concat** предназначен для соединения двух строк. У него три аргумента, каждый строкового типа, по крайней мере, два из трех аргументов должны быть связаны. Итого получаем четыре возможных шаблона или четыре варианта использования этого предиката. Опишем частоупотребляемые варианты его использования.

Первый вариант, когда связаны первые два аргумента. В этом случае третий аргумент будет означен строкой, полученной приписыванием второго аргумента к первому.

Пример – Использование concat в первом варианте:

Запрос	Ответ
`concat ("stu","dent" X)`	`X="student"`

Второй вариант, когда связанными оказались первый и третий аргументы. В этой ситуации второй аргумент будет означен строкой, при приписывании которой к первому аргументу получится третий аргумент (если, конечно, такая строка существует). Если такой строки не может быть, предикат будет неуспешен.

Пример – Использование concat во втором варианте:

Запрос	Ответ
`concat ("stu",X,"student")`	`X="dent"`

Третий вариант, когда связанными аргументами оказываются второй и третий, а свободным — первый. В этом случае, естественно, первый аргумент будет означен строкой, при приписывании к которой можно получить третий аргумент, если это вообще возможно. Иначе предикат терпит неудачу.

Пример – Использование concat в третьем варианте

Запрос	Ответ
`concat (X,"dent","student")`	`X="stu"`

Предикат **`frontchar`** служит для разделения исходной строки на первый символ и «хвост», состоящий из оставшихся после удаления первого символа, символов строки Первый и третий аргументы данного предиката принадлежат строковому домену, а второй — символьному. У этого предиката пять вариантов использования. Опишем наиболее распространенные варианты его использования.

Первый вариант, когда первый аргумент связан, а второй и третий — свободны. В этом случае второй аргумент будет означен первым символом строки, а в третий аргумент будут записаны все символы исходной строки, начиная со второго.

Пример – Использование frontchar в первом варианте:

Запрос	Ответ
`frontchar ("student",X,Y)`	`X = "s"` `Y = "tudent"`

Второй, также часто используемый вариант этого предиката, когда наоборот, первый аргумент свободен, а второй и третий — связаны. В этом случае в первый аргумент попадет строка, образованная приписыванием строки, находящейся в третьем аргументе, к символу, которым означен второй аргумент. Некий аналог конкатенации, но соединяются не две строки, а символ и строка.

Пример – Использование frontchar во втором варианте:

Запрос	Ответ
`frontchar (X,"a","nna")`	`X = "anna"`

Предикат **`frontstr`** обобщает предикат **`frontchar`** в том смысле, что он тоже позволяет «откусить» от данной строки некоторое количество символов, но не обязательно один. Предикат имеет четыре параметра. В первом параметре указывается количество символов, которые копируются из второго параметра в третий, остатком второго параметра означивается четвертый аргумент. Этот предикат может использоваться только единственным описанным способом, а именно, первые два параметра у него входные, а третий и четвертый — выходные.

Пример – Использование frontstr:

Запрос	Ответ
`frontstr (3,"world",X ,Y)`	`X = "wor"` `Y = "ld"`

Если количество символов, указанных в первом параметре предиката frontstr, превышает длину строки из второго параметра, предикат терпит неудачу.

И, наконец, предикат **fronttoken** также дробит исходную строку, указанную в качестве первого параметра предиката, на две части. Во второй аргумент предиката попадает первый атом, входящий в строку, размещенную в первом аргументе, в третий — остаток входной строки, полученный после удаления из нее атома. Напомним, что атом — это или идентификатор, или беззнаковое число (целое или вещественное), или символ. У этого предиката существует пять вариантов использования.

Первый вариант, когда первый аргумент связан, а второй и третий — свободны. В этом случае второй аргумент будет означен первым атомом строки, находящейся в первом аргументе, а в третий аргумент будет записан остаток исходной строки.

Пример – Использование fronttoken в первом варианте:

Запрос	Ответ
`fronttoken ("w2orld",X ,Y)`	`X = "w"` `Y = "2orld"`

Второй вариант использования этого предиката — когда, наоборот, первый аргумент свободен, а второй и третий — связаны. В таком случае этот предикат работает точно так же, как и конкатенация строк. В первый аргумент попадет строка, образованная приписыванием строки, находящейся в третьем аргументе, к строке, которой означен второй аргумент.

Пример – Использование fronttoken во втором варианте:

Запрос	Ответ
`fronttoken (X,"Hello","_world")`	`X = "Hello_world"`

Третий вариант получается, когда первый и второй аргументы означены, а третий нет. В этой ситуации в третий аргумент будут переписаны все символы первого аргумента, остающиеся после удаления первого атома, в случае, если первый атом первого аргумента совпадает со вторым аргументом. В противном случае предикат терпит неудачу.

Четвертый вариант подобен третьему, но свободен второй аргумент, а связаны первый и третий. В этом случае второй аргумент означивается первым атомом первого аргумента, если оставшиеся символы первого аргумента совпадают со строкой, находящейся в третьем аргументе. Иначе предикат будет неудачен.

И, наконец, пятый вариант использования этого предиката возникает, когда все три его аргументы означены. В этом случае он будет истинным, если строка, хранящаяся в первом аргументе, совпадает со строкой, полученной приписыванием к атому из второго аргумента строки из третьего аргумента. В противном случае предикат будет ложным

Пример – Использование fronttoken в пятом варианте:

Запрос	Ответ
`fronttoken ("Hello_world","Hello","_world")`	`Yes`

Рассмотри пример, где применяются данные предикаты. Создадим предикат, который будет преобразовывать строку в список символов. Предикат будет иметь два аргумента. Первым аргументом будет данная строка, вторым — список, состоящий из символов исходной строки.

Решение, как всегда, будет рекурсивным. Базис: пустой строке будет соответствовать пустой список. Шаг: с помощью встроенного предиката frontchar разобьем строку на первый символ и остаток строки, остаток строки перепишем в список, после чего добавим первый символ исходной строки в этот список в качестве первого элемента. Переведем наши рассуждения на язык Turbo Prolog:

```
str_list("",[]).      /* пустой строке соответствует пустой список */
str_list(S,[H|T]):-
frontchar(S,H,S1),  /* H — первый символ строки S,
                                S1 — остаток строки */
str_list(S1,T).       /* T — список, состоящий из символов, входящих в
                                строку S1*/
```

6.4 Задания на лабораторную работу

Реализовать на Прологе следующие операции:

1) модифицировать описанный выше предикат так, чтобы он преобразовывал строку не в список символов, а в список атомов;

2) разработать предикат, который будет преобразовывать список символов в строку. Предикат будет иметь два аргумента. Первым аргументом будет список символов, вторым — строка, образованная из элементов списка;

3) создать предикат, который по строке и символу подсчитает количество вхождений этого символа в данную строку. Предикат будет иметь три аргумента: первые два — входные (строка и символ), третий — выходной (количество вхождений второго аргумента в первый);

4) разработать предикат, который по символу и строке будет возвращать первую позицию вхождения символа в строку, если символ входит в строку, и ноль, если не входит. У предиката будет три параметра. Первые два — входные — символ и строка, третий — выходной — первая позиция вхождения первого параметра во второй параметр или ноль;

5) создать предикат, который будет заменять в строке все вхождения одного символа на другой символ. У предиката будет четыре параметра. Первые три — входные (исходная строка; символ, вхождения которого нужно заменять; символ, которым нужно заменять первый символ); четвертым — выходным — параметром должен быть результат замены в первом параметре всех вхождений второго параметра на третий параметр;

6) разработать предикат, который будет удалять часть строки. Предикат будет иметь четыре параметра. Первые три входные: первый — исходная строка, второй — позиция, начиная с которой нужно удалять символы, третий — количество удаляемых символов. Четвертым — выходным — параметром будет результат удаления из строки, указанной в первом параметре, символов, в количестве, указанном в третьем параметре, начиная с позиции, указанной во втором параметре;

7) разработать предикат, который будет копировать часть строки. Предикат будет иметь четыре параметра. Первые три входные: первый — исходная строка, второй — позиция, начиная с которой нужно копировать символы, третий — количество копируемых символов. Четвертым — выходным — параметром будет результат копирования символов из строки, указанной в первом параметре, в количестве, указанном в третьем параметре, начиная с позиции, указанной во втором параметре;

8) разработать предикат, который позволит вставить одну строку внутрь другой строки. Предикат будет иметь четыре параметра. Первые три входные: первый — вставляемая строка; второй — строка, в которую нужно вставить первый аргумент; третий — позиция, начиная с которой нужно вставить первый параметр во второй. Четвертым — выходным — параметром будет результат вставки строки, указанной в первом параметре, в строку, указанную во втором параметре, начиная с позиции, указанной в третьем параметре;

9) создать предикат, который будет подсчитывать количество имеющихся в. строке цифр. Предикат будет иметь всего два аргумента. Входным аргументом будет строка, количество цифр в которой нужно подсчитать, выходным аргументом будет количество цифр в первом аргументе.

6.5 Примерный перечень вопросов

1) Что называется строкой в Turbo Prolog?

2) Какие встроенные предикаты используются для определения длины строки? Перечислите частоупотребляемые варианты их использования.

3) Какие встроенные предикаты используются для разделения строк на части? Перечислите частоупотребляемые варианты их использования.

4) Какие встроенные предикаты используются для объединения строк? Перечислите частоупотребляемые варианты их использования.

7 Лабораторная работа №7. Реализация деревьев и основных операций с ними

Лабораторная работа №7 предназначена для приобретения навыков в использовании средств языка Турбо-Пролога для работы с деревьями.

Разработать программы в соответствии с заданиями в пункте 7.4

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 7.3 и 7.4, а также составление отчета, в соответствии с пунктом 7.2. Примерный перечень вопросов в приведен в пункте 7.5.

На выполнение и защиту лабораторной работы отводится 4 академических часа.

7.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);
2) написать программу для каждого задания в среде программирования Turbo Prolog;
3) распечатать текст и результат программы в отчет;
4) оформить отчет по лабораторной работе;
5) защитить лабораторную работу.

7.2 Содержание отчета

1) титульный лист;
2) цель работы;
3) формулировка каждого задания;

4) текст программы для каждого задания с комментариями;

5) результат выполнения программы.

7.3 Краткие теоретические сведения

Чтобы легче понять сложную структуру, ее обычно представляют в виде дерева, в котором каждому функтору соответствует вершина, а компонентам соответствуют ветви дерева. Каждая ветвь может указывать на другую структуру.

Введем определение дерева, рассмотрев немного теорию графов.

Обычно графом называют пару множеств: множество вершин и множество дуг (множество пар из множества вершин). Различают ориентированные и неориентированные графы. В ориентированном графе каждая дуга имеет направление (рассматриваются упорядоченные пары вершин). Графически обычно принято изображать вершины графа точками, а связи между ними - линиями, соединяющими точки-вершины.

Путем называется последовательность вершин, соединенных дугами. Для ориентированного графа направление пути должно совпадать с направлением каждой дуги, принадлежащей пути. Циклом называется путь, у которого совпадают начало и конец.

Две вершины ориентированного графа, соединенные дугой, называются отцом и сыном (или главной и подчиненной вершинами). Известно, что если граф не имеет циклов, то обязательно найдется хотя бы одна вершина, которая не является ничьим сыном. Такую вершину называют корневой. Если из одной вершины достижима другая, то первая называется предком, вторая - потомком.

Деревом называется граф, у которого одна вершина корневая, остальные вершины имеют только одного отца и все вершины являются потомками корневой вершины.

Листом дерева называется его вершина, не имеющая сыновей. Кроной дерева называется совокупность всех листьев. Высотой дерева называется наибольшая длина пути от корня *к* листу.

Нам будет удобно использовать следующее рекурсивное определение бинарного дерева: дерево либо пусто, либо состоит из корня, а также левого и правого поддеревьев, которые в свою очередь также являются деревьями.

Деревья часто используются для представления других структур данных, например, списков, множеств и для поиска элементов. Эффективность поиска можно улучшить, если между элементами дерева установить отношения порядка. Тогда можно элементы в дереве упорядочить слева направо в соответствии с этим отношением. Упорядочивание может производится либо в алфавитном порядке, либо в числовом порядке, либо в порядке более сложном (по частям элементов или их величине).

Непустое дерево упорядочено слева направо, если:

- все вершины левого поддерева меньше корня дерева;
- все вершины правого поддерева больше корня дерева;
- оба поддерева упорядочены.

Такое двоичное дерево называется упорядоченным деревом, или двоичным справочником в соответствии с рисунком 6. Экономия при поиске достигается за счет того, что достаточно просмотреть не более одного поддерева.

Для большего будем считать, что в вершинах дерева располагаются целые числа.

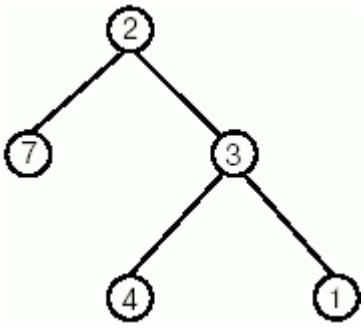


Рисунок 6 – Упорядоченное бинарное дерево

Для задания дерева в программе необходимо в разделе доменов описать домен дерева:

```
Domains
treetype = tr(integer, treetype, treetype);empty,
```

где treetype – имя домена, определяемое пользователем; tr(integer, treetype, treetype) – функтор, имя которого и порядок объектов определяет пользователь; empty – пустой функтор, имя функтора определяет пользователь.

Пример – Описание дерева на рисунке 6:

```
tr(2,tr(7,empty, empty),tr(3,tr(4,empty,empty),
                tr(1,empty,empty))).
```

В общем виде можно записать так:

tr(X, L, R),

где X – корень дерева;

L – левое поддерево;

R – правое поддерево.

Основные операции, выполняемые над деревьями:

- проверка принадлежности значения дереву;
- замена всех вхождений одного значения на другое;
- подсчет общего количества вершин дерева;
- подсчет количества листьев в дереве;
- нахождение суммы чисел, находящихся в вершинах дерева;
- вычисление высоты дерева;
- проверка принадлежности значения двоичному справочнику;
- преобразование произвольного списка в двоичный справочник;

– «сворачивание» двоичного справочника в список с сохранением порядка элементов;

– добавление нового элемента в дерево на любой уровень;

– удаление элемента дерева с любого уровня;

Приведем пример, который будет показывать реализацию проверки принадлежности значения дереву. Предикат будет иметь два аргумента. Первым аргументом будет исходное значение, вторым - дерево, в котором мы ищем данное значение.

Следуя рекурсивному определению дерева , заметим, что некоторое значение принадлежит данному дереву, если оно либо содержится в корне дерева, либо принадлежит левому поддереву, либо принадлежит правому поддереву. Других вариантов нет.

Запишем это рассуждение на Прологе.

```
tree_member(X,tr(X,_,_)):-!. /* X - является корнем дерева */
tree_member(X,tr(_,L,_)):-
tree_member(X,L),!.      /* X принадлежит левому поддереву */
tree_member(X,tr(_,_,R)):-
tree_member(X,R).        /* X принадлежит правому поддереву */
```

7.4 Задания на лабораторную работу

Реализовать основные операции, выполняемые над деревьями:

1) разработать предикат, который будет заменять в дереве все вхождения одного значения на другое. У предиката будет четыре аргумента: три входных (значение, которое нужно заменять; значение, которым нужно заменять; исходное дерево), четвертым - выходным - аргументом будет дерево, полученное в результате замены всех вхождений первого значения на второе;

2) разработать предикат, подсчитывающий общее количество вершин дерева. У него будет два параметра. Первый (входной) параметр - дерево, второй (выходной) - количество вершин в дереве;

3) разработать предикат, подсчитывающий не общее количество вершин дерева, а только количество листьев, т.е. вершин, не имеющих сыновей. Предикат будет иметь два параметра. Входной - исходное дерево, выходной - количество листьев дерева, находящегося в первом параметре;

4) разработать предикат, позволяющий вычислить высоту дерева. Предикат будет иметь два параметра. Первый (входной) - дерево, второй (выходной) - высота дерева, помещенного в первый параметр;

5) разработать предикат, позволяющий добавить в двоичный справочник новое значение. При этом результирующее дерево должно получиться двоичным деревом. Предикат будет иметь три аргумента. Первым аргументом будет добавляемое значение, вторым - дерево, в которое нужно добавить данное значение, третьим - результат вставки первого аргумента во второй;

6) создадим предикат, генерирующий дерево, которое является двоичным справочником и состоит из заданного количества вершин, в которых будут размещены случайные целые числа;

7) реализовать предикат, который будет удалять заданное значение из двоичного справочника. У него будет три параметра. Два входных (удаляемое значение и исходное дерево) и результат удаления первого параметра из второго;

8) измените предикат, удаляющий значение из двоичного справочника, так, чтобы удалялся не минимальный элемент правого поддерева, а максимальный элемент левого поддерева;

9) создайте предикат, находящий максимальное из значений, находящихся в вершинах дерева;

10) создайте предикат, проверяющий, что дерево является двоичным справочником;

11) создайте предикат, переписывающий дерево в двоичный справочник;

12) создайте предикат, который будет находить среднеарифметическое значений, находящихся в листьях дерева;

13) создайте предикат, находящий сумму чисел, расположенных в вершинах дерева так, чтобы он суммировал только положительные числа;

14) измените предыдущий предикат так, чтобы он вычислял произведение отрицательных чисел.

7.5 Примерный перечень вопросов

1) Что такое бинарное упорядоченное дерево (определение, описание на Турбо-Прологе)?
2) Какие основные операции выполняются над деревьями?
3) Процедурный смысл поиска в дереве.
4) Построение двоичного дерева.

8 Лабораторная работа №8. Файлы. Основные операции с файлами

Целью лабораторной работы №8 является приобретение навыков в использовании средств Turbo Prolog для работы с файлами.

Разработать программы, в соответствии с заданиями в пункте 8.4.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 8.3 и 8.4, а также составление отчета, в соответствии с пунктом 8.2. Примерный перечень вопросов приведен в пункте 8.5.

На выполнение и защиту лабораторной работы отводится 4 академических часа.

8.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);
2) написать программу для каждого задания в среде программирования Turbo Prolog;
3) распечатать текст и результат программы в отчет;
4) оформить отчет по лабораторной работе;
5) защитить лабораторную работу.

8.2 Содержание отчета

1) титульный лист;
2) цель работы;
3) формулировка каждого задания;
4) текст программы для каждого задания с комментариями;
5) результат выполнения программы.

8.3 Краткие теоретические сведения

Файлом называют именованную совокупность данных, записанных на диске. Файл состоит из компонентов (элементов). При чтении или записи файловая переменная перемещается к очередному компоненту и делает его доступным для обработки. Попробуем разобраться с тем, как можно работать с файлами в Прологе.

Для начала вспомним, что пользовательские файлы описываются в разделе описания доменов следующим образом:

file = datafile

где file – стандартный тип домена Turbo Prolog; datafile – имя файлового домена.

Если в программе необходимо объявить несколько файловых доменов, то это можно сделать в одном описании:

file = datafile1; datafile2; datafile3.

Следует обратить внимание на то, что описание файлового домена отличается от описания доменов других типов (тип домена file записывается по левую сторону от знака равенства). Имя файлового домена или символическое имя (кодификатор) может быть логически отождествлено с физическим именем файла.

Символические имена файлов еще называют внутренними или логическими именами файлов, в отличие от внешних или физических имен файлов. Символическое имя файла должно начинаться со строчной буквы.

Кроме пользовательских файлов, имеются стандартные файлы (или устройства), которые не нужно описывать в разделе описания доменов. Это:

- `stdin` (стандартное устройство ввода);
- `stdout` (стандартное устройство вывода);
- `stderror` (стандартное устройство вывода сообщений об ошибках);
- `keyboard` (клавиатура);
- `screen` (монитор);
- `printer` (параллельный порт принтера);
- `coml` (последовательный порт).

По умолчанию стандартным устройством ввода является клавиатура, а стандартным устройством вывода — монитор. Чтобы начать работу с пользовательским файлом, его нужно открыть, а по завершении работы— закрыть. Стандартные устройства ввода/вывода, а также параллельный и последовательный порты открывать и закрывать не нужно.

В Turbo Prolog существуют встроенные предикаты, с помощью которых можно осуществлять операции открытия и закрытия файлов, а также многие другие операции с файлами. Данные предикаты приведены в приложение В.

Примечание – Все приведенные предикаты, предназначенные для открытия файлов имеют два входных параметра. Первый параметр — это внутреннее символическое имя, указанное в разделе описания доменов, второй параметр — это строка, представляющая внешнее имя файла.

Символ "\", обычно используемый для разделения имен каталогов, применяется в Турбо Прологе для записи кодов символов, поэтому требуется использовать вместо одного обратного слэша два ("\\").

Пример – Запись пути в Пролог:

```
"C:\\Prolog\\BIN".
```

В Приложении Г приведен пример, который демонстрирует программу, выводящая содержимое произвольного файла на экран.

8.4 Задания на лабораторную работу

Реализовать операции, выполняемые над файлами:

1) напишите замену для стандартного предиката openwrite, который будет открывать файл на запись, если файл существует, и выводить соответствующее сообщение, если он отсутствует;

2) напишите замену для стандартного предиката openmodify, который будет открывать файл на чтение и запись, если файл существует, и выводить соответствующее сообщение, если файл отсутствует;

3) напишите замену для стандартного предиката openappend, который будет открывать файл на дозапись, если файл существует, и выводить соответствующее сообщение, если он отсутствует;

4) создать предикат, который будет формировать файл из символов, вводимых с клавиатуры. Предикат будет иметь один параметр, представляющий собой внутреннее имя файла;

5) создать предикат, который будет выводить содержимое файла на экран. Предикат будет иметь один параметр, представляющий собой внутреннее имя файла;

6) создать предикат, который будет переписывать компоненты одного файла в другой файл так, чтобы в итоговом файле все английские буквы были большими. У предиката будет два аргумента строкового типа. Первым параметром будет внешнее имя исходного файла, вторым параметром — внешнее имя итогового файла;

7) создайте предикат, вычисляющий сумму чисел, хранящихся в файле.

8.5 Примерный перечень вопросов

1) Что понимают под файлом в Turbo Prolog?

2) Какие стандартные устройства ввода – вывода существуют?

3) Какие отличия работы с файлами, относительно императивных языков, существуют?

4) Какие особенности в записи директории существуют?

5) Перечислите некоторые встроенные предикаты для открытия файлов.

6) Перечислите некоторые встроенные предикаты для перенаправления ввода-вывода файлов.

9 Лабораторная работа №9. Работа с динамическими базами данных

Целью лабораторной работы №9 является приобретение навыков в использовании средств Turbo Prolog для работы базами данных.

Разработать программы, в соответствии с заданиями в пункте 9.4.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 9.3 и 9.4, а также составление отчета, в соответствии с пунктом 9.2. Примерный перечень вопросов приведен в пункте 9.5.

На выполнение и защиту лабораторной работы отводится 4 академических часа.

9.1 Ход работы

1) изучить теоретический материал по теме лабораторной работы (лекции, учебники);
2) написать программу для каждого задания в среде программирования Turbo Prolog;
3) распечатать текст и результат программы в отчет;
4) оформить отчет по лабораторной работе;
5) защитить лабораторную работу.

9.2 Содержание отчета

1) титульный лист;
2) цель работы;
3) формулировка каждого задания;
4) текст программы для каждого задания с комментариями;
5) результат выполнения программы.

9.3 Краткие теоретические сведения

Факты, описанные в разделе clauses, можно рассматривать, как статическую базу данных (БД). Эти факты являются частью кода программы и не могут быть оперативно изменены. Для создания динамической базы данных в Turbo Prolog предусмотрен специальный раздел database.

Предикаты в этом разделе могут иметь такую же форму представления, что и в статической части ПРОЛОГ-программы, но должны иметь другое имя.

Пример – Объявление динамической БД в программе.

```
DOMAINS
name = symbol
rost, ves = integer
DATABASE
dplayer(name, rost, ves)
PREDICATES
player(name, rost, ves)
CLAUSES
player("Михайлов", 180, 87)
player("Петров", 187, 93)
player("Харламов", 177, 80)
```

Допустим, что необходимо после запуска программы переместить данные из статической БД в динамическую. Для этого можно описать следующее правило:

```
assert_database:-player(N, R, V), assertz(dplayer(N, R, V)),fail.
```

В этом правиле использован встроенный предикат assertz, который помещает утверждение в базу данных после всех утверждений, которые там уже имеются.

Существуют встроенные предикаты Turbo Prolog, которые позволяют производит модификации существующей статической БД, и создание новой БД. В таблице 2 перечислены основные из них.

Таблица 2 – Встроенные предикаты для работы с базами данных

Название предиката	**Формат**	**Действие, производимое предикатом**
asserta	asserta (clause).	Заносит новый факт в БД перед уже имеющимися утверждениями данного предиката.
assert	assert (clause).	Заносит новый факт в БД за всеми имеющимися утверждениями данного предиката.
assertz	assertz (clauses)	Добавляет новый факт в БД после других фактов (в конец базы данных).
retract	retract (clause).	Удаляет утверждение из БД
retractall	retractall (clause).	Удаляет все утверждения данного предиката из БД.
findall	findall (C, p(_, _,_ C,_)L)	Позволяет собрать все имеющиеся в БД утверждения данного предиката в список.
save	save(file_name).	Запись динамической БД в файл на диск. В каждую строку файла заносится один факт. Если заданный файл уже существует, то его содержание уничтожается
consult	consult (File_name).	Загрузка содержимого файла в динамическую БД.
readterm	readterm(Domain, Term).	Считывает из файла объекты, относящиеся к определенному домену в БД.

Главное достоинство БД на Turbo Prolog, как и любой другой БД, заключается в возможности быстрого выборочного доступа к информации. Например, в приведенном выше примере это может быть запрос:

```
GOAL: player(N, R, V),R>180
```

Если сравнивать основные понятия ПРОЛОГа и реляционных БД, то получится следующая таблица.

Таблица 3 – Сравнение основных понятий в ПРОЛОГа и реляционных БД.

Основные понятия ПРОЛОГа	**Основные понятия реляционной БД**
Предикат	Отношение (таблица)
Объект	Атрибут отношения
Отдельное утверждение	Элемент отношения (запись)
Количество утверждений	Мощность отношения

Таким образом, можно сказать, что любая ПРОЛОГ-программа может быть названа базой данных.

Каково же главное отличие ПРОЛОГ-программ от реляционных БД? В БД можно только извлекать существующие сведения или изменять данные по заданному закону. В ПРОЛОГ-программе за счет использования правил и логического вывода можно получать новые знания. Поэтому ПРОЛОГ-программа может рассматриваться в качестве базы знаний (экспертной системы).

Создание БД в Turbo Prolog начинается с этапа проектирования. На данном этапе необходимо разработать проект БД с учетом следующих факторов:

- размер базы данных;
- организация элементов базы данных;
- способы работы и содержание базы данных.

Необходимо не только создать БД, но и создать систему управления базой данных (СУБД), ориентированную на диалог с пользователем. Любая СУБД должна иметь следующие возможности:

- занесение в БД новых данных;
- удаление из БД необходимых данных;
- выборка и вывод содержащихся в БД данных.

СУБД должна содержать меню, представляющее пользователю все возможности системы, а также оконную систему, дающую полное представление о доступных пользователю средствах.

Динамическая БД может содержать в Turbo Prolog только факты.

Создать динамическую БД можно двумя способами:

– из статической БД;

– путем добавления новых утверждений в пустую БД.

Создание динамической БД из статистической БД.

Утверждения динамической БД в оперативной памяти помещается отдельно от утверждений статической БД. Предикаты статической БД описываются в разделе predicates. Часто бывает удобно часть информации БД иметь в статической БД, затем эту информацию занести в динамическую БД. В этом случае предикаты статической имеют имя, отличное от имени динамической БД. Обычно используется добавление латинской буквы d в начало имени предиката динамической БД для отличия от имени предиката статической БД. Форма представления данных, домены объектов предикатов статической и динамической БД должны совпадать. В статической БД утверждения одного и того же предиката должны быть сгруппированы в одном месте. В следующем примере представлена реализация создания динамической БД из статической БД.

Пример – создание динамической БД из статической БД

```
domains
title,author,publ=symbol
year=integer
database
dbook(title,author,publ,year)
predicates
book(title,author,publ,year)
create_database
goal
create_database.
clauses
```

```
    book('Использование Турбо-Пролога', 'Ц,Ин,Д.Соломон',
'Мир', 1998).
    book('Программирование на языке Пролог для
искусственного интеллекта', ‘И. Братко, 'Мир',1995).
    book('Турбо-Пролог в сжатом изложении', 'А.Янсон','Мир',
1997).
    . . .
    create_database:-
    book(Title, Author, Publ, Year), (1)
    assertz(dbook(Title, Author, Publ, Year),fail.
    create_database:-!.   (2)
```

Целевое утверждение данной программы create_database представляет процедуру состоящую из правила (1) и правила (2). Правило (1) использует метод откат после неудачи, которая создается предикатом fail. Первая подцель правила (1) присваивает значения переменным Title, Author, Publ, Year. Вторая подцель заносит эти переменные в динамическую базу данных. Предикат book – предикат статической БД, находящейся в разделе clauses. Предикат dbook – предикат динамический БД.

Пример – Процедура создания динамической БД путем добавления новых утверждений в БД

```
    create_database:-                    (1)
    write  (“Введите название книги”), nl,
    readln (Title), Title <>”*'”,
    write  (“Введите автора книги”), nl,
    readln (Author),
    write  (“Введите название издательства”), nl,
    readln (Publ),
    write  (“Введите год издания”), nl,
```

```
readln (Year),
assertz (dbook(Title, Author, Publ, Year),
create_database.
create_database:- !.                    (2)
```

Процедура состоит из рекурсивного правила (1) и правила (2). Создание динамической БД происходит до тех пор, пока не будет введен признак конца ввода ("*"). Приведем еще одни пример.

Пример – Процедура поиска в динамической БД книги по ее названию

```
search:-write("Введите название книги"), nl,
readln(Title),
dbook(Author,Title,Publ,Year),
write(Author, " ",Title, " ",Publ, " ",year).
search:-write("Книга с данным названием в БД отсутствует").
```

9.4 Задание на лабораторную работу

1) Для предметной области задания 1 из лабораторной работы № 2 осуществить следующее:

a) преобразование статической базы данных в динамическую;

b) возможность добавления записей в базу данных;

c) возможность удаления записей из базы данных;

d) возможность коррекции записей в базе данных;

2) Создать динамическую БД о школьниках. Имеющиеся сведения о школьниках приведены в следующей:

Фамилия	Год рождения	Класс	Средний бал аттестата
Белов	1992	11В	4
Здорин	1991	11Б	5
Грищанов	1994	9А	3
Смирнов	1993	9Г	4
Алексеев	1990	11А	5

Предикат статистической БД, хранящей информацию о школьниках, имеет вид: school(name, year, team,mark).

3) Сделать модуль menu, которое высвечивает 4 доступные пользователю опции:

a) занесение в БД новой информации о школьниках;
b) удаление информации о школьниках;
c) выдача интересующей пользователя информации;
d) окончание работы программы.

Модуль также должен запрашивать у пользователя режим работы согласно опции. В соответствии с выбранным режимом работы осуществляется выбор правил:

a) process(1) для занесения информации в БД;
b) process(2) для удаления информации из БД;
c) process(3) для выборки данных;
d) process(4) для окончания работы с программой.

Каждый из этих модулей создает свое диалоговое окно.

9.5 Примерный перечень вопросов

1) Статическая и динамическая базы данных, определения, описания предикатов.

2) Предикаты для работы с утверждениями динамической базы данных. Их особенности и описания.

10 Лабораторная работа №10. Ознакомление со средой программирования Visual Prolog. Логический вывод и логическое следствие. Унификация и сопоставление

Лабораторная работа №10 предназначена для приобретения необходимых навыков в работе со средой Visual Prolog, в использовании меню системы и основных команд трассировки программ, а также в написании программ на Visual Prolog.

Разработать программы, в соответствии с заданиями в пункте 10.4.

Заканчивается работа защитой с оценкой «зачтено» или «незачтено» в виде экспресс-опроса в рамках теоретического и практического материала, отнесенного к пунктам 10.1 – 10.3.

На выполнение и защиту лабораторной работы №10 отводится 2 академических часа.

10.1 Краткие теоретические сведения

Решение задачи средствами языка Visual Prolog осуществляется путем логического вывода необходимых сведений из уже известных.

Программа на VP представляет собой конечную последовательность определенных фактов, относящихся к определенной предметной области и правил вывода логических заключений из этих фактов.

Программа состоит из следующих блоков, начало которых обозначается различными ключевыми словами:

domains – раздел описания доменов (начинается со строчной буквы!), которые позволяют задавать разные имена различным видам данных, чтобы они не выглядели одинаково и программист помнил что и в каком порядке используется для обработки

predicates – раздел определения фактов и правил. Объявление предиката начинается с имени предиката (начинается с буквы, можно использовать знак «_», но нельзя знаки «–» , «*» , пробел), а затем в () перечисления доменов (стандартных, т.е. типов и описанных ранее в разделе доменов)

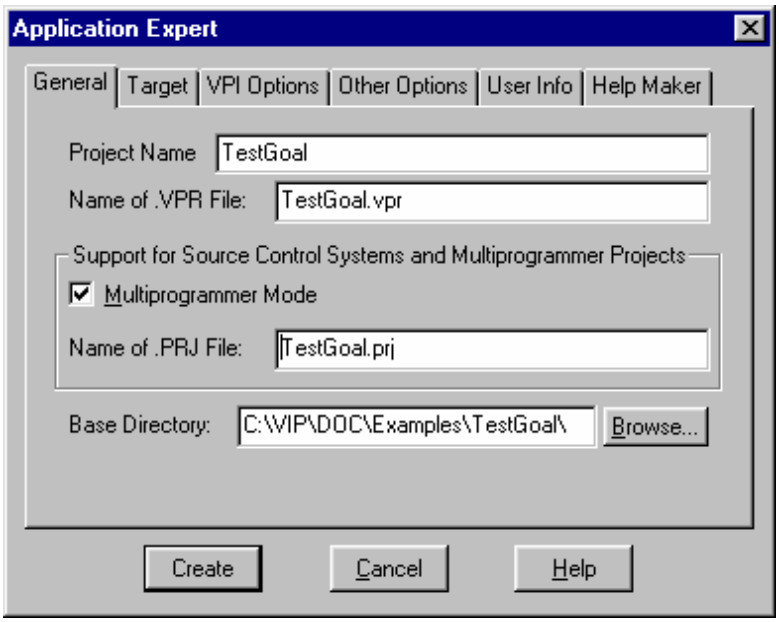
clauses – раздел записи фактов, отношений фактов и правил. Правило создается с помощью логических выражений. В VP дизъюнкция обозначается «;», конъюнкция «,», если «:-». Каждое правило и факт заканчивается точкой.

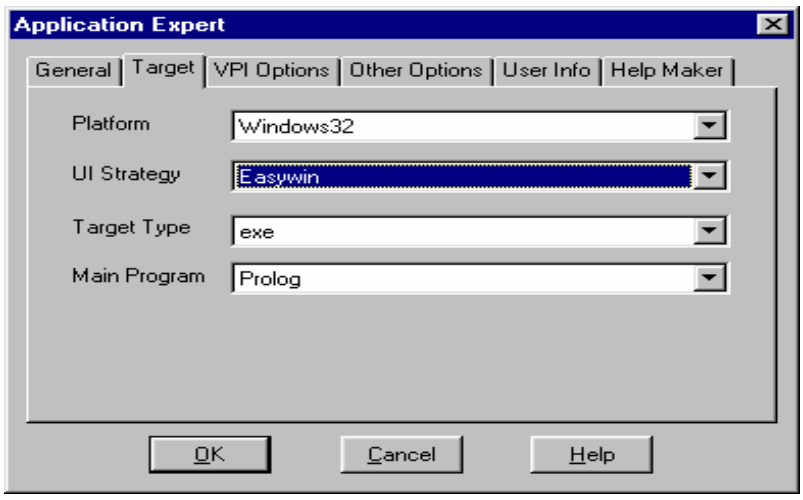
goal – раздел запросов, как и в правилах использует переменные, логические операции и имена фактов и правил.

Для более комфортного запуска программ на VP рекомендуется использовать утилиту среды визуальной разработки Test Goal (Ctrl+G), создав специальный TestGoal – проект. Выполните следующую последовательность операций:

1) создайте новый проект Project – New Project;

2) в появившемся диалоговом окне Application Expert введите следующие данные;





3) после нажатия кнопки Create следует установить требуемые опции компилятора для созданного TestGoal-проекта:

а) Options – Project – Compiler Option;

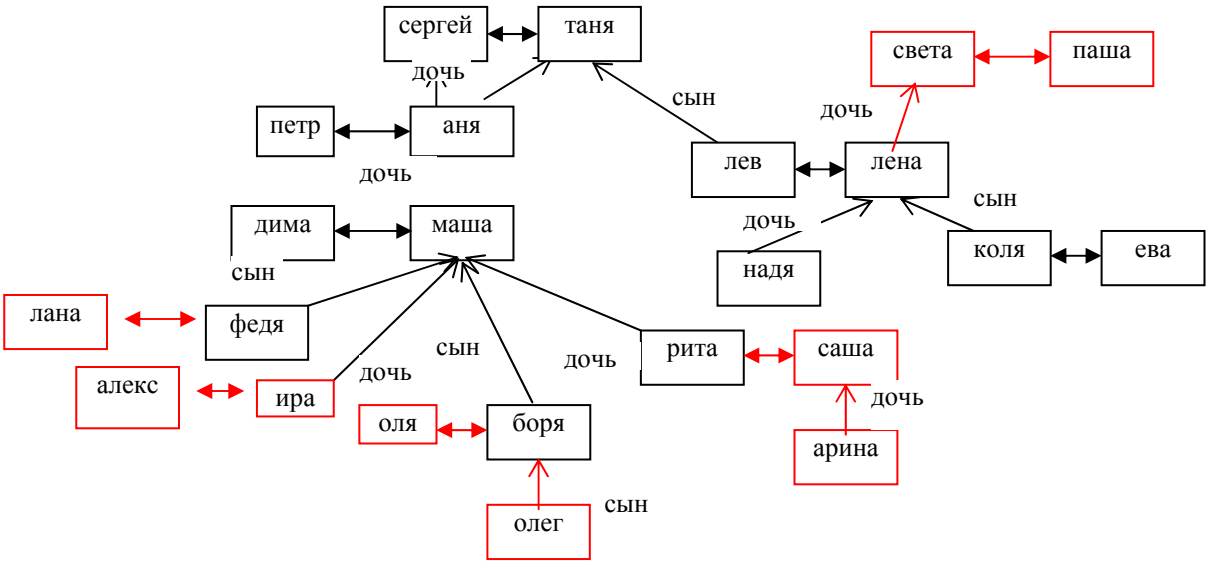
б) откройте вкладку Warnings. Установите переключатель Nondeterm, чтобы компилятор все определенные пользователем предикаты как недетерминированные;

в) снимите флажки Non Quoted Symbols, Strong Type Conversion Check, Check Type of Predicates. Это будет подавлять некоторые возможные предупреждения компилятора, которые не важны для выполнения несложных программ.

Теперь все программы следует открывать в запущенном командой Project – Open project проекте командой File – Open.

10.2 Задания на лабораторную работу

Рассмотрим предметную область «Родственные связи»:



Примечание – Двусторонняя стрелка указывает на то, что пары женаты.

Требуется узнать, у кого из мужчин есть дети и кто они. А так же вывести список замужних женщин.

Ход работы:

1) запустите среду Visual Prolog;

2) откройте новое окно редактирования программы (File - New), файлы программ имеют расширение *.pro;

3) наберите расположенный ниже текст программы, сохраните во избежание утери и нажмите Ctrl+G, а затем Continue Evaluation;

4) далее в окне сообщений (внизу экрана) вы увидите происходящий процесс, который закончится либо выдачей списка ошибок, либо полной компиляцией;

5) если необходимо, исправьте ошибки, проверьте правильность результата и выполните самостоятельное задание, для которого возможно потребуется добавить связи, отмеченные на рисунке красным цветом.

```
domains
    name=symbol
predicates
    men(name)
    women(name)
    married(name,name)
    son(name,name)
```

```
        daughter(name,name)
        father(name,name)
        wife(name,name)
    clauses
    /*определяем факты-свойства – кто является  мужчиной
        men(sergey).men(petr).men(lev).men(dima).
        men(kolya).men(fedya).men(borya).
    /*определяем факты-свойства – кто является женщиной
        women(tanya).women(anna).women(lena).women(masha).
        women(eva).women(nadya).women(rita).
    /*определяем факты-отношения женатых пар
        married(sergey,tanya).married(petr,anna).
        married(lev,lena).married(dima,masha).
        married(kolya,eva).
    /*определяем факты-отношения сыновей
        son(lev,tanya).son(fedya,masha).son(kolya,lena).
        son(borya,masha).
    /*определяем факты-отношения дочерей
        daughter(anna, tanya).daughter(masha, anna).
        daughter(nadya, lena).daughter(rita, masha).
    /*выводим правило, по которому возможно определить отношение отец-дочь
или отец-сын
        father(X,Y):-
        men(X),married(X,Z),son(Y,Z);men(X),married(X,Z),
        daughter(Y,Z).
    /*выводим правило определения жен
        wife(X,Y):-married(Y,X).
    goal
        write("\t\t\tFathers and childes"),nl,father(X,Y);
        write("\t\t\tWifes"),nl,wife(X,_).
```

10.3 Примерный перечень вопросов

1) Что представляет из себя программа на VP?
2) Из каких блоков состоит программа на VP?
3) Как начинается работа в среде VP?

Список использованных источников

1. Бураков, М.В. Язык логического программирования Пролог: методические указания к выполнению лабораторных работ / М.В. Бураков. – СПб.: СПбГУАП, 2003. – 37 с.
2. Касаткин, В.Н. Логическое программирование в занимательных задачах / В.Н. Касаткин. – Киев: Техніка, 1980. – 79 с., ил.
3. Лебедева, Н.С. Логическое программирование: учебное пособие / Н.С. Лебедева. – Рига: Transporta un sakaru institutes, 2002. – 91 с.
4. Логическое программирование: пер. с англ. и фр. В. Н. Агафонова. – М.: Мир, 1988. – 368 с., ил. – ISBN 5-03-000972-8.
5. Метакидес, Г. Принципы логики и логическое программирование / Г. Метакидес, А. Нероуд; пер. с англ. под ред. к.ф.м.н. В.А. Захарова и акад. В.А. Садовничего. – М.: Изд-во «Факториал», 1998.–288 с.– ISBN 5-88688-037-2.
6. Стерлинг, Л. Искусство программирования на языке Пролог: пер. с англ. / Л. Стерлинг, Э. Шапиро. – М.: Мир, 1990. – 235 с., ил. – ISBN 5-03-000406-8.
7. Терехин, В.В. Turbo Prolog: учебное пособие / В.В. Терехин – Новокузнецк: Кемеровский государственный университет, 2005. – 119с.

Приложение А

(справочное)

Таблица А.1 – Назначение пунктов меню Турбо Prolog

Files	**Работа с файлами**
– Load	Загрузить исходный текст программы на Турбо-Прологе
– Pick	Загрузить исходный текст программы из списка программ
– New file	Создать новый файл (по умолчанию создается WORK.PRO)
– Save	Записать текст программы в файл
– Write to	Записать текст программы в файл под другим именем
– Directory	Посмотреть содержимое директорий с определенной маской для файлов
– Change dir	Сменить директорию
– OS shell	Временный выход в DOS
– Quit	Выход из Турбо-среды
Edit	**Редактирование текста программы**
Run	**Запуск программы**
Compile	**Компиляция и компановка программы**
– Memory	Компиляция в памяти
– OBJ file	Компиляция с созданием объектного файла (требуется наличие в программе раздела goal)
– Exe file (auto link)	Создание exe-файла (происходит автоматическая компановка)
– Project	Компилирование проекта (всех модулей)
– Link only	Только компановка
Options	**Различные опции**
– Link options	Опции компановщика
– Edit PRJ file	Редактирование PRJ-файла (проекта)
– Compiler directives	Директивы компилятора
Setup	**Установки**
– Colors	Цвета
– Window size	Размер окна
– Directories	Директории
– Miscellaneous	Разное
– Load SYS file	Загрузить PROLOG.SYS файл
– Save SYS fale	Записать PROLOG.SYS файл

Приложение Б

(справочное)

Таблица В.1 – Стандартные средства для организации вычислений

X + Y	Сумма X и Y
X - Y	Разность X и Y
X * Y	Произведение X и Y
X / Y	Деление X на Y
X mod Y	Остаток от деления X на Y
X // Y	Деление нацело X на Y
X ** Y	Возведение X в степень Y
- X	Смена знака X
abs(X)	Абсолютная величина числа X
max(X,Y)	Большее из чисел X и Y
min(X,Y)	Меньшее из чисел X и Y
sqrt(X)	Квадратный корень из X
random(Int)	Случайное целое число в диапазоне от 0 до Int
sin(X)	Синус X
cos(X)	Косинус X
tan(X)	Тангенс X
log(X)	Натуральный логарифм (ln) числа X
log10(X)	Десятичный логарифм (lg) числа X
float(X)	Вещественное число, соответствующее целому числу X
pi	3.14159 (приближенное значение числа π)
e	2.71828 (приближенное значение числа e)
`trunc(X)`	отбрасывает дробную часть X
`round`	округляет вещественное число X до ближайшего целого
`fail`	используется для организации поиска с возвратом
`free(X)`	истинен, если X является свободной переменной, и ложен в противном случае.
`bound(X)`	истинен, если X - это связанная переменная, и ложен, если его аргумент свободен

Приложение В

(справочное)

Таблица В.1 - Встроенные предикаты для работы с файлами

Название предиката	Формат предиката	Операции, производимые предикатом
1	2	3
openread	openread(Log_file_name, File_name)	Открывает файл File_name для чтения и отождествляет его с логическим файлом Log_file_name. Если файл с указанным внешним именем не будет обнаружен, предикат терпит неудачу и выводит соответствующее сообщение об ошибке.
openwrite	openwrite (datafile, "File.db")	Открывает файл "File.db" только для записи. Этот предикат создает на диске новый файл. Если файл с указанным внешним именем уже существует, он будет стерт.
openappend	openappend (Log_file_name, File_name)	Открывает файл File_name для записи новых данных в конец файла. Если файл с указанным именем не будет обнаружен, предикат выводит соответствующее сообщение об ошибке.
openmodify	openmodify (Log_file_name, File_name)	Открывает файл прямого доступа File_name для чтения и записи одновременно. Если файл с указанным именем не будет обнаружен, предикат выводит соответствующее сообщение об ошибке.
existfile	existfile (File_name)	Проверяет, существует ли файл с указанным именем в указанном месте. Предикат истинен, если файл с именем, указанным в качестве его единственного параметра, существует, и ложен — в противном случае.
closefile	closefile (Log_file_name)	Закрывает физический файл, отождествленный с логическим файлом Log_file_name. Предикат успешен даже в том случае, если этот файл уже был закрыт.
deletefile	deletefile (File_name)	Удаляет файл, указанный в качестве его единственного параметра. Если по какой-то причине удалить файл не получается, этот предикат выдает сообщение об ошибке.
renamefile	renamefile (Old_File_name, New_ File_name)	Изменяет имя файла, указанного в качестве его первого параметра, на имя, указанное в качестве его второго параметра. Если не существует файла, чье имя указано в первом параметре, или существует файл, чье имя указано во втором параметре, предикат выдаст сообщение об ошибке.

Продолжение таблицы В.1

1	2	3
Disk	disk (Path)	Если переменной Path присвоен корректный путь доступа, то заданная с его помощью директория становится текущей. Если же переменная Path не означена, то данный предикат присваивает Path путь доступа к текущей директории.
dir	dir (Path, File_spec, File_name)	Переменной Path должен быть присвоен корректный путь доступа, переменная File_spec задает расширение, представляющей интерес группе файлов. Данный предикат выдает каталог имен файлов с заданным расширением. Можно выбрать среди них нужный и нажать Enter. Имя файла будет присвоено переменной File_name.
eof	eof (Log_file_name)	Успешен, если достигнут конец файла, в противном случае он неуспешен. В качестве его единственного входного параметра указывается символическое имя файла. Он обычно используется при организации рекурсивного считывания всех компонентов файла. Если его попытаться применить к файлу, открытому на запись, будет выдано сообщение об ошибке.
filepos	filepos (Log_file_name, fileposit,_ Mode)	Задает в файле File_name позицию, из которой будет считан, или в которую будет записан очередной символ. Число, обозначающее позицию в файле, записывается параметру File_posit, параметр Mode может принимать одно из трех значений: 0, 1, 2. Эти значения определяют то, как будет интерпретировано значение File_posit. 0–смещение указателя относительно начала файла; 1–смещение указателя относительно текущей позиции; 2–смещение указателя относительно конца файла.

Продолжение таблицы В.1

1	2	3
Предикаты для перенаправления потоков ввода-вывода:		
readdevice	readdevice (Log_file_name)	Служит для переопределения текущего устройства ввода или для того, чтобы узнать, какое устройства ввода является текущим. Имеет один параметр, в качестве которого указывается символическое имя файл.
writedevice	writedevice_ (Log_file_name) writedevice (screen) – на экран; writedevice (printer) – на принтер	Используется для переопределения текущего устройства вывода или для получения имени текущего устройства вывода. Он имеет один параметр, который может быть использован либо как входной, либо как выходной.
write	write (Record)	Служи для записи данных в текущее устройство записи.
readln	readln (Record)	Служит для чтения строки из активного устройства вывода.
readint	readint(Integer)	Служит для чтения целого числа из активного устройства вывода.
readreal	readreal (Real)	Служит для чтения вещественного числа из активного устройства вывода.
readchar	readchar (Char)	Служит для чтения символа из активного устройства вывода.
readterm	readterm (Term)	Служит для чтения терма из активного устройства вывода.
file_str	file_str (File_name, String)	Целиком читает символы файла в строку или, наоборот, записывает содержимое строки в файл, в зависимости от того, свободен ли второй параметр этого предиката.
flush	flush (File_name)	Используется для принудительной записи в файл содержимого внутреннего буфера, выделенного для файла, указанного в его единственном параметре. Обычно он используется при работе с принтером.
filemode	filemode (Log_file_name, Mode)	Он позволяет узнать или задать режим доступа к файлу. Первый параметр — символическое имя файла, второй параметр задает или принимает режим работы с файлом, имя которого указано в первом параметре.

Приложение Г

(справочное)

Листинг программы, которая выводит содержимое произвольного файла на экран

```
DOMAINS                 /* раздел описания доменов */
file = f                /* f — внутреннее имя файла */
PREDICATES              /* раздел описания предикатов */
write_file(file)
writeFile(string)
CLAUSES                 /* раздел описания предложений */
write_file(f):-
not(eof(f)),!,          /* если файл f еще не закончился */
readchar(C),            /* читаем из него символ */
write(C," "),           /* выводим символ на экран*/
write_file(f).          /* продолжаем процесс */
write_file(_)           /* это предложение нужно, чтобы предикат не потерпел
                        неудачу в случае, когда будет достигнут конец файла */
writeFile(F_N):-
existfile(F_N),!,       /* убеждаемся в существовании файла с именем F_N */
openread(f,F_N),        /* связываем внешний файл F_N с внутренним файлом f и
                        открываем его на чтение */
readdevice(f),          /* устанавливаем в качестве устройства чтения файл
                        f */
 write_file(f),         /* вызываем предикат, выводящий на экран все символы
                        файла f */
closefile(f),           /* закрываем файл */
readdevice(keyboard),   /* перенаправляем ввод на клавиатуру */
nl,nl,                  /* пропускаем строку */
write("Нажмите любую клавишу"),  /* выводим сообщение на экран */
readchar(_).            /* ждем нажатия любой клавиши */
writeFile(F_N):-
write("Файл с именем ",F_N," не наден!").  /* выдаем сообщение,
                        если предикат existfile потерпел неудачу */
GOAL                    /* раздел описания внутренней цели*/
write("Введите имя файла: "),
readln(F_N),            /* читаем название файла в переменную F_N */
writeFile(F_N).
```