МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

О.Г. ГАБДУЛЛИНА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПРОЛОГ

Рекомендовано Ученым советом государственного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет» в качестве учебного пособия для студентов, обучающихся по программам высшего профессионального образования по специальности 050202.65 – Информатика

Оренбург 2008

УДК 004.438(075.8)
ББК 32.973.26- 018.1я73
Г12

Рецензент
доктор экономических наук, профессор Шепель В.Н.

Габдуллина, О. Г.
Г 12 Программирование на языке Пролог [Текст]:
учебное пособие - Оренбург: ГОУ ОГУ, 2008. – 122 с.

В пособии изложены данные и рекомендации по изучению Турбо-Пролога. Приводятся задания для выполнения лабораторных работ.
Учебное пособие предназначено для студентов, обучающихся по программам высшего профессионального образования по специальности 050202.65 – Информатика при изучении дисциплины "Основы искусственного интеллекта".

Г 2404010000

ББК 32.973.26−018.1я73

Содержание

Введение

С введением в 1985 г. в школах предмета «Основы информатики и вычислительной техники» началось осуществление регулярной подготовки учителей информатики в педагогических вузах. Характерной особенностью информатики является быстрое развитие предметной области. С момента введения нового школьного курса информатика существенно расширила свои границы, что не могло не повлиять на систему подготовки учителей информатики. Анализ государственных образовательных стандартов выявил тенденцию к увеличению времени и объема подготовки будущих учителей информатики в области искусственного интеллекта. В государственном образовательном стандарте, утвержденном в 2005г. и действующем в настоящее время, основы искусственного интеллекта представляют собой отдельную дисциплину предметной подготовки учителей информатики, в структуру которой включено такое направление науки, как модели представления знаний; значительное место отводится знакомству с языком программирования Пролог.

Искусственный интеллект – это раздел информатики, посвященный моделированию интеллектуальной деятельности человека. Учебная дисциплина «Основы искусственного интеллекта» призвана формировать совокупность идей, обеспечить использование методов, закономерностей, понятий и средств подхода к проектированию компьютерных систем, характерного для искусственного интеллекта.

Методы искусственного интеллекта позволили создать эффективные компьютерные программы в самых разнообразных, ранее считавшихся недоступными для формализации и алгоритмизации, сферах человеческой деятельности, таких как медицина, биология, зоология, социология, бизнес, криминалистика. Идеи обучения и самообучения компьютерных программ, накопления знаний, позволили создать программы, выполняющие трудные интеллектуальные задачи. В систему знаний по курсу «Основы искусственного интеллекта» мы включили наиболее общие универсальные научные знания из выделенных разделов искусственного интеллекта. К ним с полным правом относится язык программирования Пролог. Пролог - машинный язык пятого поколения, используемый при построении экспертных систем, настраиваемых баз данных, естественно-языковых интерфейсов и интеллектуальных систем управления. Пособие содержит описание языка программирования Turbo Prolog. Теоретические сведения сопровождаются предлагаемыми для выполнения лабораторными работами. В пособии содержится большое количество упражнений для самостоятельной работы.

1 Особенности среды Турбо-Пролог для программирования баз знаний

1.1 Программирование на языке логики

Пролог - машинный язык пятого поколения, используемый при построении экспертных систем, настраиваемых баз данных, естественно-языковых интерфейсов и интеллектуальных систем управления. Свое название Пролог получил от слов "Программирование на языке Логики". Теоретической основой Пролога является раздел символьной логики, называемый исчислением предикатов. Предикат - это логическая функция, которая выражает некоторое отношение между своими аргументами и принимает значение "истина", если это отношение имеется или "ложь", если оно отсутствует.

Название "*Пролог*" произошло от словосочетания "*Программирование при помощи логики*" (PROgramming in Logic). Пролог был разработан и впервые реализован в 1973 г. *Алэном Колмероэ* и другими членами "группы искусственного интеллекта" Марсельского университета (Франция). Главной задачей группы было создание системы *для обработки естественного языка*.

Turbo-Prolog 2.0. реализован компанией *Borland International*. Он является *компиляторно - ориентированным* языком высокого уровня. Turbo-Prolog 2.0. особенно хорош для создания *экспертных систем, динамических баз данных, программ с применением естественно-языковых конструкций*. Наряду с Turbo-Prolog 2.0. созданы еще несколько реализаций языка Prolog, например, *Arity Prolog, Wisdom Prolog, Micro Prolog*. Шотландский вариант C&M Prolog получил название в честь авторов *Уильяма Клоксина и Кристоффера Меллиша* классической работы "Программирование на Прологе", которая считается неофициальным стандартом.

Одной из причин предпочтительности выбора Турбо-Пролога является то, что написанные на нем программы компилируются, в отличие от других версий Пролога, где программа интерпретируется. Преимущество откомпилированной программы является в том, что данная программа в исходном тексте не нуждается и работает на много быстрее.

Самым существенным отличием Турбо-Пролога от других версий языка является наличие в Турбо-Прологе *строгой типизации элементов данных*. Сделанные отступления позволили значительно увеличить скорость трансляции и счета программ.

Такие языки как *Паскаль, Бейсик и СИ* относятся к разряду императивных или процедурных. Программа, написанная на процедурном языке, состоит из последовательности команд, определяющих шаги, необходимые для достижения назначения программы.

Пролог является *декларативным языком*. Программа на декларативном языке представляет собой набор логических взаимосвязанных описаний, определяющих цель, ради которой она написана. Обозначения, используемые в Прологе для выражений логических взаимосвязей, унаследованы из логики предикатов.

Пролог освобождает программиста от составления программы в виде последовательности действий. В нем отсутствуют такие явные управляющие структуры, как DO WHILE, IF THEN и т.п. Пролог базируется на естественных для человека логических принципах.

В отличие от Паскаля, Бейсика и других традиционных процедурных языков программирования, где программист должен описать процедуру решения шаг за шагом, приказывая компьютеру, как решать проблему, при использовании Пролога программисту достаточно описать задачу и основные правила се решения, а Пролог сам определит, как прийти к решению. Прологу присущ ряд свойств, которыми не обладают процедурные языки программирования, что делает его мощным средством логического программирования. К таким свойствам относятся:

1) механизм вывода с поиском и возвратом;
2) встроенный механизм сопоставления с образцом;
3) простая, но выразительная структура данных с возможностью ее изменения.

Пролог отличается единообразием программ и данных. Данные и программы лишь две различные точки зрения на объекты Пролога. В единой базе данных можно свободно создавать и уничтожать отдельные элементы. Поскольку не существует различия между программой и данными, то можно менять программу во время ее работы. Естественным методом программирования является рекурсия.

Пролог является декларативным языком. Это означает, что, имея необходимые факты и правила, он может использовать дедуктивные выводы для решения задач программирования. Пролог - язык описаний и вместо серии шагов, определяющих, как компьютер должен решать проблему, программа на Прологе содержит описание проблемы, которое включает в себя:

1) Имена и структуры объектов, используемых в задаче;
2) Имена отношений (предикатов), существующих между объектами;
3) Основную цель (задачу), решаемую данной программой;
4) Факты и правила, описывающие отношения между объектами.

Такое описание используется для спецификации желаемых отношений между имеющимися входными данными и выходными данными, генерируемыми на основании входных. Первые две области описаний аналогичны секции объявлений программ на Паскале. Третья описывает формулировку основной цели программы. Четвертая содержит список логических утверждений в форме фактов, таких как, Иван любит Марью Петр любит пиво или в форме правил, например, Иван любит X, если Петр любит X (Иван любит то, что любит Петр). На основе имеющихся правил и

фактов Пролог делает выводы. Имея два вышеприведенных факта и одно правило. Пролог придет к выводу, что Иван любит пиво. А если задать Прологу задачу, Например, Найти всех, кто любит пиво, то он найдет все решения этой задачи. При этом он автоматически управляет решением задачи, стараясь во время выполнения программы найти все возможные наборы значений, удовлетворяющие поставленной задаче. При этом поиск производится по всей базе данных, ранее введенной в систему (термин база данных используется при объединении набора фактов для совместного их использования при решении некоторой конкретной задачи).

В Прологе используется метод поиска с возвратом, который позволяет в случае нахождения одного решения пересмотреть все сделанные предположения еще раз, чтобы определить, не приводит ли новое значение переменной к еще одному решению.

Программа на Прологе состоит из множества предложений (фраз). Каждое предложение - это либо факт, либо правило. Факт - это утверждение о том, что соблюдается некоторое отношение. Он записывается как имя, за которым следует список аргументов, заключенный в скобки. Например:
likes(“Иван", "Марья").

Правило - это факт, истинное значение которого зависит от истинности других фактов. Например:
likes("Иван",X) if likes("Петр",X).

Аргументы предложений Пролог-программы называются термами, а саму Пролог-программу можно рассматривать как сеть отношений, существующих между термами. Каждый терм обозначает некоторый объект предметной области и записывается как последовательность литер, которые делятся на четыре категории: прописные буквы, строчные буквы, цифры и спецзнаки. Существует три типа термов: константа, переменная или структура (составной терм).

Константами являются поименованные конкретные объекты или конкретные отношения. Существует два вида констант - атомы и числа. Атом - это либо последовательность латинских букв, цифр и знака подчеркивания, начинающаяся со строчной латинской буквы, либо произвольная группа символов, заключенная в кавычки (апострофы). Например: ivan, “Ivan", invoice_n, "иван", "n_счета", "Иван".

Переменная в Прологе должна иметь имя, начинающееся с прописной буквы или знака подчеркивания. Например: Name, X, Invoice_n. Переменная называется связанной, если имеется объект, который она обозначает. При отсутствии такого объекта переменная называется свободной. Для обозначения переменной, на которую отсутствует ссылка в программе, используется анонимная переменная - одиночный знак подчеркивания (_).

Структура (или составной терм) - объект, состоящий из совокупности других объектов, которые называются компонентами. Структура записывается на Турбо-Прологе с помощью указания се функтора и компонент. Компоненты заключаются в круглые скобки и разделяются

запятыми. Функтор записывается перед скобками. Компоненты сами являются термами. Например, в факт likes входит структура book:
likes ("Иван" , book ("название", "автор")).

Составные термы аналогичны записям Паскаля или структурам Си, то есть - это определяемые программистом объекты произвольной сложности. По этой же аналогии - функтор и количество компонентов составного терма показывают тип записи, а компоненты составного терма соответствуют полям записи.

Структура программ Турбо-Пролога

Существует множество реализации языка Пролог для разных классов вычислительных систем. В этом пособии за основу принимается программный продукт компании Borland International 'Турбо-Пролог" для IBM-совместимых компьютеров. Программа на Турбо-Прологе состоит из нескольких секций, каждая из которых идентифицируется ключевым словом и имеет следующую обобщенную структуру:

```
domains
        /* секция объявления доменов */
database
        /* секция объявления динамических баз данных */
predicates
        /* секция объявления предикатов */
goal
        /* подцель_1, подцель_2, и т. д. */
clauses
        /* предложения (факты и правила) */
```

Обязательным в программе является присутствие двух секций с именами predicates и clauses. В первой из них описываются структуры используемых в программе отношений, а во второй эти отношения; Для набора фактов и правил, рассмотренных выше, один из возможных примеров программы на Турбо-Прологе будет иметь вид:

```
/* программа 1 */

predicates
        likes (string, string) /* описание предиката */
clauses
likes ("Иван","Марья"). /* факт */
likes ("Петр","пиво").  /* факт */
likes ("Иван",X) if likes ("Петр",X). /* факт */
```

В этой программе предикат likes описывает структуру, отношения, домены которого имеют тип строки символов. Факты и правила записаны в виде предложении Пролог-программы, каждое из них заканчивается точкой, а текст, заключенный в /* ... */ - это комментарии. Отсутствие в программе

секции goal предполагает, что запросы к программе будут осуществляться из оболочки Турбо-Пролога. Так, например, при запросе о поиске всех любителей пива, цель должна быть сформировала в окне диалога Турбо-Пролога в виде:
Goal: likes(Who,"пиво").

В ответ на этот запрос в том же окне диалога Турбо-Пролог выведет всех тех, кто любит пиво, связывая их с переменной Who.

Однако использование в Пролог-программах только двух секций, т.е. predicates и clauses, является достаточным только для простейших программ, которые не используют описание данных и их структур, не работают с динамическими базами данных. Рассмотрим более подробно назначение каждой из секций программы.

Секция domains Пролог-программы

В секции domains объявляются любые нестандартные домены, используемые для аргументов предикатов. Домены в Прологе являются аналогами типов в других языках. Основными стандартными доменами Турбо-Пролога являются:

char - символ, заключенный в одиночные кавычки (например, 'a');

integer - целое от -32768 до 32767 (переводится в вещественное автоматически, если необходимо);

real - вещественное (например, -68.72, бе-94, -791e+21);

string - последовательность символов, заключенных в двойные кавычки (например, "нажмите ввод");

symbol - либо набор латинских букв, цифр и символов подчеркивания, в котором первый символ - прописная буква (например, n_fax); либо последовательность символов, содержащая пробелы или начинающаяся со строчной буквы, заключенная в кавычки (например, "Список СУБД").

file - символическое имя файла, которое начинается с прописной буквы.

Кроме стандартных доменов пользователь может использовать свои. Для этого в области объявления доменов можно использовать следующие форматы:

а) name = stanDom ,

где stanDom - один из стандартных доменов: int, char, real, string или symbol, name - одно или несколько имен доменов. Например, fio = symbol или year, height = integer;

б) mylist = elementDom* ,

где mylist - область, состоящая из списков элементов из области elementDom, которая может быть определена пользователем или иметь стандартный тип. Например, numberh5t = integer* или letter = char*;

в) myCompDom = functor1(d11,…,d1n); functor2(d21,..., d2n) ; ... functorm(dm1,...,dmq) ,

где myCompDom - область, которая состоит из составных объектов, описываемых указанием функтора и областей для всех компонентов. Правая часть такого описания может определять несколько альтернатив, разделенных "; « или "or". Каждая альтернатива должна содержать единственный Функтор functor, и описание типов для компонентов dij.Например,

auto = car (symbol, integer) ,packing = box (integer, integer, integer) ; bottle(integer)

описывает две области auto и packing. Область auto соответствует двухкомпонентной структуре с функтором car, а область packing соответствует одной из двух возможных структур box и bottle, которые различаются не только именами, но и количеством компонент.

г) file = name1; name2; ... namen

используется, когда пользователю необходимо ссылаться на файлы по их символическим именам.

Пример. Требуется разработать структуру данных для хранения информации о компьютерах. При этом каждый компьютер будет рассматриваться как набор входящих в него устройств, среди которых могут быть: процессоры с указанием их наименования и частоты, НЖМД с указанием их объема и фирмы изготовителя, также мониторы определенного типа. Область определения доменов для этого примера будет иметь вид

domains

name, firm, type = symbol

freq, vol = integer

Device = processor (name, freq); disk (firm, vol); monitor (type)

Computer = device*

В этом описании домен computer является списком элементов типа device, то есть каждый элемент этого списка может иметь структуру типа либо processor, либо disk, либо monitor, содержащих одну или две компоненты, каждая из которых имеет стандартный символьный ют целый тип.

Секция predicates

В секции predicates объявляются предикаты и типы (домены) аргументов этих предикатов. Имена предикатов должны начинаться со строчной латинской буквы, за которой следует последовательность букв, цифр и символов подчеркивания (до 250 знаков). В именах предикатов нельзя использовать символы пробела, минуса, звездочки, обратной (и прямой) черты. Объявление предикатов имеет форму:

predicates

predicateName1 (domen11, domen12,..., domen1m) predicateNamen (domenn1, domenn2,..., domennk)

Здесь domenij - либо стандартные домены, либо домены, объявленные в секции domains. Объявление домена аргумента и описание типа аргумента - суть одно и то же. Количество доменов (аргументов) предиката определяют арность (размерность) предиката. Предикат может не иметь аргументов и указываться только именем. Обычно выбирается такое имя предиката, чтобы оно отражало определенный вид взаимосвязи между аргументами предиката. Пример описания предикатов:

```
predicates
        Student (string, real)
        start
        good_student( string)
```

Можно использовать несколько описаний одного и того же предиката. При этом все описания должны следовать одно за другим и должны иметь одно и то же число аргументов. Пусть требуется определить отношение между тремя аргументами, первые два из которых соответствуют слагаемым, а третий - сумме двух первых. Этот предикат может быть описан в следующем виде

```
predicates
        add(integer,integer,integer)
        add(real,real,real)
```

и позволит его аргументам принимать значения, как из области целых, так и действительных чисел.

Секция clauses

В секции clauses размещаются факты и правила, с которыми будет работать Турбо-Пролог, пытаясь разрешить цель программы.

Факт - это утверждение о существовании некоторого отношения между аргументами, обозначаемого именем предиката. Факты - это фразы без условий и они содержат утверждения, которые всегда абсолютно верны. Форма записи фактов:

```
clauses
        predicateName1 (term11, term12,..., term1к).
        ...
        predicateNameN (termN1, terniN2,..., termNL).
```

где predicateNamel - имена предикатов, описанных в секции predicates , а term11, ... , term11 - аргументы предикатов (термы), количество которых должно соответствовать арности описания предиката. Пример фактов, определяющих отношение, заданное предикатом student, может иметь вид:

```
clauses
        student("Петров" ,4.5).
        student("Сидоров" ,3.75).
```

Правила содержат утверждения, истинность которых зависит от некоторых условий (подцелей), образующих тело правила. В Прологе правила представляются в виде

Заголовок:- Подцель1 , Подцель2 , ... , ПодцельN . ,
где обозначение:- читается как "если", а Заголовок имеет такую же форму, как и факт. Тело правила - это список подцелей, разделенных запятыми. Запятая понимается как конъюнкция и читается как "и". Пример записи правила
good_student(Name) :- student(Name , B) , B > 4.

Для того чтобы заголовок правила оказался истинным, необходимо, чтобы каждая подцель, входящая в тело правила, была истинной. Переменные в заголовке правила, квантифицированы универсально. Это означает, что правило, заголовок которого содержит переменные, будет истинным для любых термов, удовлетворяющих подцелям правила. С другой стороны переменные, входящие только в тело правила, квантифицированы экзистенциально. Если учесть такую квантификацию, то приведенное выше правило можно прочитать так:
Для любого лица Name, Name является хорошим студентом

ЕСЛИ	существует средний балл B такой, что Name является студентом со средним баллом B
И	средний балл B больше 4.

Множество правил, заголовки которых содержат одинаковые имена предикатов и количество аргументов, называются процедурой. В Прологе предикаты определяются (реализуются) при помощи процедур. Так следующие два правила
max(X , Y , X):- X >=Y.
max(X , Y , Y) :- X < Y.
реализуют процедуру нахождения наибольшего из двух чисел, определяемую предикатом вида max(number1, number2, max_number). Считается, что между этими двумя правилами неявно присутствует соединительный союз "или".

Секция goal

В секции goal задается внутренняя цель программы. Это позволяет программе запускаться независимо от среды разработки. Если внутренняя цель включена в программу, то Турбо-Пролог выполняет поиск только одного первого решения, и связываемые с переменными значения не выводятся на экран.

Если внутренняя цель не используется, то в процессе работы есть возможность вводить в диалоговом окне внешнюю цель. При использовании внешней цели Турбо-Пролог ищет все решения и выводит на экран все значения, связываемые с переменными.

В систему Турбо-Пролог включено более 200 встроенных стандартных предикатов и более дюжины стандартных доменов. В случае использования этих предикатов и доменов нет необходимости объявлять их в программе.

Рассмотрим пример программы, в которой задана внутренняя цель и используется обращение к стандартным предикатам:

```
/* Программа 2 */
predicates
        hello
goal
        hello.
clauses
        hello :-
        makewindow(l,7,7,"My first programm",4,54,10,22),
        nl, write("Please, type your name "),
        cursor(4,5),
        readln(Name) ,nl,
        write(" Welcome ",Name).
```

В этой программе формируется на экране окно, заданного размера и цвета, запрашивается Ваше имя, а затем оно выводится на экран. Перечень и назначение стандартных предикатов приведен в Приложении А.

Однако чаще всего целью является сложный запрос к программе. Для разрешения какой-либо сложной цели Пролог должен разрешить все его подцели, создав при этом необходимое множество связанных переменных. Если же одна из подцелей ложна. Пролог возвратится назад и просмотрит альтернативные решения предыдущих подцелей, а затем вновь пойдет вперед, но с другими значениями переменных. Этот процесс называется "поиск с возвратом".

Секция database

Ключевое слово database указывает на начало последовательности описаний предикатов динамической базы данных. Динамическая база данных является базой, в которую факты добавляются во время исполнения программы. Требования к описаниям предикатов такие же, как и в секции predicates. Факты, принадлежащие динамической базе данных, обрабатываются отличным от обычных предикатов образом для того, чтобы ускорить работу с БД большого объема. Факты динамической базы могут модифицироваться в течение сеанса работы, загружаться из дискового файла с помощью стандартного предиката consult или записываться в дисковый файл с помощью предиката save.

Оболочка системы Турбо-Пролог

Для ввода Пролог-программы и ее выполнения необходимо, прежде всего, загрузить в компьютер систему Турбо-Пролог. Для этого необходимо запустить на выполнение файл PROLOG.EXE (RP.EXE), после чего на экран дисплея будет выдано сообщение о загрузке системы программирования

Турбо-Пролог и ее конфигурации на вашем компьютере. Нажатие на клавишу <пробел> позволит войти в оболочку системы.

На экране отображается главное меню системы и четыре системных окна: окно редактирования, окно диалога, окно сообщений и окно трассировки (рисунок 1)

Рисунок 1 Оболочка системы Турбо-Пролог

Эти окна могут быть использованы в любой конфигурации, и любое из них может занимать либо целый экран, либо его часть. Нижняя строка экрана содержит сообщения о состоянии системы, описывая доступные команды и назначение функциональных клавиш. Назначение клавиш меняется при изменении режима работы.

Главное меню содержит набор команд и подчиненных иерархических меню. Для выбора нужной команды надо переместить к ней засветку и нажать «Ввод». Ниже приводится краткое описания назначения основных команд главного меню.

Команда "Редактировать" (Ред)

По этой команде вызывается встроенный в систему текстовый редактор. С его помощью можно вводить и редактировать текст программы. Если предварительно не было задано имя файла Пролог-программы, то по умолчанию принимается имя WORK.PRO. Методы работы с этим редактором такие же, как и с обычным текстовым редактором, а набор его команд близкий к стандартному набору команд для любых Турбо-систем. Перечень основных команд и комбинаций клавиш для вызова этих команд можно получить, нажав клавишу F1. Особенностью встроенного редактора является наличие в его составе дополнительного окна, позволяющего одновременно работать с двумя файлами и обмениваться между ними блоками информации. В частности, копировать и переносить блоки программного кода из одного файла в другой.

Команда "Выполнить" (Вып).

Команда "Выполнить" используется для выполнения откомпилированной программы, находящейся в памяти. При этом возможны две ситуации:

1) Если цель содержится внутри программы (то есть программа содержит секцию goal), то после выполнения команды Вып результат работы программы выводится в окне "Диалог". Нажатие клавиши <пробел> обеспечит возврат в главное меню системы;
2) Если секция goal в программе отсутствует, то после выполнения команды Вып активизируется окно "Диалог" и пользователь сам вводит цель или набор целей, общаясь с программой в интерактивном режиме через окно "Диалог". В ходе выполнения программы некоторые из функциональных клавиш имеют специальные значения:

F8 - выводит повторно предшествующую цель в окне "Диалог";
F9 - вызывает редактор;
Shift+F9 - выбирает системное окно, чтобы изменить его размеры;
Shift+F10 - изменяет, размеры или двигает окно "Диалог".
Ctrl+F10 - активное окно на весь экран, повторное нажатие стандартный размер окна Ctrl+P - перенаправляет выводной поток на принтер;
Ctrl+S - останавливает вывод на экран, повторное нажатие - продолжает; Ctrl +C или Ctrl+Break - прерывают исполнение программы.

Команда "Компилировать" (Компил)

По этой команде компилируется загруженная в окне редактора программа. Результат будет помещен на резидентный диск в .OBJ файл или. EXE файл, в зависимости от установки переключателя компиляции в меню "Режимы"

Меню "Файлы"

Выбор команды "Файлы" главного меню приводит к выводу на экран дисплея нового меню по работе с каталогами и файлами. Ниже перечисляются основные команды этого меню и их назначение.

"Загрузить" - загружает рабочий файл из PRO каталога. После выбора этой команды система запрашивает имя файла. Вы можете ввести:
1 любое допустимое для ДОС имя файла. Если расширение в имени файла опущено, система автоматически добавляет .PRO;
2 файл из каталога. Если на сообщение системы "Имя файла: « будет нажата клавиша ВВОД, то отобразится содержимое текущего (PRO) каталога и выбор файла осуществляется клавишами управления курсором.

"Сохранить" - сохраняет текущий файл на диске.

"Каталог" - используется для выбора каталога (PRO - но умолчанию, для остальных надо указать путь к каталогу).

"Имя файла" - используется для переименования рабочего файла. Удобна для сохранения предшествующей версии редактируемого файла.

"Вывод" - направляет программу или ее часть на принтер.

"Удалить" - уничтожает дисковый файл, имя файла может быть указано непосредственно, или выбрано из каталога.

"Переименовать" - переименовывает файл.

"Удалить файл из редактора" - удаляет текущий файл из редактора и засылает рабочий файл.

"Переход к ДОС" - вызывает временный выход в ДОС. Возврат из ДОС в Турбо-Пролог по команде exit.

Меню "Установки" (Устан)

Выбор **"Устан"** вызывает на экран дисплея меню конфигурирования и установок оболочки системы программирования. Ниже перечисляются основные команды этого меню и их назначение.

"Размер окон" - высвечивается меню с названиями текущих окон экрана. Можно выбрать окно и изменять его размеры с помощью клавиш управления курсором: «стрелка влево" и "стрелка вправо" - уменьшать и увеличивать ширину окна;

"стрелка вниз" и "стрелка вверх" - уменьшать и увеличивать высоту окна. Нажатие клавиши Ctrl u любой, из указанных выше, делает изменения быстрее.

Нажатие Shift и стрелки меняет позицию окна.

"Каталоги" - устанавливает текущие каталоги для разных типов файлов. В системе Турбо-Пролог используются каталоги:

- Каталог PRO - текущий по умолчанию для файлов с расширениями .PRO. " Каталог OBJ - используется для файлов с расширениями .OBJ u .PRJ.

- Каталог EXE - для файлов с расширением. EXE, генерируемых Турбо-Прологом.

- Каталог TURBO используется для самой системы Турбо-Пролог.

"Цвета" - выбрав окно, используйте "стрелку влево" и "стрелку вправо" для изменения атрибутов основного цвета, а "стрелку вверх" и "стрелку вниз" - для атрибутов символов. Результирующее значение атрибутов выводится в нижней части экрана.

"Разные установки" - используются для определения специальных параметров. Автозагрузка, по умолчанию OFF, используется, когда необходимо загрузить в память (Файл, содержащий сообщения об ошибках. Размер стека используется для переопределения размеров стека, по умолчанию 600 параграфов (1 параграф = 16 байт).

"Загрузка структуры" - загружается .SYS файл из TURBO каталога и можно изменять параметры системы, которые этот файл содержит;

"Запись структуры" - сохраняются текущие установки в .SYS файле.

Команда "Выход» (Вых).

По этой команде обеспечивается выход из системы Турбо-Пролог. До выхода из системы необходимо сохранить все измененные тексты Пролог-программ, а также файлы баз данных, использованные в данном сеансе работы с Турбо-Прологом.

1.2 Лабораторная работа 1

Тема: Работа с простейшими программами в системе Турбо-Пролог

Цель работы:

1 Знакомство со структурой Пролог-программ, использующих внешние или внутренние цели.

2 Получение навыков работы в оболочке системы Турбо-Пролога.

Знакомство с методикой отладки и трассировки программ.

3 Разработка простейших программ с использованием стандартных предикатов Турбо-Пролога.

4 Разработка простейших программ и их выполнение в среде системы Турбо-Пролог требуют начальных сведений по основным конструкциям языка Пролог и структуре Пролог-программ, которые должны содержать как минимум две секции:

- predicates - секцию описания структур отношений, используемых в программе, в виде предикатов,
- clauses -секцию определения предикатов в виде набора фактов и правил.

Если в программе описаны только эти две секции, то предполагается, что цель (или цели), решаемые программой будут формулироваться в интерактивном режиме работы внутри системы программирования.

Работа в системе программирования предполагает наличие у пользователя элементарных знаний по работе в Турбо-оболочках таких, как загрузка, ввод и редактирование программных файлов, запуск их на выполнение, а также отладка программ и конфигурирование системы.

Построение программ, не зависящих от среды выполнения, требуют включения в их структуру еще одной секции goal, обеспечивающей описание цели, решаемой программой.

При этом цель формулируется в виде запроса к программе, который представляет собой конъюнкцию подцелей. Для разрешения любой сложной цели Пролог должен разрешить все его подцели, создав при этом необходимое множество связанных переменных. Если же одна из подцелей ложна. Пролог возвратится назад и просмотрит все возможные альтернативные решения предыдущих подцелей, а затем вновь пойдет вперед, но с другими значениями переменных. Такой процесс называется "поиском с возвратом".

Для получения всех этих элементарных сведений по работе с простейшими программами на Прологе требуется познакомиться с материалами предыдущих разделов данного пособия.

Каждая программа на Прологе представляет собой текстовый файл, который для запуска из системы Турбо-Пролог должен иметь расширение .PRO.

Содержимое какого-либо текстового файла или программы на языке Пролог может быть включено в другую программу на режиме ее компиляции. Для такой текстовой подстановки режима компиляции используется директива компилятора include, которая имеет следующий синтаксис:

include "dos_file_name"

где dos_file_name - имя текстового файла системы DOS, включаемого в текущий программный файл. Имя файла может включать путь доступа к нему.

Включаемые файлы могут быть использованы только в естественных границах программы. Таким образом, ключевое слово include может появиться только там, где допускается одно из ключевых слов domains, predicates, clauses или goal.

Если в программе использована эта директивы, то при компиляции в данное место текущей программы будет вставлено содержимое другого текстового файла, как части текущей программы. Это позволяет некоторый набор уже отлаженных описаний и предикатов хранить в отдельном тестовом файле и включать его в другие программы при использовании в них обращений на эти, уже отлаженные, предикаты.

Включаемый файл сам может содержать директивы include. Однако включаемые файлы не должны использоваться рекурсивно. То есть так, чтобы тот же самый файл включался более чем однажды в процессе компиляции. Использование многих уровней включаемых файлов требует больше памяти в процессе компиляции, чем, если бы те же самые файлы были включены непосредственно в главную программу.

Загрузка системы Турбо-Пролог, ввод и запуск программ

Войдите в свой рабочий каталог и, находясь в нем, загрузите систему программирования Турбо-Пролог. Далее, пользуясь сведениями, изложенными в разделе 4 вводной части данного пособия:

1 ознакомьтесь с опциями главного меню и изучите их назначение;

2 установите режим компиляции в память компьютера;

3 сконфигурируйте, если необходимо, размеры и цветовые палитры всех окон;

4 определите пути доступа к файлам, установив нужные для вас каталоги;

5 выполненные установки запишите в файл конфигурации.

Войдите в режим редактирования системы Турбо-Пролог, воспользовавшись командой "Ред" главного меню, и введите программу 1. Закончив ввод, выйдите из редактора в главное меню системы, нажав клавишу "Esc".

Запустите программу на выполнение, выбрав команду "Вып" главного меню. В данной программе нет секции goal, т.е. в программе отсутствует внутренняя цель, определяющая решение конкретной задачи. Такие программы могут использоваться только в среде системы Турбо-Пролога. Поэтому, после ее запуска на выполнение системой активизируется окно "Диалог" и появляется приглашение на ввод внешней цели (GOAL:)

Работа с Пролог-программами в режиме диалога

Внешние цели - это запросы к программе, формируемые пользователем в окне "Диалог". Введите запрос
GOAL: likes(Who,"пиво").

Объясните: что обозначает данный запрос, к каким элементам языка Турбо-Пролога следует отнести такие объекты запроса, как "пиво". Who и likes.

Активизируйте введенный Вами запрос. Для этого надо после окончания его набора нажать клавишу "Enter". До нажатия "Enter" запрос можно редактировать. В ответ на Ваш запрос в окне "Диалог" должны появиться сообщения.
Who=Петр
Who=Иван 2
GOAL:

Объясните полученный результат и смысловое назначение выводимых в окне "Диалог" сообщений.

Система запоминает последний из введенных запросов. Для того, чтобы вызвать повторно предыдущий запрос, следует нажать функциональную клавишу F8. Вызовите повторно предыдущую цель и отредактируйте ее так чтобы она имела вид
GOAL: likes (Who,"пиво","Марья").
Запустите ее на выполнение и объясните полученный результат.

Аналогичные действия проделайте по вводу и запуску запроса вида:
GOAL: likes(Иван, X)

Объясните полученный результат, внесите изменения в запрос, чтобы он удовлетворял синтаксису языка Турбо-Пролог и повторно запустите запрос на выполнение.

Измените описание предиката так, чтобы было ясно, между какими объектами реального мира отношение likes устанавливается. В частности, для рассматриваемого примера, отношение likes определяется между некоторым лицом (person) и некоторым другим лицом или вещью (thing). Для учета введенного дополнения, измените в секции predicates описание предиката на новое likes(person,thing) и запустите программу на выполнение.

При этом система выдаст сообщение об ошибке и в окне редактирования курсором будет отмечено то место, где транслятор обнаружил ошибку. Текст сообщения об ошибке "Необъявленный домен или ошибка в написании" дает

подсказку о том, что, перейдя к использованию нестандартных (т.е. определенных пользователем) доменов мы забыли объявить их типы.

Так как областью изменения обоих, вновь определяемых, доменов являются символьные данные (точнее данные типа строки символов), то в программу должна быть добавлена секция domains, где должны быть объявлены нестандартные домены и их типы. Для данной программы она может иметь один из двух возможных видов:

```
domains
        person = stringthing = string
```

или

```
domains
        person,thing = string
```

Введите эти добавления в программу, запустите ее на выполнение и задайте любой, из ранее вводимых запросов. Результат должен соответствовать предыдущему.

Трассировка программ в среде системы Турбо-Пролога

Познакомиться с тем, как Пролог-система осуществляет поиск ответов на запросы, а также отследить последовательность согласования фактов и правил Пролог-программы можно, используя пошаговый режим ее выполнения. Для перехода в режим пошагового выполнения программы (трассировки) необходимо в программе использовать директиву trace.

Вставьте первой строкой программы директиву trace, которая при выполнении программы обеспечит трассировку всех предикатов. Запустите программу на выполнение и задайте один из ранее выполнявшихся запросов. Помните о возможности вызова предыдущего запроса с помощью F8.

Осуществите, с использованием клавиши F10, пошаговое выполнение программы, тщательно отслеживая все перемещения курсора в окне редактирования и регистрируя все выводимые в окне трассировки сообщения.

Разберитесь в последовательности доказательства цели Пролог-системой. Для этого записать все сообщения режима трассировки и дать им подробное толкование в отчете по лабораторной работе.

Работа с программами, содержащими внутреннюю цель

Измените программу таким образом, чтобы один из ранее использованных запросов, явился бы внутренней целью программы. Для этого в программу надо добавить еще одну секцию goal, где должна быть описана основная цель, решаемая программой. Пусть, для нашего примера, она будет иметь вид:

```
goal
        likes(Who,"пиво").
```

и аналогична запросу из 3.1 данной лабораторной работы.

Модифицированную программу запустите на выполнение. Если при запуске на решение у Вас появилось сообщение о синтаксической ошибке, при мигающем курсоре в соответствующем месте экрана, то посмотрите - не забыли ли Вы о том, что каждое предложение Турбо-Пролога должно заканчиваться точкой.

Если в программе отсутствуют синтаксические ошибки, то после запуска ее на выполнение в окне диалога появится сообщение "Нажмите ПРОБЕЛ", которое свидетельствует о том, что программа отработала и запрос выполнен. Но тогда возникает вопрос, а где же результаты запроса? А все дело в том, что в сформированной цели дается запрос о согласовании переменной Who с предложениями программы, а об отображении или выводе полученных результатов ничего не говорится.

В отличие от диалогового режима работы, когда внешняя цель формулируется в виде запроса к программе, а Турбо-система сама управляет процессом поиска и отображения результатов в некотором стандартном виде, при формировании внутренней цели все эти функции возлагаются на пользователя.

Формирование внутренней цели программы требует от пользователя не только задания запроса, но и обеспечения отображения его результатов на экране. Поэтому изменим цель, добавив еще одну подцель, обеспечивающую отображение терма на экране дисплея:

```
goal
    likes(Who,"пиво"), write(Who).
```

Запустим на выполнение программу, в которой цель представляет собой уже конъюнкцию двух подцелей. В результате выполнения этой программы в окне диалога выдается сообщение об одном из любителей пива,

```
Диалог
Петр
Нажмите ПРОБЕЛ
```

а после нажатия клавиши пробел, происходит остановка программы и выход в главное меню системы.

Связано это с тем, что при значении Who="Петр" каждая из подцелей принимает значение "истина" и вся цель становится истинной, что приводит к окончанию процесса вывода. Таким образом, мы обнаружили еще одно существенное отличие в формулировке одного и того же запроса в виде внешней или внутренней цели. Если при внешней цели система сама, управляя поиском, ищет все удовлетворяющие запрос ответы, то при использовании внутренней цели ищется только первый из них.

Если цель решения задачи должна полностью соответствовать запросу и ее требуется включить в тело программы, а домены отношения, для большей определенности, должны быть поименованы, то в этом случае программа 1 может быть представлена в виде:

```
/* Программа 3 */
domains
```

```
        person,thing = string
predicates
        likes(person,thing)
goal
        likes(Who,"пиво"), write(Who), nl, fail. /* цель */
clauses
        likes("Иван","Марья") . /* факт */
        likes("Петр","пиво") . /* факт */
        likes("Иван",X) :- likes("Петр",X) . /* правило */
```

В этой программе домены отношения likes имеют имена person и thing, которые имеют тип строки символов. Цель решения состоит из четырех подцелей, соединенных между собой запятыми. Запятая в предложениях Пролога равносильна логической функции "И" (and), т.е. цель решения задачи представляется конъюнкцией подцелей. Цель будет достигнута, т.е. примет значение "истина" (true), если каждая из подцелей будет истинной. Подцели данной программы содержат: один определенный пользователем предикат likes и три встроенных стандартных предиката Турбо-Пролога:

- write(Term) - выводит терм на дисплей,
- nl - обеспечивает переход на новую строку,- fail - вызывает состояние неудачи при доказательстве целевого утверждения.

Включение в цель дополнительных подцелей связано с тем, что задание цели внутри программы требует от пользователя не только формулирования запроса, но и обеспечения отображения его результатов на экране, а также обеспечение поиска всех удовлетворяющих запросу значений.

Так, если в цели отсутствует четвертая подцель, то будет найден только один любитель пива, а именно Петр. Поэтому добавление в цель предиката fail вызывает состояние неудачи при доказательстве целевого утверждения и переход к повторному его доказательству при иных значениях.

Отредактируйте Вашу программу до состояния программы 3 и запустите ее на выполнение. Если цель достигнута, сохраните программу в Вашем рабочем каталоге на диске под именем "LAB1_1.PRO"

Простейшая программа ввода-вывода данных

Рассмотрим еще один пример программы с внутренним описанием цели, которая демонстрирует простейшие возможности Турбо-Пролога по организации интерфейса с пользователем на основе использования стандартных предикатов Турбо-Пролога.

Загрузить программу 2 из раздела 3 вводной части данного пособия. Разобраться в ее структуре и запустить на выполнение. Используя приложение 1, познакомиться с синтаксисом, семантикой и назначением стандартных предикатов, используемых в данной программе.

Установив режим трассировки, ознакомиться с последовательностью выполнения программы и действием стандартных предикатов.

Модифицировать программу таким образом, чтобы окно создавалось в середине экрана и в другой цветовой палитре. Для этого по приложению 1 изучить описание предиката makewindow и воспользоваться приложением 2 по вычислению параметров цветовой палитры.

Исключить из программы внутреннюю цель, а описание и определение предиката hello изменить таким образом, чтобы можно было использовать внешнюю цель, задавая в режиме диалога, например, hello("Torn") или hello("").

Отлаженную по п.6.3. программу записать на диск с именем "LAB1_2.PRO".

Построение простейшего интерфейса для вывода результатов запросов

Если Вы правильно сформировали и хорошо отладили предикат hello(person) в программе "LAB1_2.PRO", то его можно использовать для вывода в окно любых, определенных в нем термов, что позволяет использовать его в качестве интерфейса для ранее разработанной программы "LAB1_1.PRO". Возможно два варианта совместного использования предикатов из этих двух программ:

- режим переноса в исходный модуль описаний и определений предиката путем копирования из другого файла;
- режим текстовой подстановки в исходный модуль файла, содержащего описание и определение требуемых предикатов.

В рамках данной лабораторной работы следует изучить оба этих варианта и составить две различные программы, реализующие поставленную цель.

Загрузить программу "LAB1_1.PRO". Войти в режим редактирования и используя режим "копия извне", вызываемый по нажатию клавиши F9, перенести в исходный файл описание и определение предиката hello из файла "LAB1_2.PRO". В исходную внутреннюю цель вставить в качестве первой подцели, например:

```
goal
    hello("Любители пива: nl"), ... , ... , ... , ... .
```

где последовательность символов nl - это стандартная константа "перевод строки".

Изменив текст программы, отладить ее и записать в файл "LAB1_3.PRO".

Загрузить программу "LAB1_1.PRO". Войти в режим редактирования и в первой строке программы ввести директиву:

```
include "LAB1_2.PRO"
```

В исходную внутреннюю цель в качестве первой подцели вставить другой предикат. Для совместной отладки основного и подгружаемого программных модулей использовать двухоконный режим работы редактора (F5 - вход в дополнительное окно редактирования, F10 - выход из него).

Выполнив необходимую модификацию, отладить программу и записать ее в файл "LAB1_4.PRO".

Исследуйте возможность применения пользовательского предиката hello() вместо стандартного предиката write() в разработанных программах.

Содержание отчета по лабораторной работе

Отчет по лабораторной работе должен содержать:

1 Результаты исследований по п.3.1.-п.3.5.
2 Протокол трассировки программы по п.4 и пояснения к ней.
3 Тексты программ LAB1_1.PRO, LAB1_2.PRO,. LAB1_3.PRO, LAB1_4.PRO.

Контрольные вопросы

1 Как производится запуск среды Turbo Prolog?
2 Какова структура среды Turbo Prolog?
3 Каким образом осуществляется вход в Turbo-среду?
4 Как создать и отредактировать программу в среде Turbo Prolog?
5 Как производится компиляция программы в среде Turbo Prolog?
6 Каким образом осуществляется выполнение программ в среде Turbo Prolog?
7 Как можно сохранить программу, подготовленную в Turbo-среде?
8 Как производится выход из среды Turbo Prolog?
9 Каково назначение функциональных клавиш среды Turbo Prolog?
10 Какие команды редактора Turbo Prolog вам известны?

1.3 Лабораторная работа 2

Тема: Пролог-программы как простейшие базы данных и знаний

Цель работы:

1 Знакомство с организацией баз данных как совокупности фактов.
2 Получение навыков организации явных и неявных баз данных.
3 Изучение способов построения универсальных запросов к базам.
4 Знакомство с представлением знаний в виде правил и процедур.

Система понятий для представления знаний несколько отличается от понятий для представления данных. Вместе с тем база знаний (БЗ) способна хранить данные и как простую разновидность знаний в виде базы данных (БД). Запросы, которые формирует пользователь к базе, реализуются одним из двух возможных способов:

- сообщения, являющиеся ответом на запрос, хранятся в явном виде в БД, и процесс получения ответа представляет собой выделение подмножества значений из БД, удовлетворяющих запросу;
- ответ не существует в явном виде в БД и формируется в процессе логического вывода на основании имеющихся данных.

Последний случай принципиально отличается от технологий использования БД и рассматривается в рамках представления знаний, то есть информации, необходимой в процессе вывода новых фактов. В Пролог-программах вывод новых фактов возможен на основании набора правил, включаемых в программу и представляющих собой упрощенный вариант БЗ. Представление знаний в виде набора правил имеет следующие преимущества:

- простота создания и понимания отдельных правил;
- простота механизма логического вывода.

К недостаткам этого способа организации БЗ относится его отличие от человеческой структуры знаний.

Запросы к базе данных

Простейшая Пролог-программа представляет собой множество фактов, которое неформально называют базой данных.

Рассмотрим пример. Пусть для хранения информации о служащих и местах их работы необходимо создать БД со структурой отношения РАБОТАЕТ (ИМЯ, ОТДЕЛ). При этом атрибут ИМЯ описывает домен данных типа строки символов, а атрибут ОТДЕЛ - домен целочисленных данных.

```
/* Программа 4 */
domains
        name = string office = integer
predicates
        work( name , office )
clauses
        work( "Петров" , 101 )
        work( "Павлов" , 211 )
        work( "Сидоров" , 101)
```

Для решения поставленной задачи в Пролог-программе:

- исходное отношение описывается предикатом аналогичной структуры;

- в секции domains задаются области изменения каждого аргумента предиката;

- каждый кортеж данного отношения представляется в секции clauses в виде факта.

Одним из примеров программы, реализующих данную задачу, может быть приведенная здесь программа 4.

После того, как данная программа введена в систему Турбо-Пролога с помощью текстового редактора или загружена с диска, а затем запушена на выполнение командой Вып главного меню системы, активизируется окно

диалога (появляется сообщение Goal:) и система готова к приему запросов. Синонимом слова запрос является слово цель. В переводе с английского goal обозначает цель.

. Простые запросы

Простой запрос состоит из имени предиката, за которым располагается список аргументов.

Если в запрос входят только константы (т.е. атомы и числа), то такой запрос называется запросом с константами и на него система выдает только один из двух ответов - True или False. Ответ True свидетельствует о том, что система доказала истинность запроса в соответствии с множеством фактов, загруженных в нее в данный момент. Ответ False - это невозможность системы доказать истинность запроса. Приведенные в окне диалога запросы соответствуют вопросам:

- "Работает ли Павлов в 211 отделе?". Ответ на него можно трактовать так: "Да, работает Павлов в 211 отделе"
- "Работает ли Павлов в 101 отделе?". Ответ на который: "Нет, не работает Павлов в 101 отделе".

Однако применение запросов с константами весьма ограничено, поэтому наиболее часто применяются запросы, которые используют переменные - запросы с переменными.

Переменная - это вид терма, начинающийся с заглавной буквы. В запросе, содержащем переменную, неявно спрашивается о том, существует ли хотя бы одно значение переменной, при котором запрос будет истинным.

Т.е. переменные в запросах квантифицированы экзистенциально. Если это иметь в виду, то приведенный в окне диалога запрос можно прочесть так: "Существует ли хотя бы один человек, который работает в 101-м отделе?". Запрос будет истинным, если такое лицо будет найдено в текущей базе.

Система пытается унифицировать (т.е. согласовать) аргументы запроса с аргументами фактов, входящих в базу данных "work". Запрос окажется успешным при его сопоставлении с первым же фактом, поскольку атом "101" в запросе унифицируется с атомом "101" первого факта, а переменная "Who" унифицируется с атомом "Петров", входящим в этот факт. В результате данного процесса переменная "Who" примет значение атома "Петров", сообщение о чем и выводится в окне диалога.

Далее происходит сопоставление запроса с другими фактами БД и о всех успешных унификациях и их количестве выдается сообщение в окне диалога. Говорят, что переменная конкретизируется, когда при выполнении запроса она унифицируется с некоторым значением.

Составные запросы

Составные запросы образуются из простых, соединенных между собой запятыми. Каждый простой запрос называется подцелью. Для того, чтобы составной запрос оказался истинным, необходимо, чтобы каждая из его подцелей была бы истинной. Введем составной запрос, который будет соответствовать вопросу: "Есть ли такой отдел, где вместе работают Петров и Сидоров?"

```
Goal: work("Петров",X),work("Сидоров",X)
X=101
1
Goal:
```

Переменная X входит в обе подцели, т.е. для истинности всего запроса требуется, чтобы вторые аргументы в каждой из подцелей принимали одни и те же значения.

Использование констант в аргументах запроса эквивалентно заданию входных параметров программы. Использование переменных - эквивалентно требованию получить от программы выходные данные.

Запросы с анонимными переменными

Символ подчеркивания (_) выступает в качестве анонимной переменной, которая предписывает системе проигнорировать значение аргумента. Эта переменная унифицируется с чем угодно, но не обеспечивает выдачу на экран данных. Каждая анонимная переменная, входящая в запрос, отличается от других переменных этого запроса.

Посмотрим, что будет, если ввести запрос work(Worker,_). При выполнении этого запроса будут выданы все возможные значения первого аргумента предиката work(...), вне зависимости от того, какие значения будет принимать второй аргумент. Таким образом, данный запрос соответствует вопросу:

"Существует ли хотя бы один служащий, который работает в каком-либо из отделов?"

Сформированный к программе запрос аналогичен желанию получить данные о всех служащих, работающих во всех отделах фирмы.

```
Goal: work(Worker,_)
Worker="Петров"
Worker="Павлов"
Worker="Сидоров"
3
Goal:
```

Статические и динамические базы данных

Как уже отмечалось, множество фактов Пролог-программы можно рассматривать как базы данных, к которой могут формулироваться любые произвольные запросы. Однако данные такой БД жестко связаны с самой Пролог-программой. Любое манипулирование с БД требует изменения или добавления того или иного факта в текст программы. В связи с этим такие базы данных называют статическими БД.

Так, для ввода информации о поступлении на службу в 101 отдел Иванова, нам потребуется войти в режим редактирования программы и добавить новый факт.

```
/* Программа 4 */
. . . .
clauses
        . . . .

        work( "Петров" , 101 ).

Goal: work(Worker,_)
        Worker="Петров"
        Worker="Павлов"
        Worker="Сидоров"
        Worker="Иванов"
        4
Goal:
```

Если затем в режиме выполнения мы зададим запрос о сотрудниках всех отделов, то получим информацию уже о четырех служащих. Причем обратите внимание, что в ответе на запрос данные об Иванове будут выводиться в той, по номеру, строке, каким по счету в программе является факт о работе Иванова в 101 отделе.

Попробуйте, войдя в режим редактирования, переставить факты местами. Затем, войдя в режим выполнения, сформулировать запрос к базе. Заметьте, что система находит ответы в БД в том порядке, как они введены в программе. Это еще одна особенность работы со статическими БД.

Динамические базы данных - это БД, в которых факты могут модифицироваться во время выполнения программы или выбираться из файла. Для работы с такими БД в тексте программы в секции database должны быть декларированы предикаты для описания структуры динамической БД. Работа с такими БД будет описана ниже.

Явные и неявные базы данных. Правила логического вывода

Рассмотренная выше база данных о местах работы служащих является явной БД, так как она составлена из фактов, аргументы которых константы. Работая даже с такой простой БД, у человека хватает интеллекта установить

между имеющимися данными, и совсем иные отношения, кроме работы служащего в конкретном отделе.

Так человеку хватает знаний на то, чтобы назвать двух служащих коллегами, если они работают в одном и том же отделе. Вместе с тем, работая с БД work(...) , для определения коллег Петрова пользователю системы необходимо:

- определить номер отдела (N), в котором работает Петров:
- найти всех сотрудников, работающих в отделе с номером N;
- исключить из полученного списка самого Петрова;
- сформировать все сочетания Петрова с лицами из полученного списка.

Таким образом, у пользователя появляется возможность свои представления о понятии коллега сформировать в виде составного запроса к явной БД и получить ответ на интересующий его вопрос. И такая ситуация будет повторяться при каждом использовании понятия коллега при формировании запросов, таких как "Являются ли Петров и Павлов коллегами?", "Кто коллега Петрова и Сидорова?" и т.д. Ясно, что такая процедура длительна, требует от пользователя системы должной квалификации по составлению составных запросов.

Вместе с тем эти несложные познания по преобразованию отношений и своп представления о понятии коллега пользователь может передать" Пролог-программе, описав свои знания набором декларативных правил.

Для этого можно ввести новое отношение КОЛЛЕГА (СЛУЖАЩИЙ1 , СЛУЖАЩИЙ2), определив его как пару разных служащих, работающих в одном и том же отделе. Новое отношение следует описать в секции predicates двуместным предикатом со структурой, аналогичной отношению КОЛЛЕГА(...).

Аргументы предиката должны принимать значения из области символьных данных. А сам предикат в секции clauses определяется на основании логического правила вывода:

Любые два служащих Служащий1 и Служащий2 являются коллегами
ЕСЛИ
существует такой отдел N_отд , что работает Служащий1 в отделе N_отд
И
работает Служащий2 в отделе N_отд
И
служащие Служащий1 и Служащий2 различны.

Выполнив все необходимые действия по модификации исходной программы, мы получим программу, которая содержит не только факты, но и знания, в виде правил манипулирования этими фактами. Это позволяет к правилам применять все возможные типы запросов, что и к фактам.

Если нас интересуют все коллеги служащего Петрова, то запрос надо задать в виде:

Goal: colleague ("Петров", Who),

а если интересуют общие коллеги и Петрова и Иванова, то запрос будет иметь вид:
Goal: colleague ("Петров",X), colleague ("Иванов",X)
и система найдет только один ответ о служащем по фамилии Сидоров, так как именно он работает в одном отделе с Петровым и Ивановым.

```
/* Программа 5 */
domains
        name = string
        office = integer
predicates
        work( name , office )
clauses
        colleague(Manl,Man2) :- work(Man1,X)
        work(Man2,X),
        Man1<>Man2.
        work( "Петров" , 101 ).
        work( "Павлов" , 211 ).
        work( "Сидоров" , 101 ).
        work("Иванов" , 101).
```

Из изложенного следует, что если БД work(...) - это явная база данных, так как составлена из фактов, аргументы которых константы, то база colleague(...) - это неявная база данных, поскольку правило для ее формирования определено с использованием переменных, значения которых зависят от подцелей этого правила. С точки зрения пользователя, формирующего запрос, не имеет значения, является база явной или неявной.

Использование структур в качестве доменов отношений

Турбо-Пролог позволяет создавать объекты, компонентами которых являются другие объекты. Причем сложные объекты рассматриваются и обрабатываются так же, как и простые. Это сильно упрощает составление программ и организацию баз данных.

В предыдущем разделе мы сформировали неясную базу данных о коллегах па основе наших представлений о понятии коллега. При этом, формулируя правила для представления этого понятия, мы ограничились только представлением о коллегах, как о людях, которых объединяет совместный труд, т.е. как о сослуживцах.

Более точно определить понятие коллеги можно, как пару лиц, объединенных единой целью, интересом, предметом деятельности и т.д. При такой формулировке понятия коллеги мы можем для его реализации сформировать отношение
ОБЬЕДИНЯЕТ (ЛИЦ01 , ЛИЦ02 , ПРЕДМЕТ)
или в синтаксисе языка Турбо-Пролога
unite(name , name, object)

И тогда мы могли бы, например, называть людей коллегами, если:
- объединяет Тома и Билла работа
- объединяет Сидорова и Петрова общее хобби, которым является спорт
- объединяет Петрова и Тома проект по новым системам для IBM
- объединяет Козлова и Сидорова совместная трудовая деятельность

У нас не возникнет никаких сложностей при представлении первого предложения в виде факта на Прологе, который будет иметь вид
unite(tom , bill, labour)
Но если второе предложение записать в аналогичной форме, т.е.
unite('"Сидоров" , "Петров", sport) ,
то оно не будет соответствовать действительности, так как из него совсем не следует, что спорт является общим увлечением двух лиц, т.е. является их хобби. Более того, оно ложно, так как из него можно заключить, что Петров и Сидоров профессиональные спортсмены, объединенные общими спортивными делами. Не верным будет и наше решение в качестве третьего аргумента указать
unite("Сидоров" , "Петров", "hobby sport") ,
так как hobby является некоторым свойством объекта совместной деятельности, a sport является конкретным значением этого свойства. Т.е. hobby - это атрибут объекта object , a sport - конкретный экземпляр этого атрибута. При таком подходе единственный вариант записи второго предложения будет
unite("Сидоров" , "Петров", hobby(sport)) ,
где hobby(sport) - это составной терм или структура Турбо-Пролога. Тогда по аналогии можно записать пролог-факты для второго и третьего предложений
unite("Петров", tom , project("New system», IBM)) ,
unite("Козлов","Сидоров" , labour) ,

Таким образом, можно сделать вывод о том, что во введенном для определения понятия коллеги отношении unite первые два домена являются простыми объектами, а третий - это сложный объект, атрибуты которого сами являются объектами.

Описание данного отношения на Прологе в виде предиката и определение областей изменения его аргументов будет иметь вид:

```
domains
        name,firm = symbol
        object = labour ; hobby(name) ; project(name,firm)
predicates
        unite( name , name, object )
```

где символ ";" (точка с запятой) эквивалентен логической операции "ИЛИ" и в данном описании использован для того, чтобы показать, что домен object может иметь одну из возможных структур, описанных для него в области domains.

Задание 1

1 Откорректируйте программу 5, включив в нее описание предиката unite и определив его для четырех фраз, приведенных в данном разделе.

2 Сформируйте запросы, соответствующие вопросам:
- "Кого объединяет совместный труд?",
- "Есть ли пара любителей шахмат?",
- "Кто является коллегой Тома?"
- "Для кого и с кем Петров выполняет проект?"
- "Кто является коллегами Тома?" (если есть сложности, то вспомните о логической операции дизъюнкции и ее использовании в Прологе).

3 Сформулируйте самостоятельно еще три составных запроса к базе и введите в программу.

4 Все вопросы, соответствующие им запросы и результаты вывода представить в отчете по лабораторной работе.

Процедуры как элемент представления знаний

Смысл предложений Пролог-программ может быть понят либо с позиций декларативного подхода, либо с позиций процедурного подхода. Декларативный смысл подчеркивает статическое существование отношений. Порядок следования подцелей в правиле не влияет на декларативный смысл этого правила.

При процедурной трактовке программы подчеркивается последовательность шагов, которые выполняются при обработке запроса. В этом случае, приобретает значение порядок следования подцелей в правиле.

Множество предложений, имеющих одно и то же имя предиката с одинаковым количеством аргументов, называют процедурой. Когда обрабатывается запрос к процедуре, то он анализирует фразы, образующие процедуру, в том порядке, как они в ней представлены. Считается, что между правилами процедуры неявно присутствует соединитель '"или".

В предыдущих разделах мы сформировали два подхода к представлению наших знаний о понятии коллега. С одной стороны коллегами являются сослуживцы, т.е. любая пара лиц, которые работают вместе, с другой - любая пара лиц, которая объединена объектом общей деятельности. Мы подошли к тому, чтобы два наши знания о понятии коллега объединить в одно.

Представьте себе, что одну из формулировок какого-либо понятия (в нашем случае это коллега) мы получили от одного эксперта, имеющего свои знания этой проблемы, а вторую - от эксперта, имеющего иные знания этой же проблемы. Объединяя в Пролог-программе два знания одной и той же проблемы, мы получим систему, которая задает больше каждого отдельного эксперта.

Понятие коллеги, удовлетворяющее обеим точкам зрения, может быть описано отношением вида:

ПОЛНЫЙ_КОЛЛЕГА (ЛИЦ01, ЛИЦ02 , ПРЕДМЕТ ОБЩЕЙ_ДЕЯТЕЛЬНОСТИ), которое на Прологе будет описано предикатом all_colleague, структура которого будет полностью аналогична структуре предиката unite, а определить его можно в виде процедуры, содержащей три декларации предиката all_colleague.

```
predicates
        all_colleague( name , name, object ) clauses
        all_colleague(X,Y,Z) :- colleague(X,Y), Z=labour.
        all_colleague(X,Y,Z) :- unite(X,Y,Z).
        all_colleague(X,Y.Z) :- unite(Y,X,Z).
```

С декларативной точки зрения это описание процедуры полный_коллега можно прочитать так.

Для любых двух лиц X и Y и любой общей деятельности Z
X и Y являются коллегами по общей деятельности Z
ЕСЛИ
X и Y являются сослуживцами
И общая их деятельность Z - это труд
ИЛИ X объединяет с Y общая деятельность Z
ИЛИ
Y объединяет с X общая деятельность Z

Последнее правило устраняет асимметрию отношения unite по отношению к лицам, объединенным общей деятельностью. Действительно, если Козлов является коллегой Сидорова по работе, то, очевидно, что и Сидоров является кометой Козлова по работе.

Задание 2

1. Измените программу, добавив в нее описание предиката all_colleague и процедуру для его определения.
2. Определите состав явных и неявных баз, используемых в данной программе и опишите их структуру.
3. Введите запрос "Кто является коллегой Тома?". Третье правило процедуры заключите в /*...*/ и повторите запрос. Объясните, почему разные ответы.
4. В чем заключается разница в выполнении запросов unite("Петров",Who,X) и all_colleague("Петров",Who,X). Введите еще ряд произвольных запросов.
5. Определите, кто является коллегой Козлова и кто - коллега Сидорова.
6. Все вопросы, запросы и результаты вывода привести в отчете по работе.

Целостность и непротиворечивость баз данных и знаний

С этими двумя сложными понятиями, одними из основных при построении баз данных и знаний, мы постоянно будем оперировать дальше.

Здесь же остановимся лишь на одном небольшом примере, иллюстрирующем их важность.

При выполнении п.5 задания 2 мы определили, что у Козлова только один коллега - Сидоров, связанный с ним совместным трудом. Вместе с тем у Сидорова кроме Козлова есть еще два коллеги, которые связаны с ним совместным трудом.

Но из этих двух посылок и наших представлений о понятии коллеги любому человек ясно, что Козлов работает в том же отделе, что и Сидоров. А если это так, то и имеет он более одного коллеги, в отличие от ответа системы. Т.е. у нашей системы не хватает интеллекта на такой вывод.

А на запрос work("Козлов",office) система вообще даст отрицательный ответ. Налицо противоречивость данных. Частично исправить ситуацию можно, если доопределить предикат work в виде

work(Man1,N) :- unite(Man1,Man2,labour), work(Man2,N).

Тогда на запрос о номере отдела у Козлова и его коллегах система будет давать более точные ответы. Но ведь в базе work() отсутствуют данные о Козлове в виде фактов, т.е. в явном виде. Стало быть после нашего доопределения эта база стала не совсем явной, так как часть данных хранится в явном виде, а часть выводима из других на основе правил. В первом приближении - это уже прообраз базы знаний. Текст программы 5 со всеми добавлениями, введенными по ходу работы, имеет вид:

```
/* программа 6 */
domains
        name,firm = symbol
        office = integer
        object = labour;hobby(name);project(name,firm)
predicates
        work( name , office )
        colleague( name , name )
        unite( name , name ,object )
        all_colleague( name , name , object )
clauses
        colleague(Man1,Man2) :- work(Man1,X), work( Man2,Y), Man1<>Man2.
        all_colleague(X,Y,Z):- colleague(X,Y), Z=labour.
        all_colleague(X,Y,Z) :- unit(X,Y,Z).
        all_colleague(X,Y,Z) :- unit(Y,X,Z).
        unite(tom,bill,labour).
        unite("Сидоров","Петров",hobby(sport)).
        unite("Петров",tom, project("New system», IBM)
        unite("Козлов","Сидоров",labour)
        work( "Петров" , 101 ).
        work( "Павлов" , 211 ).
        work( "Сидоров" , 101 ).
        work( "Иванов" , 101 ).
        work(Man1,N) :- unite(Man1,Man2,labour),work( Man2,N).
```

Содержание отчета по лабораторной работе

Отчет по лабораторной работе должен содержать:

1. Текст программы с набором фактов, который применяется при запросах.
2. Запросы, сформированные самостоятельно при изучении п.2 и п.3 данной лабораторной работы, а также результаты их выполнения.
3. Результаты выполнения задания 1 и 2 данной лабораторной работы.
4. Результаты выполнения индивидуального задания, выданного преподавателем.

Контрольные вопросы

1. Что такое факт в Turbo Prolog,
2. Что такое запрос в Turbo Prolog?
3. Какие разделы могут присутствовать в программе на языке Turbo Prolog?
4. Какие виды переменных используются в программе на языке Turbo Prolog и для чего?
5. Для чего используется трассировка?

1.4 Лабораторная работа 3

Тема: Управление ходом выполнения программ в системе Турбо-Пролог.

Цель работы:

Познакомиться с процессами унификации и помеха с возвратом.

Изучить правила унификации термов.

Изучить работу системы Турбо-Пролог при выполнении запроса.

Ознакомиться с методом отката после неудачи.

Работа системы Турбо-Пролог при выполнении запросов

Запрос к системе - это всегда последовательность, состоящая из одной или нескольких целей. Для ответа на запрос система пытается достичь всех целей, т.е. показать, что утверждения, содержащиеся в запросе, истинны в предположении, что все отношения программы истинны. Другими словами, достичь цели - это показать, что она логически следует из фактов и правил программы, а если в запросе есть переменные, то еще и конкретизировать их, т.е. найти те конкретные объекты, которые, будучи подставленные вместо переменных, обеспечат достижение цели.

Для этого система опускается в структуру программы так глубоко, как это необходимо, чтобы найти факты, требующиеся для доказательства истинности запроса. Затем система вернется в исходное состояние, доказав

или оказавшись не в состоянии доказать истинность запроса. В основе работы системы лежит рекурсивный циклический процесс унификации (т.е. сопоставления с образцом) и доказательства подцелей.

При задании запроса система просматривает всю программу в поисках первого предложения, заголовок которого будет унифицироваться с запросом. Для того, чтобы запрос и заголовок предложения унифицировались между собой, необходимо:

- совпадение у них имени предиката;
- совпадение количества аргументов предиката;
- возможность унификации всех аргументов предиката (правила унификации термов приведены ниже).

Если предложение, унифицирующееся с запросом, будет обнаружено, то оно начинает обрабатываться:

- если тело предложения является пустым (т.е. это факт), то запрос сразу оказывается успешным, переменные запроса конкретизируются объектами факта, и это решение помечается указателем возврата.
- если тело предложения содержит подцели, то, последовательно слепа направо, каждая из них обрабатывается точно так же, как исходный запрос.

Если система в тексте программы не находит предложения, унифицирующегося с запросом, то она:

- возвратится к последней успешно доказанной подцели;
- ликвидирует конкретизацию всех переменных, явившуюся результатом успешной обработки этой подцели, т.е. делает переменные свободными;
- приступает к поиску во множестве предложений текущей программы; заголовка другого предложения, унифицирующегося с данной подцелью.

Такая процедура обработки запроса получила название поиск с возвратом. Применяя ее, при успешном выполнении запроса система выдает:

- либо значения всех переменных, входящих в состав запроса, которые были конкретизированы в результате обработки;
- либо слова да или кет, если переменные в запросе отсутствовали.

Если запрос был сформирован как внешняя цель решения задачи, то система после найденного первого ответа вернется в точку, помеченную указателем возврата и попытается найти иной ответ.

Унификация термов

Наиболее важной операцией над термами является унификация, при которой осуществляется: конкретизация переменных, доступ к структурам данных через общий механизм согласования и определенные виды тестов на равенство. Процесс унификации соответствует передаче параметров в других

языках программирования. Унификация термов регулируется следующими правилами:

Свободная переменная унифицируется с константой или структурой. В результате этого переменная становится конкретизированной, т.е. принимает значение этой константы или структуры.

Goal: X = john
X = john

Переменная унифицируется с переменной, при этом обе они становятся одной и той же переменной.

Goal: X=Y
X=_1
Y=_1

Анонимная переменная унифицируется с чем угодно.

? _ = john

Константа может быть унифицирована с другой константой, если они идентичны или со свободной переменной соответствующего типа.

Goal: "джон" = "джон"
True

Структура унифицируется с другой структурой при условии, что их функторы одинаковы, а аргументы могут попарно унифицироваться.

? father(X) = father(john)
X = john

Для того, чтобы познакомиться с процессом унификации различных классов объектов Пролога, рассмотрим выполнение программой 7 нескольких запросов.

```
/* Программа 7 */
domains
title , author = symbol
pages = integer
publication = book(title , pages)
predicates
written_by(author,publication)
long_novel(publication)
clauses
written_by("И.Братко", book("Программирование на языке Пролог",560)).
written_by("Ц.Ин,Д.Соломон" ,book("Использование Турбо-Пролога",608)).
long_novel(book(Title,Lcngth)) :- written_by( _ , book(Title, Length)), Length > 600.
```

Рассмотрим запрос вида: written_by(X,Y) . При решении задачи система должна поочередно согласовать цель с предложениями программы, пытаясь достичь соответствия между параметрами X и Y с одной стороны и параметрами предложений программы с другой, выполняя операцию унификации.

Так как в данном запросе переменные X и Y являются свободными и могут согласовываться с любой константой, то самое первое предложение для предиката written_by дает желаемое соответствие:
written_by(X, Y) - written_by("Н.Братко" , book(" Программирование на языке Пролог",560)).
т.е. X конкретизируется константой "И.Братко", а Y принимает значение структуры book("Программирование на языке Пролог" ,560). Система Турбо-Пролог помечает эту точку указателем возврата и выдаст на экран сообщение.
X = "И.Братко" Y = book(" Программирование на языке Пролог" ,560)

Так как мы задавали запрос как внешнюю цель, система возвращается в точку, помеченную указателем возврата и продолжает, начиная с этой точки, процесс унификации и находит второе предложение, которое также может быть согласовано с запросом. После унификации переменных система выдает на экран X = "Ц.Ин, Д.Соломон" Y = book("Использование Турбо-Пролога",608) 2 и заканчивает процесс унификации.

Если введем запрос written_by(X,book("Использование Турбо-Пролога",Y)), то попытка унификации переменных с первым предложением программы будет выглядеть так:
written_by(X, book("Использование Турбо-Пролога", Y) - written_by("И.Братко" , book("Программирование на языке Пролог" ,560)).

Так как X свободна, то она принимает значение константы "И.Братко", и делается попытка установить соответствие между двумя структурами. Но составной объект согласуется с другим составным объектом при условии, что они оба имеют один и тот же функтор, одинаковое количество аргументов, и все аргументы могут быть попарно унифицированы. Однако, константа "Использование Турбо-Пролога" может быть унифицирована только со свободной переменной или сама с собой. Так как между первыми двумя компонентами структуры book соответствие невозможно, то формируется признак неудачи, и система пытается согласовать цель со следующим предложением программы, переходя к проверке соответствия между:
written_by(X, book(" Использование Турбо-Пролога", Y)) - written_by("Ц.Ин, Д.Соломон " ,book(" Использование Турбо-Пролога",608)).

Свободная переменная X унифицируется с константой "Ц.Ин, Д.Соломон". Обе структуры имеют один и тот же функтор book, содержат равное число компонентов, и первые компоненты обеих структур - одинаковые константы. Т.е. эти структуры могут быть унифицированы, и при этом константа 608 унифицируется с переменной Y. Т. е. цель достигнута, и Турбо-Пролог выводит сообщение:
X = "Ц.Ин, Д.Соломон" Y = 608

Наконец, рассмотрим выполнение запроса: long_novel (X). Прежде всего система пытается отыскать предложения, заголовки которых согласуются с запросом.
long_novel(X) - long_novel(Title) :-written_by(_ , book(Title,Length)) , Length > 600.

После этого согласовываются левая и правая части правила. Переменные X и Title согласуются, т.к. они свободны и становятся одной и той же переменной. Затем Турбо-Пролог объявляет первое предложение указанного выше правила подзадачей и делает попытку ее унификации:
written_by(_,book(Title, Length)) - written_by("И.Братко" ,book(" Программирование на языке Пролог", 560)).

Так как анонимная переменная согласуется с любым объектом и структуры book также согласуются, то эти два предиката могут быть унифицированы. В результате этого переменная Title принимает значение "Программирование на языке Пролог", а переменная "Length' становится равной 560.

После этого делается попытка согласовать вторую подзадачу тела правила, а именно: Length > 600. Перед попыткой унификации связанная переменная Length заменяется своим численным значением 560. Так как выражение: 560 > 600 ложно, то Турбо-Пролог делает возврат назад к уже доказанной подцели и пытается его передоказать. Т.е. снова пытается унифицировать первую подзадачу
written_by(_ ,book(Title,Length)),
используя следующий из имеющихся фактов
written_by(_,book(Title,Length)), используя следующий из имеющихся фактов
written_by(_,book(Title, Length))
written_by("Ц.Ин, Д.Соломон " ,book(" Использование Турбо-Пролога", 608)).
который связывает Title с "Использование Турбо-Пролога" и Length с 608. В данном случае оказывается: Length > 600, т.е. вторая подзадача также становится истинной. Правило полностью согласовано при полученных значениях переменных, задача решена, о чем выдается сообщение:
X = "Использование Турбо-Пролога"

Исходная цель является полностью доказанной, и функционирование системы Турбо-Пролог по выдаче ответа на введенный запрос заканчивается. Система готова к приему новых запросов.

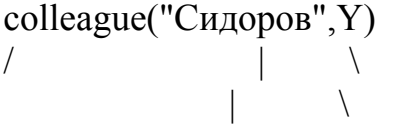
Поиск с возвратом при выполнении Пролог-программ

Знание метода поиска с возвратом, реализованном в Пролог-системе, позволит Вам правильно составлять Пролог-программы и управлять поиском решений. Более подробно рассмотрим этот процесс на примере программы б из лабораторной работы 2.

Задание 1

Загрузите в систему Турбо-Пролог программу б из предыдущей лабораторной работы, войдите в режим редактирования и введите директиву пошагового выполнения программы. Для сокращения пространства поиска последнее правило в базе данных work(...) сделайте комментарием.
Сформируйте запрос о поиске всех сослуживцев Сидорова и в режиме пошаговой трассировки отследите процесс поиска решений.

Решая любую задачу. Пролог строит дерево подзадач, отмечает, какие подзадачи решены, а какие еще нет. Так при решении поставленной в задании задачи дерево подзадач будет иметь вид:



В данном примере будет истинным.

При решении любых подзадач Турбо-Пролог использует два основных правила:

- поиск решений подзадач производится слева направо.
- выбор предикатов для рассмотрения при решении каждой подзадачи осуществляется в том порядке, в котором они появляются в программе.

При этом, если какая-либо из подзадач не может быть успешно согласована, то выполняется возврат (или откат), чтобы найти другие возможные пути ее вычисления.

Рассмотрим процессы поиска с возвратом и унификации, реализуемые Пролог-системой при выполнении запроса colleague("Сидоров",Y). На рис.2, приведена схема работы Пролог-системы при выводе ответа на поставленный запрос. Решение всей задачи представляет собой последовательность четко определенных этапов (шагов) выполнения задачи:

После ввода, запрос помещается в вершину стека активных запросов, тут же система приступает к поиску предложений с тем же самым именем предиката, что и у запроса.

Анализируя найденное предложение, система унифицирует каждый аргумент запроса с соответствующим аргументом предложения (правила). После унификации запроса с заголовком, система переходит к телу предложения и унифицирует все его переменные в соответствии с заголовком правила.

Тело правила составное, поэтому первая подцель помещается в стек запросов, становится активным запросом и начинает обрабатываться как запрос. На данном этапе стек запросов имеет вид:

colleague("Сидоров",Y)

work("Сидоров",W) <- активный запрос

Новый активный запрос приступает к поиску предложений программы с тем же самым именем предиката, что и у активного запроса.

Если попытка унификации запроса с заголовком фразы заканчивается неудачей, то система переходит к анализу следующего предложения. Этот процесс продолжается до тех пор, пока не обнаружится предложение, которое будет унифицироваться с запросом. Если его не будет, то запрос завершится неудачей.

Если унифицирующее предложение найдено и оно является фактом, то подцель сразу же оказываете успешной, переменные унифицируются с фактом и в стек запросов заносится вторая подцель, которая становится активной:

colleague("Сидоров",Y)
work("Сидоров",101) * (истина)
work(Y,101) <-- активный запрос
И

Данные шаги аналогичны п.5-п.7. с той лишь разницей, что если в первом случае поиск в базе work(...) осуществлялся по фамилии, то во втором - поиск в той же базе выполняется по номеру отдела. Первое предложение базы work(...), являющееся фактом, делает активный запрос успешным. Его система помечает маркером возврата, унифицирует переменную Y с константой "Петров", и третья подцель загружается в стек запросов:

colleague("Сидоров","Петров")
work("Сидоров",101) * (истина)
work("Петров",101) * (истина)
"Сидоров"<>"Петров" <- активный запрос

Проверка активного запроса показывает, что последняя подцель является истинной, а так как значения истины приняли ранее и первые две подцели, то и вся цель является истинной при значении переменной Y="Петров", о чем выдается сообщение на дисплей. Т.е. задача решена.

Но так как цель является внешней, то после поиска первого решения и вывода его на экран система искусственно генерирует состояние неудачи и делает откат к повторному доказательству предыдущей подцели с освобождением переменных. При этом новое состояние стека запросов будет иметь вид:

colleague("Сидоров",Y)
work("Сидоров",101) * (истина)
work(Y,101) <- активный запрос

Повторяется процесс аналогичный и.8. Существенное отличие в том, что поиск осуществляется не сначала, а с той позиции, которая отмечена маркером возврата, что сокращает пространство поиска.

После согласования активного запроса с базой и унификации переменной Y константой "Сидоров", система помечает эту позицию базы маркером возврата, и стек запросов принимает вид:

colleague("Сидоров","Сидоров")
work("Сидоров",101) * (истина)
work("Сидоров",101) * (истина)
"Сидоров"<>"Сидоров" <- активный запрос

Сопоставление в активном запросе дает ложное значение, что приводит к тому, что и весь запрос является ложным. Пролог-система автоматически пытается его передоказать, вызывая ситуацию отката (возврата) к последней успешной подцели, освобождая конкретизованные в последнем запросе переменные. Стек запросов аналогичен п. 11. и

Поиск в базе, начиная с позиции, помеченной маркером возврата согласование с фактом, унификация переменной Y новым значением и переход к третьей подцели приведут к тому, что состояние стека запросов будет иметь вид:

colleague("Сидоров","Иванов")

work("Сидоров",101) * (истина)

work("Иванов",101) * (истина)

"Сидоров"<>"Иванов" <- активный запрос

Проверка в активном запросе показывает, что последняя подцель является истинной, а так как значения истины приняли ранее и первые две подцели, то и вся цель является истинной при значении переменной Y="Иванов", о чем выдастся сообщение на дисплей. Т.е. задача решена.

А так как маркер возврата стоит на конце базы данных, то у оболочки Турбо-Пролога отсутствует возможность искусственного вызова состояния неудачи, что приводит к окончанию задачи.

Задание 2

Выполните трассировку программы и составьте упрощенную схему работы Пролог-системы для случая поиска всех сослуживцев.

Задание 3

Повторите задание 2 для случая выполнения запроса о поиске всех коллег Козлова: all_colleague("Козлов",X,Y).

Откат (или возврат) автоматически инициируется системой Турбо-Пролога, если не используются специальные средства управления им.

Для управления процессом отката в Прологе предусмотрены два предиката: fail (неудача) и cut (отсечение).

Использование внешней цели побуждает переменные получать все возможные значения одно вслед за другим. При этом все их значения в не отформатированной форме выдаются в окне диалога системы. Если их число велико, то текст в окне будет быстро меняться и прочитать его будет довольно сложно или даже невозможно. Поэтому в этом случае встает задача обеспечения некоторого интерфейса вывода.

Использование отката после неудачи при использовании внутренней цели для организации простейшего интерфейса вывода

Если простейший интерфейс вывода будет реализован как Внутренняя цель программы, то внутренние унификационные процессы Турбо-Пролога закончат поиск решения сразу после первого успешного вычисления цели. Для поиска всех значений Пролог-программа должна заставить повторно работать внутренние унификационные средства Турбо-Пролога.

Одним из способов реализации данной задачи является использование метода отката после неудачи (ОПН), использующий предикат fail. Пример использования этого предиката демонстрирует программа 8.

```
/* программа 8 */
include "lab3.pro"
predicates
query
do_answer(namee) goal
query.
clauses query :-
makewindow(2,7,15," Запрос коллеги :", 18,0,6,50),
cursor 1,10),
write("Введи фамилию -> "),
readln(Who),
makcwindow( 1,7,15," Коллеги :", 1,50,22,29),
do_answer(Who).
do_answer(X) :- colleague(X.Y), write(" ",X," ; ",Y),nl,fail.
```

Два предиката этой программы позволяют формировать запрос и получать на него ответ. Текстовая подстановка файла программы б из лабораторной работы 3 обеспечивает доступ ко всем се предикатам и базам данных.

Таким образом эта программа являет собой пример интерфейса для ввода и вывода данных программы 6.

Программа содержит внутреннюю цель в виде предиката query, который создает па экране окно ввода данных, выдает подсказку, обеспечивает ввод и означивание переменной Who. а также формирует окно вывода данных.

Предикат do_answer(...) обеспечивает запрос на поиск сослуживцев введенного служащего. Однако на 11 шаге система не будет осуществлять принудительный возврат, и решение задачи остановится.

Чтобы этого не случилось, в программу введен предикат fail, который всегда вызывает состояние неудачи, что заставляет систему выполнять, откат после неудачи к предыдущей подцели.

Содержание отчета по лабораторной работе

Отчет по лабораторной работе должен содержать:

- результаты трассировки программы 7 для трех запросов, описанных в параграфе 1 данной лабораторной работы.
- результаты выполнения заданий 1,2 и 3 данной лабораторной работы.
- результаты исследования программы 8 при наличии и отсутствии предиката fail в программе.
- модифицированный вариант программы 8 для случая, когда окно вывода занимает целый экран, а окно ввода присутствует на экране только на момент ввода.
- список, назначение и описание простейших предикатов Турбо-Пролога для ввода-вывода данных разного типа.

Контрольные вопросы

1 Что такое правила? Привести примеры.
2 Для чего используется дерево вывода? Привести примеры.
3 Какова роль вопроса в программе на Прологе?
4 Что такое сопоставление?
5 Что такое конкретизация переменных?

1.5 Лабораторная работа 4

Тема: Встроенные предикаты неудачи и отсечения, арифметические выражения и сравнения

Цель работы: Ознакомление с действиями предикатов неудачи и отсечения. Изучение методов организации повторного выполнения группы задач. Знакомство со способом построения программ-меню. Управление ограничением пространства поиска с использованием отсечения.

Организация повторяющихся процессов

При выполнении запросов программа последовательно обращается к фактам и правилам. При этом правила часто требуют, чтобы задачи типа поиска элементов в базе, ввода или вывода данных выполнялись многократно. Однако в Прологе отсутствует возможность непосредственного задания итерационного процесса, т.е. не реализованы синтаксические конструкции типа FOR, WHILE или REPEAT.

Существует два способа построения правил, выполняющих одну и ту же задачу несколько раз. Это повторение, использующее возврат, и рекурсия, использующая вызов процедуры самой себя.

Повторение реализуется с использованием метода отката после неудачи, когда осуществляется возврат к последней, имеющей альтернативное решение, подцели, реализация альтернативы и очередной возврат. Поэтому поиск с возвратом можно использовать для выполнения повторяющихся процессов. Примером построения правила, использующего повторение, является предикат do_answer() в программе 8.

Но если этот подход применить к предикату query той же программы, т.е. искусственно вызвать состояние неудачи, добавив предикат fail в виде последней подцели правила, то никакого повторного запроса на ввод мы не получим.

Задание 1

Вызовите программу 8, добавьте fail в конец описания query и, запустив программу на выполнение, убедитесь что процесс не повторяется.

Объясняется это тем, что предикаты формирования окна и ввода не являются альтернативами, т.е. не имеют альтернативных решений, а предикат do_answer() все альтернативные решения получил, т.е. цель query является истинной и решение задачи заканчивается.

Для того, что обеспечить многократный ввод данных, необходимо, чтобы откат выполнялся к некоторому предикату, имеющему альтернативное

решение, с одной стороны, и чтобы это решение было бы постоянно истинным, с другой стороны. При невыполнении последнего условия откат может произойти к еще более ранним подцелям и не обеспечит повторения именно нужной нам группы действий.

Используя простейшую рекурсию, процедура задания предиката для правила повтора, определяемого пользователем, может иметь вид:

```
repeat.
repeat:- repeat.
```

Первая строка - это факт, который всегда успешен и объявляет предикат repeat истинным. Однако, так как есть для этого предиката еще одно правило, то указатель отката устанавливается на первый repeat. Вторая строка - это правило, которое использует самовызов. Правило (второй repeat) вызывает подцель (третий repeat), и этот вызов вычисляется успешно, так как факт (первый repeat) удовлетворяет подцели. Следовательно, правило также всегда успешно. Предикат repeat будет успешно вычисляться при каждой новой попытке его вызвать после отката. Таким образом, repeat - это рекурсивное правило, которое никогда не бывает неуспешным.

Задание 2

Введите в программу описание предиката repeat. Вставьте обращение к этому предикату в качестве первой подцели правила query. Запустите на выполнение программу. Теперь у Вас появилась возможность повторного ввода, но нет признака окончания и единственный вариант выхода из программы - это Ctrl+Break.

Но для того, чтобы сформировать признак окончания повторений, давайте разберемся, как работает программа, и кто инициирует повтор. Первым выполняется repeat, который ничего не делает, далее - не имеющие альтернатив стандартные предикаты и последним предикат do_answer, который и обеспечивает откат после неудачи к предикату repeat за счет присутствия в нем предиката fail.

Задание 3

Закомментируйте в правиле do_answer предикат fail и запустите программу на выполнение. Обратите внимание, что наличие в программе предиката repeat еще не обеспечивает повторов, что для организации повторов необходим возврат к предикату repeat. Снимите комментарий.

Но если откат к repeat вызывает do_answer, то он должен и обеспечить, при определенных условиях, окончание этого процесса. Если за условие выхода будет принят, например, ввод слова stop вместо фамилии, то можно доопределить предикат do_answer еще одним правилом:

```
do_answer(X) :- X="stop",
write("good bye").
```

которое будет истинным при согласовании, и которое, ввиду отсутствия признака неудачи, не вызовет отката к предикату repeat.

Задание 4

Добавьте новое правило в последнюю строку процедуры. Запустите программу на выполнение сначала в обычном режиме, а затем - в режиме трассировки, но только предиката do_answer. Переставьте новое правило на первую строку процедуры и выполните трассировку. Какой получился результат? Что будет, если правило для условия выхода записать в виде: do_answer("stop"):-write("good bye").? Какой из этого следует вывод?

Обобщая изложенное, можно сделать вывод о том, что условие выхода из цикла может определяться любым предикатом, одно из множества альтернативных описаний которого должно содержать предикат fail или вызывать поиск с возвратом, т.е. обеспечивать откат.

Из анализа даже простейших случаев организации повторяющихся процессов, как реализуемых самой Пролог-системой, так и определяемых пользователем, можно сделать вывод о необходимости управления этими процессами со стороны пользователя.

Управление поиском с возвратом

Один из вариантов управления поиском с возвратом можно иллюстрировать на примере предиката do_answer, записанного в несколько другой форме.

```
do_answer(X) :-colleague(Z.Y),X==Z,write(" ",X," -> ",Y),nl,fail.
```

Альтернативная форма записи не изменяет сути данного правила, но дает возможность показать, что введением новых подцелей в правило можно управлять поиском с возвратом. Из этого примера видно, что вторая подцель может оказаться неуспешной из-за несоответствия служащего, унифицированного по первой подцели, со служащим, унифицированным через заголовок правила, т.е. введенного с клавиатуры. Неуспех унификации второй подцели приведет к тому, что откат возникнет до выдачи информации на экран и предикат fail не потребуется. Включен предикат fail в правило, чтобы вызвать откат, если условия правила будут выполнены и все правило окажется успешным.

```
show_menu:-makcwindow(l,7,15," Меню ",1,50,10,20), repeat, clearwindow, write(" 1 - процесс 1"), nl, write(" 2 - процесс 2"), nl, write(" 0 - выход "), nl, nl, write(" Ваш выбор -> "), readint(Menu), Menu < 3, process(Menu), stop_menu(Menu).
stop_menu( 0 ) .
stop_menu( _ ) :- fail.
```

Еще одним примером по управлению поиском с возвратом является процедура построения меню show_menu.

В ней repeat используется так, что после выхода из любого модуля, вызываемого предикатом process(), идет возврат в меню.

Исключением является выбор нуля, что вызывает око1гчанис программы.

В данном правиле управление откатом используется дважды: в виде предикатов Menu<3 и slop_menu(Menu).

Подцель Menu<3 проверяет значение, введенное с клавиатуры. Если введено значение большее или равное трем, то подцель закончится неуспехом, и произойдет откат. Если введено значение меньшее 3, то подцель закончится успехом и будет сделана попытка выполнения следующей подцели process().

Если процедура process() завершится успехом, то система делает попытку выполнить процедуру stop_menu(), которая при любых, кроме 0, значениях выбора завершается неудачей, что вызывает откат к предикату repeat. При значении выбора равном 0 - она является истинным фактом, и программа завершается.

Задание 5

Сохраните на диске Ваш рабочий файл с именем lab4.prg. Он нам скоро понадобится. Создайте новый файл lab4menu.pro, в котором, используя приведенные выше предикаты, напишите программу организации выполнения двух произвольных процессов с выбором их через меню. Процессы должны быть описаны соответствующими предикатами и иметь простейший вид. Например, открытие окна и выдача сообщения.

Запустите программу па выполнение. Затем, используя трассировку, внимательно изучите ход решения задачи, откаты и повторы, выполняемые программой.

Управление ходом выполнения программ с использованием отсечения

Турбо-Пролог содержит средство, препятствующее поиску с возвратом в определенных условиях. Эта операция называется отсечением и выполняется предикатом cut, который в программах обозначается восклицательным знаком (!). Воздействие этого средства просто сводится к прекращению поиска. Отсечение используется в двух случаях:

1 Для ограничения пространства поиска в случаях, когда заранее известно, что некоторые возможные пути не приведут к интересующим вас решениям, т.е. обработка их приведет к ненужной потере времени. С использованием отсечения программа решается быстрее и требует меньшего объема памяти.

2 Когда отсечение требуется по логике программы для:

а) недопущения возврата к предыдущей подцели правила при откате;

б) предотвращения перехода к следующему предложению процедуры.

Пусть какое-либо правило имеет вид:

r1(X,Y,Z) if a(X) , b(Y) , ! , c(X,Y,Z).

Правило, записанное в такой форме, указывает на то, что система пройдет через предикат cut только в том случае, если и подцель a(X), и подцель b(Y) будут успешными. После того, как предикат cut будет обработан, система не сможет вернуться назад для повторного рассмотрения

подцелей a(X) и b(Y). Если подцель c(X,Y^) потерпит неудачу при текущих значениях переменных X,Y и Z.

- Пусть процедура описания предиката r состоит из трех правил. Обозначим через r1. r2 и r3 - записи одного и того же предиката r в каждом из трех предложений процедуры. Тогда два варианта записи этой процедуры в виде:

а) r1(X,Y) if I, a(X),b(Y), c(X.Y).
r2(X,Y) if !,d(X,Y)
r3(X,Y) if e(X,Y)

б) r1(X,Y) if a(X), b(Y), c(X,Y),!.
r2(X,Y) if d(X,Y), !.
r3(X,Y) if e(X,Y)

соответствуют тому, что, в первом случае, при обработке предиката r будет использовано лишь одно из правил r1, r2, r3, а, во втором случае, истинность какого-либо одного из правил приводит к окончанию процедуры и исключению из рассмотрения всех записанных ниже.

Пример.

Процедуру поиска максимального из двух чисел можно записать в виде двух правил для предиката max. Но эти правила взаимно исключающие. Если неудачу терпит первое, то второе будет выполняться. Поэтому с использованием отсечения возможна значительно более короткая формулировка процедуры.

```
max(X,Y,X):-X> =Y
max(X,Y,Y):-X< Y
max(X,Y,X):-X> =Y,!
max(X,Y,X)
```

Таким образом, если предикат fail инициирует бектрекинг (возврат к перебору очередных альтернатив после первой найденной), то предикат cut его завершает.

Рассмотрим на простых примерах использование различных вариантов отсечения для управления ходом выполнения Пролог-программ.

Использование метода отката и отсечения

Использование этого метода начнем с рассмотрения задачи о сослуживцах" когда возникают требования по выдаче ответов на более широкий круг вопросов, чем определение сослуживцев только одного конкретного служащего, задаваемого с клавиатуры (как это было в предыдущем примере). Предположим, что перечень возможных для обращения к системе запросов включает в себя требования по:

- поиску всех сослуживцев,
- поиску всех сослуживцев конкретного служащего,
- выяснению наличия сослуживцев у конкретного служащего,
- выдаче списка всех сослуживцев, ограниченного снизу некоторым служащим.

Один из вариантов реализации требований по запросам к системе может иметь вид расширенной процедуры do_answer()

```
/* правило 1 */
```

```
do_answer("stop") :- write("good bye"). /* правило 2 */
do_answer(X) :- X="all", colleague(Z,Y), write(" ",Z," -> ",Y),nl, fail,!.
/* правило 3 */
do_answer(X) :- frontchar(X,"!",Z), colleague(Z,_),!, write(Z." имеет сослуживцев"), nl, fail.
/* правило 4 */
do_answer(X) :- frontchar(X,"< ",Z), colleague(Q,Y), write(" ",Q," -> ",Y),nl, Q=Z,!. fail.
/* правило 5 */
do_answer(X) :- colleague(X,Y), write(" ",X," -> "Y),nl, fail.
```

Процедура do_answer() состоит из пяти правил. Два из них, первое и пятое, уже были рассмотрены ранее.

Задание 6

Загрузите с диска Ваш файл lab4.pro и дополните процедуру do_answer() недостающими правилами. Изучите их назначение, способ организации и исполнения системой Турбо-Пролог.

Правило 1 обеспечивает прекращение повтора запросов за счет того, что его истинность при согласовании не вызывает отката к предикату repeat.

Правило 5 реализует второе требование ни запросам к системе и выдаче всех сослуживцев конкретного служащего. Рассмотрим подробнее остальные правила.

Правило 2 обеспечивает поиск и выдачу всех сослуживцев при условии, что в ответ на запрос о фамилии вводится слово ail (т.е. вес). Если с клавиатуры введено любое другое значение, первая подцель этого правила не согласуется и выполняется откат, который приводит к дальнейшему поиску согласующих предложений среди заголовков правил процедуры do_answer().

Если первая подцель согласуется, то выполнение второй подцели обеспечит поиск первой пары коллег, а третья подцель выводит их на экран. Предикат fail, вызывая неудачу, обеспечивает откат ко второй подцели для поиска всех альтернативных решений, которые затем выдаются на экран.

Разработчику программы заранее известно, что для обработки слова all никаких других правил в процедуре do_answer не предусмотрено, но это неизвестно системе, которая будет просматривать все правила процедуры и проводить их унификацию.

В этом случае имеет смысл ограничить время и пространство поиска, закончив выполнение всей процедуры do_answer после обработки слова all. С этой целью в данное правило последним введен предикат отсечения, который приводит к прекращению процедуры do_answer, т.е. исключению из дальнейшего согласования правил 3,4 и 5.

Правило 3 реализует третье требование к запросам, при условии, что в ответ па приглашение к вводу, фамилия вводится, начиная с восклицательного знака. Выбор начального символа абсолютно произволен. Здесь этот символ выбран, как еще одно напоминание об отсечении.

Пусть, например, с клавиатуры введено! Иванов. При согласовании заголовка правила 3 переменная X унифицируется со строковой константой "!Иванов" и начинается обработка тела правила.

Первая подцель вызывает предикат frontchar(X,"!",Z) обработки символьных строк (см. приложение 1). Если первый символ X совпадает с '!', то переменная Z принимает значение остатка строки X, начиная со второго символа (Z = "Иванов"). Если первый символ в X отличный от '!', то первая подцель заканчивается неуспехам и происходит откат к обработке, следующего правила процедуры.

Вторая подцель будет успешна, если будет согласован предикат colleague(Z,_)

при каком-либо значении Z. Использование анонимной переменной обусловлено характером запроса, в котором требуется определить только наличие коллег, а не их фамилии. Успешное согласование второй подцели вызовет переход к предикату отсечения, который установит признак запрета на откат в этом месте правила.

После этого согласование правила продолжается в порядке слева направо, и осуществляется вывод на экран дисплея сообщения "Иванов имеет сослуживцев".

Последняя подцель вызывает состояние неуспеха в доказательстве правила, что ведет к откату. Но так как предикат write() альтернатив не имеет, то происходит откат к предикату отсечения, которым уже установлен запрет на дальнейший откат.

Это приводит к тому, что в отличие от предыдущих примеров, повторное согласование предиката colleague() выполняться не будет.

На этом завершается согласование данного правила, но не только его, но и всей процедуры do_answer(). Причем завершение процедуры неудачей ведет к тому, что неудачей завершится и последняя подцель в query и, как следствие, произойдет откат к предикату repeat и повтору ввода данных.

Задание 7.

Что будет, если из этого правила исключить предикат fail? Попробуйте это сделать. Изучите процесс выполнения программы в этом случае. Восстановите программу. В отчете приведите описание процесса выполнения программы.

Правило 4 реализует требование на ограничение выводимого списка некоторым условием. В качестве условия выступает фамилия служащего, которым должен оканчиваться список коллег. При этом фамилия этого служащего вводится, начиная с символа '<'. Выбор этого символа произволен и никакой смысловой нагрузки, с точки зрения языка Пролог, не имеет.

Первая подцель даст истинный результат, если первым символом X будет '<'. Переменная Z в этом случае примет значение остатка строки X. начиная со второго символа. В иных случаях, подцель закончится неуспехом, вызывая неуспех и всего правила. Фактически эта подцели является условием входа на обработку правила.

Следующие две подцели согласуют и выводят на экран первую пару коллег, а четвертая - обеспечивает возврат к поиску следующей пары, если фамилия первого из коллег не совпадает с введенной с клавиатуры. Как только эти фамилии совпадут, т.е. Q=Z станет истиной, выполнится отсечение, которое не позволит предикату fail обеспечить откат к поиску новой пары коллег, что вызовет окончание выполнения данного правила.

Наличие в правиле предиката отсечения не только ограничивает область поиска в базе данных заданной фамилией, но и обеспечивает завершение всей процедуры do_answer0. Завершение процедуры приводит к возврату в query.

Следует особо обратить внимание на то, что откат к поиску альтернативных решений подцели colleague(Q, Y) в данном правиле, в отличие от правил 2 или 5, выполняет не fail, а подцель Q=Z. Тогда встает вопрос, а нужен ли в этом правиле вообще предикат fail, а если нужен, то зачем?

Все дело заключается в том, что процедура do_answer() выполняется не сама по себе, а вызывается как последняя цель правила query. Поэтому, кроме функций по поиску данных в базе, на нее возложена функция управления откатом к предикату repeat, т.е. организация повторений по вводу.

Присутствие в правиле 4 предиката fail приводит к тому, что, несмотря на успешно выполненный запрос к базе данных, правило заканчивается неудачей. Неудачей завершается и запрос из query к процедуре do_answer(), что вызывает откат к repeat и повторение ввода данных.

Если в правиле 4 не будет предиката fail, то после выполнения поиска в базе правило будет успешно выполненным. Успешным будет и обращение из query к процедуре do_answer(), что ведет к тому, что и правило query станет истинным. Это вызовет его окончание, а вместе с этим и окончание ввода запросов.

Но так как признаком окончания ввода должен быть ввод слова stop, то окончание работы программы по другому требованию явилось бы ошибкой, что и требует обязательного включения предиката fail в правило 4.

Задание 8

Выполните программу в пошаговом режиме для правила 4. Изучите процесс выполнения отсечения. Повторите то же самое без fail. В отчете приведите схему выполнения программы для этих случаев.

Рассмотренные выше подходы к организации откатов и отсечении могут аналогичным способом использоваться и при формировании внешних целей в виде составных запросов.

Задание 9

Сформируйте внешние цели для запросов, аналогичных реализованным в процедуре do_answer() и выполните их.

5. Откат и отсечение при реализации отношений вида "один-ко-многим"

Учет ассоциаций (связей) между объектами отношений служит средством оптимизации запросов в Прологе. Рассмотрим это на примере простого отношения

ОТЕЦ(ИМЯ , РЕБЕНОК)

которое в программе определяется набором некоторых фактов. Между объектами данного отношения существует ассоциация "один-ко-многим". Запрос к данной базе может заключаться:

- либо в поиске родителя конкретного ребенка (в этом случае имеет место простая связь)
- либо в поиске всех детей конкретного родителя (в этом случае имеет место множественная связь)

Самый примитивный способ реализации простой связи заключается в написании правила, которое должно выполнить поиск факта, а затем пройти через предикат отсечения. Для реализации сложной связи необходим поиск всех альтернативных решений в базе данных. Использование новой процедуры parent (родитель), которая учитывает введенные замечания, обеспечит интерфейс работы с БД father() .

Причем этот интерфейс учитывает имеющуюся связь между объектами отношения БД father() и оптимизирует выполнение запроса Прологом

Предикат bound(C) успешен в случае, если переменная C означена, а предикат free(C) успешен, если переменная C свободная.

Следует отметить, что два правила процедуры являются взаимоисключающими.

А если это так, то проверку во втором правиле состояния переменной C можно не выполнять, так как если она будет несвободной, то будет обработана первым правилом. Однако, если эти правила поменять местами, то этого делать нельзя.

```
/* программа 10 */
domains
namee,child = symbol
predicates
father(name, child)
parent(namee,child)
clauses
father( "иван" , "петр" ).
father( "иван" , "павел" ).
father("петр" , "олег").
father("олег" , "борис").
/* поиск родителя (отца) конкретного ребенка */
parent(F,C) :- bound(C), father(F,C),!.
/* поиск всех детей конкретного родителя */
parent(F,C) :- free(C), father(F,C).
```

Задание 10

Модифицируйте программу 6 таким образом, чтобы интерфейс по работе с БД work() учитывал ассоциации между объектами этого отношения и позволял оптимизировать выполнение запросов Прологом за счет использования нового отношения СЛУЖАЩИЙ(ИМЯ,ОТДЕЛ)

С целью тестирования программы выполните различные варианты запросов, используя внешние цели. Результаты их выполнения вместе с текстом модифицированной программы приведите в отчете по лабораторной работе.

Используя режим пошагового выполнения, выясните, как сказывается отсечение на режиме выполнения программы и поиске по базам данных.

Задание 11

Используя результаты выполнения задания 10, па основе программы формирования меню lab4menu.pro сформируйте запросную систему, которая должна выдавать ответы на вопросы: "В каком отделе работает конкретный служащий" (процесс 1) и "Кто работает в конкретном отделе" (процесс 2)

.

Ступенчатые функции и отсечение

Часто в экономических расчетах приходится использовать ступенчатые функции для вычисления различных коэффициентов, зависящих от диапазонов изменения объемов производства, прибыли или совокупного годового дохода. Для примера рассмотрим двухступенчатую функцию, аналогичную вычислению процента подоходного налога, считая, что D - совокупный доход, а N - величина налога:

На Прологе данную функцию выражают с помощью бинарного отношения F(N,D), которое можно определить набором фактов вида

```
F(D ,0) :- D< 10.
F(D, 12) :- 10=<D,D< 15.
F(D, 15) :- 15=<D.
```

Три правила, входящие, в отношение F(), являются взаимоисключающими, поэтому успех возможен самое большое в одном из них. Следовательно, мы (но не Пролог) знаем, что, как только успех наступил в одном из них, нет смысла проверять иные, поскольку они все равно обречены на неудачу. Для предотвращения ненужного перебора следует воспользоваться отсечением. Предикат отсечения предотвращает возврат из тех точек программ, где он поставлен. С учетом этого новая процедура

```
F(D ,0):- D< 10,!.
F(D, 12):- D< 15,!.
F(D, 15).
```

дает тот же результат, что и исходная, но она значительно более эффективна при реализации в программе на Прологе.

Содержание отчета по лабораторной работе

Отчет по лабораторной работе должен содержать:
Результаты выполнения задания 4 данной лабораторной работы.
В соответствии с заданием 5 текст отлаженной программы формирования меню, которая в ответ на выбор опции меню открывает некоторое окно, а после нажатия клавиши возвращается в основное меню, закрывая окно.
Полный текст программы запросной системы по сослуживцам и результаты ее исследований в соответствии с заданиями 6,7,8.
Список внешних целей, аналогичных реализованным в запросной системе, и результаты их выполнения (задание 9)
Результаты выполнения задания 10.
Текст отлаженной программы запросной системы по служащим и отделам, с использованием меню и ее описание.
В качестве дополнительного задания - программа полной запросной системы.

Контрольные вопросы

1. Каковы отличия программы на прологе от программ на процедурных языках?
2. Какова сфера применения языка Пролог?
3. Каким образом можно управлять процессом логического вывода на языке Пролог?
4. В каких случаях успешен предикат bound?
5. В каких случаях успешен предикат free?

2 Рекурсия и организация рекурсивных вычислений

2.1 Лабораторная работа 5

Тема: Рекурсия и рекурсивные процедуры в Прологе
Цель работы:
Знакомство с рекурсией, как с алгоритмическим методом.
Изучение способов построения рекурсивных процедур.
Знакомство с особенностями рекурсии в Пролог-программах.
Изучение способов обеспечения целостности отношений.

Определение понятия рекурсии

Рекурсия - алгоритмический метод, часто используемый в Прологе. Рекурсию применяют для тех же целей, что и циклические конструкции в процедурных языках. В рекурсивных правилах более сложные входные аргументы выражаются через менее сложные. Например, рекурсивный набор инструкций по погрузке контейнеров может иметь такой вид:
Для того, чтобы погрузить N контейнеров, нужно:
Если N=0, то остановиться.

Если N>0, то погрузить один контейнер, затем погрузить еще N-1 контейнер.

Будем считать, что здесь дано определение процедуры "погрузка N контейнеров", где N - аргумент процедуры и обозначает некоторое целое число.

Данная процедура рекурсивна, т.к. последняя строка - '"погрузить N-1 контейнер" - является вызовом процедурой самой себя. Заметим, что аргумент при рекурсивном вызове проще, чем исходный аргумент N, в том смысле, что N-1 - это число меньшее, чем число N. Поэтому "погрузка N контейнеров" представляет собой более сложный случай, выраженный через менее сложный случай выполнения тех же самых действий, т.е. через "погрузит N-1 контейнер".

Классическим примером рекурсивного определения в Прологе может служить процедура "предок", которая состоит из двух правил:

предок(А,Б):-родитель(А,Б).

предок(А,Б):-родитель(В,Б), предок(А.В).

Совокупность этих правил определяет два способа, в соответствии с которыми одно лицо (А) может быть предком другого липа (Б).

Согласно первому правилу, А является предком Б, если А - родитель Б, т.е. А является ближайшим предком Б.

Согласно второму. А будет предком Б, если есть некоторый В, который, являясь родителем Б, имеет своим предком А. Другими словами, А - предок Б, если А -предок родителя Б, т.е. А - отдаленным предок Б Таким образом второе правило зависит от более простой версии самого себя, т.е. от подцели "предок".

Задание 1

В программу 10 введите описание процедуры "предок". Выполните ряд произвольных запросов к программе. Используя режим трассировка, изучите последовательность обработки в Турбо-Прологе процедуры "предок". Сохраните программу в файле "lab5.pro"

Состав рекурсивной процедуры

Любая рекурсивная процедура должна включать по крайней мере по одной из перечисленных ниже компонент:

1. Нерекурсивное предложение (правило или факт), определяющее исходный вид процедуры, т.е. вид процедуры в момент, прекращения рекурсии. Это, так называемые, граничные условия.
2. Рекурсивное правило. Начальные подцели, располагающиеся в теле этого правила, вырабатывают новые значения аргументов. Далее размещается рекурсивная подцель, в которой используются новые значения аргументов.

Первое предложение процедуры "предок" определяет исходный вид процедуры. Как только данное предложение станет истинным, дальнейшая рекурсия прекратится. Говоря иначе, первое предложение является граничным условием.

Второе предложение - это рекурсивное правило. При каждом вызове данное правило поднимается на одно поколение вверх. Подцель родитель(В,Б), входящая в тело правила, вырабатывает значение переменной В. Затем располагается рекурсивная подцель предок(А,В), где используется новый аргумент.

Рассмотрим еще один пример построения рекурсивной процедуры для вычисления факториала любого целого числа.

Из определения факториала известно, что О! = 1, а факториал любого числа N может быть вычислен как факториал N-1, умноженный на N. Это определение является рекурсивным, поскольку сводит задачу нахождения N! к более простой задаче нахождения (N-1)! и затем умножения полученного значения на N.

```
/* программа 11 */
predicates
f(integer,integer)
clauses
f(U) :- !.
f(N,R) :- M=N-1, f(M,V), R=V*N.
```

Для обозначения факта, что факториал числа N равен R, используем предикат f(N,R). Рекурсивное его определение будет иметь вид, приведенный в программе 11.

Здесь первое правило определяет граничное условие для рекурсивной процедуры. Второе правило является рекурсивным, так как вторая подцель этого правила содержит вызов самой процедуры, правда с изменёнными первой подцелью значениями аргументов.

При выполнении заданной цели f(3,X) Турбо-Пролог трижды обращается к процедуре. При этом первые два раза согласуется второе правило, а в третий раз - первое. Особенность согласования второго правила заключается в том, что оба раза выполнение третьей подцели откладывается (заносится в стек запросов) в виду рекурсии второй подцели. И только после согласования на третьем шаге первого правила Турбо-Пролог возвращается к выполнению третьих подцелей. Они последовательно, начиная с последней, извлекаются из стека запросов и выполняются.

Задание 2

Используя режим трассировки, изучите последовательность обработки рекурсивных процедур на примере вычисления факториала. Измените программу таким образом, чтобы она вычисляла сумму заданной последовательности целых чисел. Отладьте программу, а потом вычислите сумму первых тысячи чисел, сумму первых 10 тысяч чисел. Какой получился результат? Как его можно объяснить?

3. Особенности выполнения рекурсивных процедур Пролог-системой

При использовании рекурсии и неограниченном или очень большом количестве рекурсивных вызовов, количество отложенных на выполнение подцелей в стеке запросов постоянно растет и в некоторый момент стек переполнится. На экране появится сообщение об ошибке. Частично помочь в

этой ситуации может увеличение размера стека, который может быть изменен с помощью опции меню системы Установки/Разные установки/Размер стека. Однако, если установлено предельное значение, то это уже не поможет. Недостатки, в этом случае, вызваны плохо продуманной организацией процедур. Для примера, вернемся к процедуре "предок" и определим ее несколько иначе:

С декларативных позиций смысл процедуры "предок1" идентичен процедуре "предок", но процедурные трактовки существенно разнятся. В процедуре "предок1" переменная В не конкретизирована в момент обработки подцели предок1(А,В). На практике это означает то, что система, выполняя запрос к процедуре '"предок!", вначале отыщет правильные ответы, затем будет выполнять рекурсивные действия вплоть до исчерпания доступного объема памяти.

```
предок1(А,Б):-родитель(А,Б).
предок1(А,Б):-предок1(А.В),родитель(В,Б).
```

Задание 3.

Загрузите программу "lab5.pro", замените в ней процедуру "предок" на "предок1". Выполните ряд запросов и исследуйте результат.

Процедура "предок1" называется процедурой с левой рекурсией, так как во втором правиле рекурсивная цель стоит слева от других подцелей. Турбо-Пролог не может надежно обрабатывать леворекурсивные процедуры, что обусловлено природой заложенной в него стратегии решения задач. Поэтому, строя рекурсивные процедуры, необходимо это учитывать. Это особенно важно, гак как рекурсия - это основной алгоритмический подход построения Пролог-программ.

Пример рекурсивной процедуры поиска длины маршрута на графе

Рассмотренная выше рекурсивная процедура "предок" строилась на основе бинарного отношение РОДИТЕЛЬ(ОТЕЦ, РЕБЕНОК), которое устанавливало логическую связь между этими двумя объектами. Вместе с тем в практике часто встречаются случаи, когда между объектами существует как качественная, так и количественная взаимосвязь.

Примером такого отношения является, например,

ДОРОГА (ГОРОД1, ГОРОД2, РАССТОЯНИЕ).

которое характеризует не только наличие связи между какими-либо городами, но и расстояние между ними.

Используя это отношение, можно сформировать базу данных по транспортным магистралям, на которой можно выполнять поиск маршрута между двумя заданными городами и определение расстояния между ними. Таким образом, встает задача разработки процедуры формирования нового отношения

ПУТЬ(ГОРОД1, ГОРОД2, ДЛИНА_ПУТИ).

на основе исходного отношения "дорога". Метод рекурсии, использованный при составлении процедуры '"предок" можно применить и в этом случае.

```
/* программа 12 */
domains
town = string
distance = integer
predicates
road(town,town,distance) route(town,town,distance)
clauses
road("С-П6" ,"Чудово", 110). road("Волхов","С-П6" , 124).
road("Чудово" ,"Волхов", 102). road("Чудово","Тихвин", 165).
road("Волхов","Тихвин", 122).
route(Townl ,Town2,Distance):-road(Town 1 ,Town2,Distance).
route(Town1 ,Town2,Distance):-road(Town1,X,Dist1), route(X,Town2,Dist2),
Distance=Dist l+Dist2,!.
```

В процедуре route() отношение между двумя городами будет соблюдаться, если существует дорога, по которой можно добраться из одного города в другой через ряд населенных пунктов.

Каждое предложение для предиката road() описывает дорогу из одного города в другой, имеющую определенное расстояние.

Правила процедуры route() указывают на возможность, как прямого маршрута, так и маршрута через другие города. Предикат route() определен рекурсивно.

Если маршрут содержит только один участок дороги, то в этом случае расстояние между городами равно длине этого участка.

В случае, если маршрут из Города1 в Город2 является суммой маршрутов из Города1 в X и из X в Город2, то расстояние между Город1 и Город2 будет равно расстоянию между Город1 и X плюс расстояние между Город2 и X, т.е. сумме двух расстояний.

Второе правило является рекурсивным. Отношение, записанное в заголовке правила, зависит от более простой версии самого себя. Первое правило определяет гранитное условие выхода из рекурсии. Как только оно станет истинным, то процесс рекурсии прекратиться.

Задание 4

Загрузите программу 12 и исследуйте путь от С-Пб до Волхова и до Тихвина. После чего попробуйте определить маршрут от Волхова до С-П6. Какой Вы получили результат?

Данная программа не способна учесть все возможные комбинации начальных и конечных точек маршрутов, т.к. при формировании процедуры не учтены ограничения, обеспечивающие целостность отношения.

Ограничения и свойства, обеспечивающие целостность отношения

Для простоты рассмотрим только отношения между двумя объектами (бинарные отношения). Для любого бинарного отношения справедливо одно из ограничений: один-к-одному, один-ко-многим, многие-ко-многим. Кроме этого любое отношения можно характеризовать наличием или отсутствием таких свойств, как:

- симметрия (асимметрия)
- рефлексивность (нерефлексивность)
- транзитивность (транзитивность)

По умолчанию Турбо-система выполняет действия с предикатами, считая, что они регулируются ограничением вида многие-ко-миогим. При учете ограничений вида один-ко-многим можно воспользоваться дополнительными отношениями и предикатом отсечения, как уже излагалось в предыдущей работе.

Бинарное отношение, реализуемое в языке Пролог, по умолчанию обладает свойствами рефлексивности, асимметричности и нетранзитивности.

Отношение между двумя объектами будет симметричным, если роли, которые играют эти объекты, взаимозаменяемы.

Отношение, которое соблюдается, когда оба объекта одинаковы, называется рефлексивным.

Отношение между объектами называется транзитивным, если оно сохраняется при переходе от прямого отношения к косвенному. Транзитивные отношения обычно реализуются рекурсивными процедурами типа "предок" или "маршрут".

На логических схемах симметричные отношения изображаются двунаправленной стрелкой, асимметричные - однонаправленной. Асимметричное транзитивное отношение с косвенными связями через несколько уровней будет выглядеть как дерево, "растущее" вниз.

Самый простой способ изменить свойства отношений это ввести в программу дополнительные факты, обеспечивающие соблюдение этих свойств. Так, если в программе содержатся факты

```
супруги("Иван", "Марья")
супруги(tom ,betty),
```

то отношение "супруги" можно сделать симметричным путем добавления дополнительных фактов с обратным следованием объектов:

```
супруги("Марья", 'Иван")
супруги(belly , torn)
```

Альтернативой применения данного подхода является использование процедуры, устанавливающей симметричность, нерефлексивность или транзитивность. Так, процедура

```
супруги_симетр(Супруг1,Супруг2):- супруги(Супруг1,Супруг2),!.
супруги_симетр(Супруг1,Супруг2):- супруги(Супруг2,Супруг1).
```

устанавливает симметрию отношения "супруги", а рекурсивная процедура "предок" устанавливает транзитивность отношения предок. Отношение "сослуживцы" можно было превратить в нерефлексивное путем добавления последней подцели

colleague(Manl,Man2) :- work(Manl,X), work(Man2,X), Manl<>Man2.

Таким образом, чтобы улучшить программу 12, можно базу данных "дороги" сделать симметричной путем введения дополнительной процедуры, а затем обратиться к этой базе данных из процедуры route(). Правило для route() при этом автоматически унаследует симметрию базы данных "дороги". Кроме этого, если еще добавить подцель, которая позволит сделать отношение нерефлексивным, то это позволит избавиться от некоторых неточных ответов системы.

Задание 5

Доработайте программу 12 так, чтобы используемые в ней отношения были бы симметричными и нерефлексивными и исследуйте се. Попробуйте теперь определить маршрут от Волхова до С-Пб. Какой Вы получили результат? Испытайте еще ряд целей. Попробуйте исключить предикат отсечения из второго правила.

Следует отметить, что удовлетворив отношение всем ограничениям целостности, мы вес же не полностью исключили варианты ситуации, когда Турбо-Пролог выберет маршрут, включающий один и тот же промежуточный пункт, несколько роз, т.е. езду по кругу. Добиться исключения зацикливания можно путем составления списка городов, включенных системой в маршрут и запрещения поиска маршрутов для городов, имеющихся в списке. Но для организации этого процесса надо уметь работать со списковыми структурами, которые рассматриваются в следующих работах.

Содержание отчета по лабораторной работе

Отчет по лабораторной работе должен содержать:

Текст программы по заданию 1, запросы к ней и результаты их обработки.
Программу вычисления суммы заданной последовательности целых чисел.
Результаты выполнения заданий 3 и 4. Текст программы по заданию 5.

Контрольные вопросы

1. Что такое рекурсия? Привести примеры
2. Дать рекурсивную формулировку понятия предок?
3. Для чего используются понятия шага и базиса рекурсии?
4. Каким образом можно изменить свойства отношений?

2.2 Лабораторная работа 6

Тема: Списки и процедуры обработки списков

Цель работы:

Знакомство с использованием списков в Пролог-программах.

Изучение рекурсивных процедур обработки списков.
Получение навыков работы по обработке списков.
Знакомство с преобразованием набора фактов в списки.

Списки как рекурсивные структуры данных

Список - широко используемая структура данных, которую удобно применять при рекурсивной обработке информации, состав и количество которой изменяется в ходе процесса обработки.

Список - упорядоченная последовательность элементов, которая может иметь произвольную длину. Элементами списка могут быть любые термы:

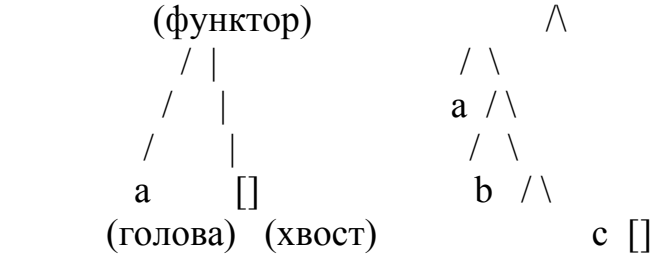
- константы
- переменные
- структуры, которые могут включать и другие списки.

Списки широко используются при создании баз данных и знаний, экспертных систем, карт городов, программ на ЭВМ и математических объектов (графы, формулы, функции).

Список - это либо пустой список (записывается в квадратных скобках []), не содержащий ни одного элемента, либо структура, имеющая в своем составе две компоненты: "голову" и "хвост" списка.

"Голова" и "хвост" списка являются компонентами функтора, обозначаемого точкой. Например, список, состоящий из одного символьного элемента "a" может быть записан в виде. (a, []) и его представление в виде дерева имеет следующую структуру:

Рассмотрим еще один пример. Пусть имеется список, состоящий из трех символьных данных 'a', 'b', и 'c', который можно записать в виде: (a, . (b, . (c, []))) или представить в виде бинарного дерева, структура которого приведена справа:



Запись сложных списков с помощью функтора не удобна, поэтому в Прологе используется иная форма записи, при которой элементы списка заключаются в квадратные скобки и разделяются запятыми. Так для рассмотренных примеров запись списков с использованием скобочной формы записи будет иметь вид: [a] и [a, b, c], соответственно.

Непустой список можно рассматривать как состоящий из двух частей:

- первого элемента списка - головы списка,
- оставшейся части списка - хвоста списка.

Голова является элементом списка и это неделимое значение. Хвост - это есть список, состоящий из всех элементов исходного списка, за исключением первого элемента. Хвост может быть поделен и дальше на голову и хвост.

Выполняя аналогичные операции, этот процесс можно продолжать, пока список не станет пустым. Из этого следует очевидный вывод, что список является рекурсивной структурой данных.

В Прологе введена специальная форма для представления списка с "головой" X и "хвостом" Y, где для разделения X и Y используется символ вертикальной черты, т.е. [X |Y]. Чтобы поставить вертикальную черту, нажмите Alt+124.

Используя данный подход, список [a, b, c] можно записать как [a | [b, c]]. Здесь "a" - голова списка, первый его элемент, а [b, c] - '"хвост" списка, часть списка, начинающаяся со второго элемента.

Использование списков с Пролог-программах

Применение списков в программе отражается на трех ее разделах. Домен списка должен быть описан в секции domains, а работающий со списком предикат - в секции predicates. Наконец, нужно задать сам список в секции clauses или goal. Рассмотрим пример простой программы, использующей в своем составе список.

```
/* программа 13_1 */
domains
list_season=season*
season=string
predicates
year(list_season)
clauses
year(["зима", "весна", "лето", "осень"])
```

Эта программа содержит список времен года. Наименование сезонов является данным символьного типа и представляет собой домен элементов списка. Отличительной чертой в описании списка является наличие символа звездочки (*) после имени домена элементов

Описание предиката ничем не отличается от обычного, когда в скобках после его имени указывается набор аргументов. Запросы к предикатам, содержащим списковые структуры, могут быть как внешними, так и внутренними. Для изучения правил формирования запросов к списковым структурам, а также с целью изучения правил унификации переменных элементами списка выполним ряд внешних запросов к программе.

Задание 1

Загрузите программу 13_1 и введите ряд запросов, анализируя получаемый результат:

"Каков весь список сезонов в году?" - year(All)

"Какой второй сезон в году?" - year([_,X,_,_]) или year([_,X|_])

"Какие сезоны составляют вторую половину года?" - year([_,|X]) или year([_ | [_|X]])
"Какой Вы надеетесь получить результат от запроса - year([X|Y])?

Придумайте еще несколько запросов, введите их в программу. Результаты диалога с ЭВМ привести в отчете по работе.

Рассмотрим еще три примера программ. Программа 13_2 показывает возможность использования в предикатах списковых структур совместно с другими доменными структурами (предикат office содержит целочисленный и списковый домены).

```
/* программа 13_2 */
domains
number=integer
worker=string
list_worker=worker*
predicates
office(number,list_worker)
clauses
office(101,["Петров", "Сидоров", "Иванов"]).
office(211,[ "Павлов" ]).

/* программа 13_3 */
domains
number,salary=integer
name=string
member=worker(name,salary)
list_worker=member*
predicates
office(number,list_worker)
clauses
office(101,[ worker("Петров", 500), worker( "Сидоров",300), worker("Иванов", 200) ]).
office(211,[ worker("Павлов", 400) ]).

/* программа 13_4 */
domains
name = string
list_worker = name*
number = integer
office == office( number , list_worker )
list_office = office*
predicates
all_office( list_offlce )
clauses
all_office( [office(101,["Петров","Сидоров","Иванов"] ), office(211, ["Павлов"]) ]).
```

Программа 13_3 иллюстрирует тот факт, что элементами списка могут быть не только стандартные домены, но и любые, определяемые пользователем составные структуры. Так, например, в данной программе элементами списка list_worker являются экземпляры структуры worker(name,salary).

Наконец программа 13_4 демонстрирует, что элементами списковых структур могут являться и сами списки. Так, элементом списка list_office является структура office(number , list_worker), одним из аргументов которой является список.

Следует особо отметить, что вес три программы служат для описания и хранения баз данных одной и той же предметной области. При этом логические структуры храпения информации существенно различаются. Использование списковых и составных структур существенно расширяет возможности проектирования и использования логических моделей данных.

Задание 2

Тщательно разберитесь со структурной организацией каждой из программ, загрузите их в ЭВМ и введите ряд запросов.

По программе 13_2: "Кто работает в 101 отделе?" - office(101,All) "Кто начальник (первый по списку) 101 отдела?" - office(101,[N|_]) "Какие есть отделы в фирме?" - office(X,_) "Каков состав всей фирмы?" - office(_.L) По программе 13_3: "Кто зам. (2-ой по списку) 101 отдела?" - office(101, [_,worker(X,_)|_]) "Какие оклады у начальников отделов?" - office(_, [worker(X,Y)|_]) По программе 13_4: "Кто входит в совет директоров (первое подразделение в составе фирмы)?" - all_office([office(_,X)|_]) Придумайте еще несколько запросов, введите их в программу. Результаты диалога с ЭВМ привести в отчете по работе. Попробуйте графически изобразить схемы логических моделей предметной области для каждого из вариантов программы.

Простейшие процедуры работы со списками

В задании 1 внешняя цель year(All) обеспечивала присвоение переменной All всего списка в целом. Напротив, цель year(_,X,_,_) позволяла извлечь из списка всего лишь один элемент. Однако, в этом случае требовалось точное знание числа элементов в списке. Если задать цель в виде year([X,Y]), то она не будет удовлетворена в виду несоответствия количества элементов в списке и целевом утверждении.

Вместе с тем, возможность отделения от списка первого элемента позволяет обрабатывать его отдельно вне зависимости от длины списка. Причем, оставшийся хвост можно снова рассматривать как список, в котором можно выделить первый элемент. Аналогичные отделения первого элемента можно проводить до тех пор, пока весь список не будет исчерпан. Этот рекурсивный подход составляет основу построения процедур обработки списковых структур.

Рассмотрим одну из простейших процедур обработки списков, обеспечивающую последовательный доступ к элементам списка и вывод их на экран дисплея. Пример ее описания и применения приведен в программе 14_1.

Процедура print_Iist() включает два правила. Первое - это факт, определяющий граничное условие рекурсивной процедуры, т.е. конец вывода элементов списка, если он пуст.

Второе правило обеспечивает разделение списка на голову и хвост, вывод первого элемента

```
/* программа 14_1 */
domains
list_season=season*
season=string
predicates
year(list_season)
print_list(list_season)
goal
year(L), print_list(L).
clauses
yeaг(["зима", "весна", "лето", "осень"]).
print_list([]).
print_list([X|Y]):- write(X), nl, print_list(Y).
```

списка на экран дисплея, перевод курсора на новую строку и рекурсивный вызов этого же правила, но уже применительно к хвостовой части списка. В общем случае это правило записывается следующим образом

```
print_list([]):- L=[X|Y], write(X), nl, print_list(Y).
```

но принимая во внимание тот факт, что Пролог сопоставляет с целью заголовок правила прежде, чем пытается согласовать его тело, возможна более короткая запись этого правила, приведенная в программе 14_1.

В программе 14_1 используется внутренняя цель, состоящая из двух подцелей. Первая из них обеспечивает формирование списка наименований времен года на основе имеющейся в БД информации. Вторая обеспечивает вывод сформированного списка на экран дисплея. __

```
/* программа 14_2 */
domains
number,salary=integer
name=string
member=worker(name,salary)
list_worker=member*
predicates
office(number,list_worker)
print_list(list_worker)
show_workcr
clauses
```

```
show_worker:- makewindow(l,7,15,"Служащие",5,10,12,30), cursor(2,l), write("
Введи номер отдела -> "), readint(Otd), office(Otd,L), print_list(L), readchar(_).
office(101,[worker("Петров", 500), worker("Сидоров" ,300), worker("Иванов",
200) ]).
office(211,worker("Павлов", 400) ]).
print_list([]).
print_list([X|Y]):- write(X),nl, print_list(Y).
```

Задание 3

Доработайте программу 13_1 до 14_1 и запустите на выполнение. Используя trace print_list(), познакомьтесь с процессом деления списка в процессе выполнения рекурсии.

Однако, внутренняя цель программы ограничивает возможные действия по модификации данных в базе: добавлению, поиску, исключению и т.д.

В этих условиях, для вывода элементов списка на экран предпочтительнее ввести новый предикат и определить для него правила, в соответствии с которыми осуществляется поиск и вывод требуемых данных из базы.

Пример такого подхода реализован в программе 14_1, с использованием предиката show_worker.

Задание 4

Доработайте программу 13_3 до 14_2, разберитесь в ней и запустеть на выполнение.

Процедуры обработки списков

Списки можно применять для представления множеств, хотя и есть некоторое различие между этими понятиями. Порядок элементов множества не существенен, в то время как для списка этот порядок имеет значение. Кроме того, один и тот же объект R списке может встретиться несколько раз. Однако, наиболее используемые операции над списками аналогичны операциям над множествами. Это проверка объекта на принадлежность множеству, объединение двух множеств, удаление из множества некоторого объекта.

Принадлежность к списку. Представим отношение принадлежности о виде member(R.L), где R - терм, а L - список. Цель member(R,L) истинна, если терм R встречается в L. Задача проверки вхождения терма в список формулируется в виде:

Граничное условие: Терм R содержится в списке, если R есть голова списка L.

Рекурсивное условие: Терм R содержится в списке, если R принадлежит хвосту L.

Один из вариантов Пролог-процедуры можно представить в следующем виде:

```
member(R,L):-L=[H|T], H=R.
member(R,L):- L=[H|T], member(R,T).
```

Цель L==[H|T] в теле обоих правил служит для того, чтобы разделить список L на голову и хвост. Можно улучшить процедуру, если учесть тот факт, что Пролог сначала сопоставляет с целью заголовок правила, а затем пытается согласовать его тело. Тогда улучшенный вариант этой же процедуры может быть записан в виде:

```
member R, [R |_]),
member R,[_ | T ]) :- member(R, T).
```

Использование анонимных переменных (_) вызвано тем, что при применении первого правила нас не интересует хвост списка, а при применении второго - голова списка.

В качестве примера рассмотрим работу процедуры при выполнении цели:

```
member(d,[a,b,d,c]).
```

Цель начинает сопоставляться с первого правила, но так как d не сопоставимо с a, то происходит откат ко второму правилу.

Переменная T получает значение [b,d,c], и порождается цель: member(d, [b,d,c]). Так как первое правило не сопоставимо, происходит повторное сопоставление второго правила и выделение нового хвоста списка.

Текущей целью становится member(d,[d,c]). Использование первого правила приводит к тому, что проверяемый элемент d сопоставляется с головой списка. [d,c]. Первое правило становится истинным, т.е. цель достигнута.

Если исходная цель member(d,[a,b,c,e]), то процесс сопоставления со вторым правилом будет рекурсивно повторяться до тех пор, пока не будет достигнута цель member(d,[]). Ни одно из правил не сопоставимо с этой целью и, поэтому исходная цель оказывается ложной.

Соединение двух списков. Для соединения двух списков определим отношение append(L1,L2,L3), где L1 - исходный список, L2 - присоединяемый (добавляемый) список, а L3 - список, получаемый при их соединении. Задача соединения списков формулируется следующим образом:

Граничное условие: Присоединение списка L2 к [] дает тот же список L2.

Рекурсивное условие: Список L2 присоединяется к хвосту L1, а затем спереди добавляется голова L1.

Тогда непосредственно из постановки задачи можно написать процедуру вида: append([],L2,L3).

Однако, как и в предыдущем случае, append(L1,L2,L3):- L1=[H|T], воспользуемся тем, что Пролог сначала append(T,L2,X), сопоставляет с целью заголовок правила, а L3=[H|X]. затем пытается согласовать его тело. Тогда усовершенствованный вариант этой же процедуры будет записан в виде:

```
append( [ ], L2, L2 ).
append( [H | L1 ], L2, [H | L3]) :- append(L1, L2, L3).
```

На запрос append([a,b,c],[d,e],L) будет получен ответ L=[a,b,c.d,e], а на запрос append([a,b],[c],[a,c,d]) ответом будет "нет".

Данную процедуру можно также использовать для поиска в списке комбинации элементов, отвечающей некоторому условию, задаваемому в виде шаблона. Например, можно найти все месяцы, предшествующие данному, и все, следующие за ним, сформулировав такую цель:

```
Goal: append(X,["апр"|Y],["янв","фев","март","апр","май","июнь"])
X=["янв","фев","март","апр"]
Y=["май","июнь"]
```

Удаление элемента. Для удаления элемента X из списка L введем отношение delete(X,L,L1), где L1 - сокращенный список. Задача будет сформулирована в виде:

Граничное условие: Если X - голова, то результатом удаления будет хвост списка.

Рекурсивное условие: Если X находится в хвосте списка, тогда его следует удалить из хвоста списка.

```
delete( X, [X | T], T ).
delete(X,[Y|T],[Y,T1]):-delete(X, T, T1).
```

Получение п-ого терма из списка. Задача доступа к заданному номером элементу списка формулируется следующим образом:

Граничное условие: Первый терм в списке (H|T] есть H.

Рекурсивное условие: N-й терм в списке [H|T] является (N-I)- м термом в списке T.

```
term_in_list( [H | _],1,H ).
term_in_list([_|T], N, X ) :- M=N-1. term_in_list(T, M, X).
```

Определение длины списка" Задача формулируется следующим образом:

Граничное условие: Длина пустого списка равна нулю.

Рекурсивное условие: Длина списка [H|T] на единицу больше длины списка T.

В предикате list_len второй аргумент принимает

```
list_len([],0).
list_len([_|T],Len):- list_len(T,Len1), Len = Len1+1.
```

значение равное длине списка. Если второй аргумент является связанной переменной, то идет проверка на ее совпадение с длиной списка.

Задание 5

Используя любую из моделей описания предметной области списковыми структурами, разработать программу, позволяющую выполнять над списками все элементарные действия, процедуры которых приведены выше.

Компоновка данных в список

Часто при работе с базами данных встает задача преобразования структур исходных отношений для выполнения тех или иных операции над ними. Одной из таких задач является выбор данные ж базы в список для последующей обработки.

В составе Турбо-Пролога для этих целей предусмотрен стандартный предикат findall (найти_все). Данный предикат вычисляет все ответы на запрос и возвращает их в виде списка. Синтаксис этого предиката имеет вид:
findall(Variable, Predicat_expression, List_name) ,
где List_name - имя переменной выходного списка ответов на запрос,

Predicat_expression - запрос с переменными, записанным в виде некоторого предикатного выражения

Variable - объект предикатного выражения Predicat_expression, определяющий структуру элемента списка List_name

Для пояснения использования данного предиката для преобразования структур данных рассмотрим пример (программа 15_1).

```
/* программа 15_1 */
domains
number, salary=integer
name=string
list_worker=name*
predicates
work(name,number,salary)
clauses
work( "Петров", 101. 500 ).
work( "Павлов", 211,400 ).
work("Сидоров",101,300).
work("Иванов". 101,200).
```

Имеем базу данных work(), описанную соответствующим предикатом и заданную набором фактов. Так как обработка фактов производится всегда в том порядке, как они заданы в программе, то могут возникнуть неудобства, если нужно будет сортировать, упорядочивать и т.л. фамилии сотрудников.

Для этих целей удобнее использовать списковые структуры организации данных.

Пусть для нашего примера требуется сформировать список фамилий сотрудников определенного отдела.

Для этого требуется описать структуру этого списка в секции domains и использовать предикат findall в следующем виде:
findall(Name, work(Name,101,_), List_Name),

Здесь Name является свободной переменной для значений фамилий, которые удовлетворяют запросу в виде предикатного выражения work(Name, 101,_). Кроме того, переменная Name определяет, что элементами списка List_Name будут фамилии. В результате выполнение предиката findall список List_Name будет содержать фамилии всех служащих 101 отдела.

Задание 6

Для программы 15_1 выполните внешний запрос по формированию списка сотрудников заданного отдела. Задайте запрос на формирование списка окладов некоторого отдела.

Приведенный пример показывает, что предикат findall обеспечивает фильтрацию, поиск и формирование списка данных, которые удовлетворяют

условию поиска в БД. Однако, данный пример иллюстрировал только сам процесс поиска и преобразования данных, не связывая его с процессом обработки.

Рассмотрим еще один пример, который должен иллюстрировать тот случай, когда задача обработки данных определяет необходимость в преобразовании их структур.

```
/* программа 15_2 */
domains
number,salary=integer
name=string
list_salary=salary*
predicates
work(name,number,salary)
sum_list(list_salary,salary,integer)
show_sum
find_sum( number)
clauses
show_sum:-makewindow( 1,7.15, "Зарплата:",5, 10,12,30), cursor(2,1),
write( "Введи номер отдела -> "), readint(Otd), find_sum(0td), readchar(_).
find_sum(0td):-findall(Many,work(_,Otd,Many),Lmany),
sum_list(Lmany,Sum,Member), write("общий фонд : ",Sum),nl, write("
служащих : ",Member),nl, Ave=Sum/Member, write( "средняя з/п: ",Ave).nl.
sum_list([],0,0).
sum_list([H|T],Sum,Num):- sum_list(T,S,N), Sum=H+S, Num=N+l.
work("Петров",101,500). work("Павлов",211,400). work("Сидоров",101.300).
work("Иванов", 101,200).
```

Пусть требуется на основе базы данных work() найти общий фонд зарплаты и среднюю зарплату каждого из отделов. Программа 15_2 решает эту задачу.

Предикат show_sum дает возможность ввести номер нужного отдела и обеспечить нужный расчет, обратившись к предикату find_sum().

Предикат find_sum() па основе фактов базы work() формирует список окладов сотрудников отдела - Lmany, вычисляет сумму элементов списка и их количество, выводит полученные данные вместе с вычисленным средним значением на экран.

Сумму элементов списка находит предикат sum_list. Он же подсчитывает число элементов в списке.

В предикате sum_list реализована рекурсивная процедура, аналогичная той, что была использована при описании процедуры поиска длины списка.

Задание 7.

Доработайте программу 15_2 так, чтобы до вывода итоговых данных по отделу R целом в окне экрана дисплея выводился список всех фамилий для сотрудников этого отдела.

6. Содержание отчета по лабораторной работе

Отчет по лабораторной работе должен содержать:

Программы для заданий 1 и 2, запросы к ним и результаты их обработки.
Текст программы обработки списков по заданию 5. Рекомендуется описание типовых процедур обработки списков организовать в виде отдельного программного файла, который мог бы подключаться к любой программе, работающей со списками.
Полный текст доработанной программы 15_2 в соответствии с заданием 7.
Результаты выполнения заданий 3, 4 и 6.

Контрольные вопросы

1. Как отражается применение списков в языке Turbo Prolog?
2. Какие основные компоненты содержит структура список?
3. Могут ли списки включать другие списки?
4. Как формулируется рекурсивное условие для проверки вхождения терма в список?

3 Способы представления баз данных в Пролог-программах

3.1 Лабораторная работа 7

Тема: Способы представления баз данных в Прологе

Цель работы:

Знакомство с различными способами организации баз данных.

Получение навыков доступа к отдельным целостным информационным элементам баз данных.

В языке Пролог существует несколько различных способов представления базы данных. К основным из них следует отнести представление базы данных как:

1) множество фактов, каждый из которых соответствует целостному информационному элементу (т.е. записи) базы данных;

2) множество фактов, каждый из которых соответствует паре значений атрибут/ключ;

3) список структур, в котором каждая структура соответствует записи базы данных;

4) линейная рекурсивная структура, где каждая структура соответствует записи базы данных;

5) рекурсивная структура в виде двоичного дерева, в которой каждый узел дерева соответствует записи базы данных.

Выберем предметную область, например, СЛУЖЕБНЫЕ ОТНОШЕНИЯ, в которой сосредоточены взаимосвязи и сведения о сотрудниках, местах их работы и должностях, а также их должностных окладах.

Для этой предметной области рассмотрим различные варианты структурной организации данных с использованием каждого из пяти названных способов представления базы данных.

Для каждого из возможных представлений попробуем реализовать запрос, позволяющий найти всех сотрудников определенного отдела.

Представление отношений в виде фактов

Модель любой предметной области может быть представлена совокупностью объектов с указанием связей между ними. Используя реляционный подход к построению БД, любой объект может быть интерпретирован как отношение, достаточно полно описывающее некоторый аспект предметной области и устанавливающее функциональную зависимость атрибутов этого отношения от некоторого ключевого атрибута. Для рассматриваемого примера таким отношением может быть

РАБОТА (ИМЯ_СТУЖАЩЕГО, ОТДЕЛ, ДОЛЖНОСТЬ, ОКЛАД),

где в качестве ключа может рассматриваться атрибут - ИМЯ_СЛУЖАЩЕГО. Отношение представляет собой множество кортежей, каждый из которых

соответствует определенному экземпляру объекта предметной области и может быть представлен в виде целостного информационного элемента.

Простейшим способом представления базы данных в языке Пролог служит запись каждого целостного информационного элемента в виде факта, например:

```
work1 ("Петров", 101, "оператор", 20000).
work1 ("Павлов", 211, "начальник", 71000).
work1 ("Иванов", 101, "менеджер", 71500).
```

Запрос work1 (Name, 101,Post,Salary) позволяет отыскать всех служащих 101 отдела.

Представление атрибутов в виде фактов

Вместо того, чтобы записывать факты, содержащие целостные информационные элементы, можно использовать в качестве фактов отдельные атрибуты (т.е. свойства) этих информационных элементов. В случае необходимости данные атрибуты можно собрать в единое целое при помощи правила. Один из атрибутов должен выступать в роли ключа, объединяющего все остальные свойства. Атрибут ИМЯ_СЛУЖАЩЕГО можно использовать в качестве ключа для БД work(). Первый экземпляр объекта РАБОТА можно представить таким образом:

```
office ("Петров". 101).        /* отдел     */
post ("Петров", "оператор").   /* должность */
salary ("Петров", 20000).      /* оклад     */
```

Все атрибуты для конкретного служащего, совокупность которых представляет собой определенный экземпляр объекта предметной области, можно объединить при помощи правила:

```
work2 (Name,Office,Post,Salary) :- office ( Name, Office ), post ( Name, Post), salary ( Name, Salary ).
```

Это правило определяет неявную базу данных того же вида, что и приведенная ранее явная база данных work1(). Для этого правила можно воспользоваться предыдущим запросом, который даст те же результаты, что и ранее.

Применение атрибутов является более гибким средством представления базы данных, чем применение целостных информационных элементов, поскольку новые атрибуты можно добавлять без переписывания заново всей существующей базы данных.

Приведем пример правила, в котором предписывается выдача премии в 1000$ всем служащим отдела 101. Это можно описать с использованием вновь вводимого атрибута "премия", предикат для которого можно определить в виде правила:

```
prize (Name, 1000):- office (Name, 101).
```

Для некоторых служащих можно ввести дополнительный атрибут "стаж_работы" и некоторый набор фактов, его описывающий:

```
work_time ( "Петров", 2 ).
work_time ( "Иванов", 1 ).
```

Добавление новых атрибутов "премия" и "стаж_работы" никак не повлияет на приведенное выше правило work2().

Задание 1

Составьте программу, позволяющую реализовать описанные в данном разделе структуры БД, запросы к ним и возможность их модификации. Сохраните программу в "lab7_1.prg"

Представление базы данных в виде списке, структур

Базу данных можно также считать потоком целостных информационных элементов, что представляется на языке Пролог в виде списка структур. Каждый элемент списка - это целостный информационный элемент:

[work ("Петров". 101. "оператор" , 20000) , work ("Павлов", 211, "начальник", 71000) , work ("Иванов", 101, "менеджер" , 71500)]

Интересно отметить, что при таком подходе к структуре базы данных поток целостных информационных элементов lie нужно включать в текущую программу. Он может существовать лишь как аргумент запроса, входящий в различные подцели, которые обрабатывают этот поток.

Доступ к отдельному информационному элементу такой БД будет осуществляться с использованием специальной процедуры, содержащей два аргумента. Входным будет второй аргумент процедуры. Он содержит список целостных информационных элементов. Результат выполнения процедуры возвращается через ее первый аргумент - по одному целостному информационному элементу за один ответ на запрос (т.е. по одному экземпляру объекта, или по одной записи).

```
record ( work(Name.Office.Post.Salary). [work(Name.Office.Post.Salary) п_]).
record ( work(Name,Office,Post,Salary), [work(_,_,_,_) | Tail]) :- record ( work(Name,Office,Post,Salary), Tail).
```

Описание процедуры record() аналогично процедуре member(). С использованием процедуры record() можно составить запрос о всех служащих отдела 101. Во втором аргументе этого запроса полностью содержится вся база данных.

```
Goal:
record(work(Name,101,Post,Salary), [work("",101,"оператор",20000), work("", 211,"начальник",71000), work("",101,"менеджер",71500)]).
```

Задание 2

Составьте программу, позволяющую реализовать данный запрос к БД в виде внешней цели.

Включите БД в виде списка структур в состав программы и выполните к ней аналогичные запросы. Сохраните программу в файле "1ab7-2.prg".

Попробуйте включить в состав программы процедуры модификации содержимого исходной БД (например, добавления и удаления элемента из списка) и сформируйте к программе ряд запросов. Сохраните программу в файле "lab7-3.pro". Тексты программы: запросы и ответы привести в отчете.

Представление базы данных в виде линейной рекурсивной структуры

Еще одним из способов представления в памяти ЭВМ записей баз данных является использование рекурсивных структур, в которых один из ар1ументов каждой записи указывает на следующую запись. Рекурсивные структуры - аналог односвязных списков в других языках программирования, но работать с такими структурами в Прологе легче, так как он берет на себя все действия по обработке указателей.

Для того, чтобы на Прологе создать рекурсивную структуру, состоящую из записей work() ("служащие"), единственное, что нужно - это ввести в эту структуру дополнительный аргумент, указывающий на следующую запись, в структуре которой также будет содержаться аргумент, указывающий на следующую за ней запись и т.д. Для указания в последней записи отсутствия следующей за ней записи, в качестве дополнительного аргумента вводится "end". Рассмотрим пример линейной рекурсивной структуры Служащий (Имя , Отдел, Должность , Оклад , Указатель на следующую запись).

Используя синтаксис языка Пролог, обозначим эту структуру work3(). По аналогии с ранее рассмотренным примером, она будет содержать три записи

work3("Петров", 101, "оператор", 20000,
work3("Павлов", 211, "начальник". 71000,
work3("Иванов", 100, "менеджер". 71500, end)))

1 запись 2 запись 3 запись конец записей

Заметьте, что уровень вложенности у каждой новой записи будет большим, чем v предыдущей. В пятом поле последней записи содержится слово "end", которое означает, что больше записей нет. Следует отметить, что дополнительный аргумент структуры, указывающий на следующие записи базы данных, сам имеет структуру этой записи. Т.е. структура записи определяется сама через себя: Служащий (Имя , Отдел, Должность , Оклад , Служащий).

База данных, сформированная в виде рекурсивной структуры, должна позволять выполнять запросы и модифицировать данные. Одним из основных требований является возможность доступа к целостному информационному элементу (записи) БД. Рассмотрим вариант процедуры record(), которая модифицирована таким образом, чтобы стала возможной обработка рекурсивной структуры.

Первым (выходным) аргументом новой процедуры является структура work(), содержащая четыре аргумента и соответствующая записи базы данных. Второй аргумент этой процедуры - это сама БД в виде рекурсивной структуры. Процедура состоит из двух правил. Первое из них строит структуру work() из верхнего уровня рекурсивной структуры данных. Второе правило игнорирует верхний уровень рекурсивной структуры и получает следующий целостный информационный элемент из оставшейся части базы данных (обозначенной переменной NextRecord).

record(work(Name,Office,Post,Salary), work3(Name, Office, Post, Salary, NextRecord)).

record(work(Name,Office,Post, Salary), work3(_,_,_,_, NextRecord)) :- work3 (work(Name,Office,Post,Salary),NextRecord).

Используя процедуру record(), можно сформировать внешнюю цель для запроса по отысканию всех служащих 101 отдела, при условии, что вторым аргументом процедуры будет вся база данных.

Goal:

record (work(Name,101,Post,Окл), work3("Петров",101,"оператор",20000, work3("Павлов",211,"начальник"71000,work3("Иванов",l00, "менеджер", 71500,end))).

Ответ на этот запрос будет получен в виде

Name=Петров,
Post=оператор,
Salary=20000
Name=Иванов,
Post=менеджер,
Salary=71500

Аналогичный вариант запроса реализован как внутренняя цель print_101 программы 16.

```
/* Программа 16 *)
domains
name = symbol
office = integer
worker = work(name,office)
/* описание рекурсивной структуры */
work3 = work3(name,office,work3) ; end
predicates
print_10l
record( worker, work3 )
goal
print_101 clauses record( work(N,O) , work3(N,0,_) ).
record( work(N,0) , work3(_,_,T) ) :- record( work(N,0) ,T).
print_101:- write(" Сотрудники 101 отдела"),nl.
record ( work(N,101), work3("Петров",101, work3("Павлов",211, work3("Иванов",101,end)))), write(n),nl, fail.
```

Эта программа работает с БД work3(), представленной в виде линейной рекурсивной структуры.

Для упрощения программы в БД work() используется всего два ПОЛЯ (ФАМИЛИЯ и ОТДЕЛ), т.е. информационные элементы work(), содержат всего по две компоненты.

Процедура record() позволяет выполнять доступ к отдельной записи БД.

Задание 3

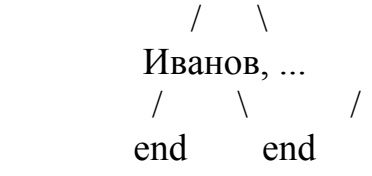
Тщательно разобравшись со структурой программы 16. измените ее таким образом, чтобы она позволяла работать со структурой БД, описанной в данном разделе и выполнять запросы к ней. Сохраните программу в файле "lab7-4.pro"

Представление базы данных в виде двоичного дерева

Можно еще более усовершенствовать метод представления данных в виде рекурсивной структуры, если преобразовать структуру work3() в структуру типа двоичное дерево. Это достигается путем введения в структуру записи БД еще одного дополнительного аргумента, который указывает на предыдущую запись.

Смысл использования двоичного дерева заключается в том, чтобы хранить базу данных в отсортированном виде. В рассматриваемом ниже примере база данных отсортирована по атрибуту, описывающему имя служащего и представляется бинарным деревом вида:

Теперь в структуре информационного элемента будет шесть аргументов:



work4(LeftTree, Name,Office,Post,Salary, RightTree)

Переменная LeftTree описывает ветвь дерева, которая содержит все целостные информационные элементы, стоящие (по алфавиту) перед текущим элементом. Переменная RightTree описывает ветвь дерева, охватывающую все целостные информационные элементы, расположенные согласно алфавиту, после данного элемента. ~ Ввиду того, что БД отсортирована, целостный информационный элемент, содержащий вес сведения о Павлове, является самым верхним узлом дерева, а саму БД, представленную в виде двоичного дерева, можно записать следующим образом:

work4 (work4(end, "Иванов", 101,"менеджер",71500, end), "Павлов", 211,"начальник",71000, work4(end,"Петров", 101,"оператор", 20000,end),)

Версия процедуры record() для доступа к целостному информационному элементу БД, представленной в виде двоичного дерева, будет базироваться на трех правилах:

- запись принадлежит дереву, если она находится в левом поддереве,
- запись принадлежит дереву, если она является корнем этого дерева,
- запись принадлежит дереву, если она находится в правом поддереве.

Эту процедуру, используя синтаксис Пролога, можно описать следующим образом:

```
record ( work(Name,Office,Post,Salary), work4(LeftTree, _,_,_,_, __)):- record
                ( work(Name, Office, Post, Salary), Let: Tree ).
record { work(Name,Office,Post,Salary), work4( _, Name.Office.Post.Salary, _) ).
record ( work(Name,Office,Post,Salary), work4(_, _,_,_,_, RightTree) ) :- record
                ( work(Name,Office,Post,Salary), RightTree ).
```

Теперь процедура record() определена, и можно написать запрос, позволяющий найти всех служащих отдела 101. Вторым аргументом запроса является целиком вся база данных.

```
Goal:
record (work(Name, 101,Post, Salary),
work4 (
work4(end, "Иванов", 101, "менеджер",71500. end), "Павлов",211,"начальник",
71000, work4(end, "Петров", 101,"оператор",20000,end).))
```

Ответ на запрос будет тот же самый, что и в предыдущих случаях. Одним из интересных свойств представления в виде двоичного дерева является то, что процедура record() всегда будет выдавать целостные информационные элементы, образующие дерево, в отсортированном порядке. Это иллюстрирует запрос

```
Goal:
record (OneRecord, work4 (work4(end, "Иванов", 101, "менеджер",71500, end),
"Павлов",211,"начальник",71000, work4(end,"Петров", 101 ."оператор",
20000,end),) OneRecord = work("Иванов"10l,"onepатор",20000); OneRecord =
work("Павлов",211, "начальник", 71000); OneRecord = work("Петров",101,"
менеджер ",71500);
```

Задание 4

Измените программу "1ab7-4.pro" для работы с БД, которая представляется в виде двоичного дерева и позволяет реализовать запросы к ней, описанные в данном разделе.

Включите содержимое исходной БД в виде двоичного дерева в состав ир01раммы и выполните к ней запросы, аналогичные выше приведенному. Сохраните программу в файле "lab7_5. pro". Текст программы, запросы и ответы на них привести в отчете.

7. Сравнение разных видов представления базы данных

Наиболее важным различием между описанными формами представления БД является то, что для доступа к данным в каждом случае требуется свой алгоритм.

Говоря конкретно, при представлении БД в виде целостных информационных элементов, являющихся фактами, или при представлении атрибутов в виде фактов, доступ к БД должен осуществляться при помощи алгоритма поиска с возвратом (backtracking algorithm), а при использовании рекурсивных структур данных (в том числе, списков и двоичных деревьев) доступ должен реализоваться рекурсивным алгоритмом. Правила и запросы, приведенные в данной работе, служат простыми примерами этих алгоритмов.

Представление в виде двоичного дерева имеет то преимущество, что время поиска конкретной записи будет обычно меньше, чем при иных представлениях. Ввиду того, что древовидная структура содержит данные в отсортированном виде, процедуре выбора для поиска интересующей записи понадобиться просмотреть меньшее количество записей.

Эффективность работы с БД в виде двоичных деревьев во многом определяется знаниями процедур их создана и преобразования. Рассмотрим

ряд процедур, оперирующих с двоичными деревьями, описываемыми структурами вида
tree(Left , Root , Right),
где Left - левое поддерево, каждый элемент которого имеет структуру, аналогичную tree(),
Root - корень (любая произвольная, определяемая пользователем, структура),
Right - правое поддерево, состоящее из элементов структуры tree().

Построение бинарного дереве. Задача создания упорядоченного дерева при добавлении некоторого элемента X к другому упорядоченному дереву может быть сформулирована следующим образом:

Граничное условие: Добавление X к пустому дереву дает tree(end,X,end).

Рекурсивные условия: При включении X в tree(Left,Root,Right) надо рассмотреть два случая, для того, чтобы результирующее дерево было упорядоченным.

1 Если X меньше, чем Root, то X добавляется к любому поддереву.

2 Если X больше, чем Root, то X следует добавить к правому поддереву. В обоих случаях значения корня и противоположного поддерева не меняются. Такой формулировке соответствует процедура insert(), которую запишем в виде:

```
insert( end, X, tree( end, X, end ) ).
insert( tree(Left, Root, Right ), X, tree(LenNew,Root,Right) ) :- X < Root, insert( Left, X, LeftNew ).
insert( tree(Left,Root,Right), X, tree(Left.Root,RightNew) ) :- X > Root, insert( Right, X, RightNew ).
```

Построение бинарного дерева из списка. Процедуру insert() можно использовать для построения упорядоченного дерева из списка. Процедура, обеспечивающая преобразование списка в упорядоченное дерево, будет иметь вид:

```
list_to_tree([], end ).
list_to_tree([H|T],AllTree) :- list_to_tree( T, Tree), insert(Tree, H, AllTree).
```

Построение отсортированного списка из дерева. Для решения этой задачи можно воспользоваться упорядоченным бинарным деревом и объединением списков.

Граничное условие: Пустое бинарное дерево (end) приводит к пустому списку [].

Рекурсивное условие: Отсортированный список для упорядоченного бинарного дерева tree(Left,Root,Right), где Left имеет отсортированный список L1, а Right имеет отсортированный список L2, получается присоединением [Root |L2] к L1.

```
tree_to_list( end , [] ).
tree_to_list( tree(Left,Root,Right), List ) :- tree_to_list( Left, L1), tree_to_list( Right, L2), append( L1, [Root п L2], List).
```

Рассмотрим пример использования этих процедур в Пролог-программе, работающей с базами данных разных структур и преобразующей эти структуры.

```
/* Программа 17 */
domains
worker = symbol
listworker = worker*
tree = tree(tree,worker,tree) ; end
predicates
db1( tree )
db2( listworker )
record( worker, tree )
append( listworker, listworker, listworker)
insert( tree, worker, tree)
list_to_tree( listworker, tree )
tree_to_list( tree, listworker)
goal1
goal2
goal3
goal4
goal5
goal6
goal7
goal8
clauses
goal1 :- db1(DB), write(DB).
goal2 :- db1(DB), record(R,DB), write(R), nl, fail.
goal3 :- db1(DB), write("введи элемент "), readln(El), insert(DB ,E1,DBnew), write(Dbnew).
goal4 :- dbl(DB), write("введи элемент "), readln(El), insert(DB,El,DBnew), record(R,DBnew), write(R), nl, fail.
goal5 :- db2(List), list_to_tree(List,Tree), write(Tree).
goal6 :- db2(List), list_to_tree(List,Tree), record(R,Tree), write(R), nl, fail.
goal7 :- dbl(Tree), tree_to_list(Tree,List), write(List).
goal8 :- db2(List), write(List), nl.
list_to_tree(List,Tree). write(Tree), nl,
tree_to_list(Tree,NewList), write(NewList), nl.
insert( end, X, tree(end, X,end) ).
insert( tree(L,Root,R). X, tree(LNew,Root.R)) :- X < Root.
insert(L,X, Lnew ).
insert( tree(L,Root,R). X, tree(L,Root,RNew)) :- X > Root.
insert( R,X, RNew).
list_to_tree( [] , end ).
list_to_tree([HпT], AllTree) :- list_to_tree( T, Tree ),
insert( Tree, H, AllTree ).
```

```
tree_to_list( end , [] ).
tree_to_list(tree(L,Root,R),List) :- tree_to_list( L, L1),
tree_to_list( R, L2),
append( L1, [ Root | L2 ], List ).
record( R, tree(LeftTree, _,_)):- record( R, LeftTree).
record( R, tree( , R,_) ).
record( R, tree( _, _,RightTree)):- record( R, RightTree).
append( [], L2, L2 ).
append( [HпL1], L2. [H|L3]) :- append(L1, L2, L3).
dbl( tree( tree(end,a,end), c, tree(end,e,end) ) ).
db2([c, e, a]).
```

Использование описанных выше процедур позволяет как модифицировать базы данных, так и осуществлять их структурные преобразования. Программа 17 даст возможность исследовать простейшие преобразования над базами данных.

В этой программе используются два способа представления баз данных: в виде списка и в виде двоичного дерева. Причем структуры целостных информационных элементов обеих баз данных приняты одинаковыми и, с целью упрощения, включают в себя всего по одному полю символьного типа.

Обе базы задаются непосредственно в программе с помощью предикатов db1() и db2(). Простейшие преобразования над базами иллюстрируются с помощью восьми целей, сформированных непосредственно в программе. Каждая из этих целей может быть вызвана из оболочки Турбо-Пролога как внешняя цель. Рассмотрим назначение каждой из сформированных в программе целей.

Первая цель (goal1) позволяет просмотреть содержимое первой базы данных заданной в виде бинарного дерева.

Вторая цель (goal2) позволяет последовательно просмотреть записи первой базы данных, что достигается выделением из БД отдельных информационных элементов с помощью процедуры record().

Третья и четвертая цели (goal3 и goal4) иллюстрируют возможность модификации первой БД путем добавления в нее нового элемента с сохранением упорядоченности.

Пятая и шестая цепи (goal5 и goal6) показывают, как база данных, заданная в виде списка, может быть преобразована в структуру типа бинарного дереза.

Седьмая цель (goal7) иллюстрирует возможность преобразования упорядоченное дерево в отсортированный список.

Восьмая цель (goal8) показывает, как двойное преобразование структуры БД позволяет отсортировать исходный список. На первом этапе исходный список преобразуется в бинарное дерево. При этом обеспечивается упорядоченность этого дерева. На втором этапе бинарное дерево преобразуется в список, но так как дерево упорядочено, то и список получается отсортированным.

Задание 5

Загрузите программу 17. Тщательно разберитесь с описанием структур и предикатов программы, а также назначением ее процедур. Исследуйте программу, выполнив описанные в ней цели. Сформируйте ряд новых целей, позволяющих исследовать преобразование структур баз данных. В программе 17 используется простейшая структура информационных элементов. Измените программу для работы со структурами типа work(). рассмотренных в предыдущих разделах данной лабораторной работы. Сохраните программу в файле "lab7-5.pro". Текст программы, запросы и ответы на них привести в отчете.

Все процедуры, используемые в программе 17, описаны в данной работе. Исключение составляет процедура слияния двух списков append(), подробное описание которой приведено в предыдущей лабораторной работе, посвященной работе со списками.

Содержание отчета по лабораторной работе

Отчет по лабораторной работе должна содержать:

Тексты программ, сохраненных на диске в соответствии с заданиями.

Запросы, сформированные самостоятельно при изучении данной работы, а также результаты из выполнения.

Результаты выполнения заданий 1, 2, 3, 4. 5 данной лабораторной работы.

Приведите четкое описание каждой из целей, исследованных при работе с программой 17.

3.2 Лабораторная работа 8

Тема: Динамические базы данных в Турбо-Прологе

Цель работы:

Знакомство с назначением и использованием динамических баз данных.

Изучение правил описания и организации динамических БД в Турбо-Прологе.

Получение навыков работы с динамическими БД" записи их содержимого на диск и считывания с диска.

Знакомство с использованием динамических БД при разработке программ, обучающихся в процессе работы.

В Прологе реляционная база данных представляется в виде набора фактов, что позволяет использовать Пролог, как мощный язык запросов для баз данных. Его алгоритм унификации автоматически выбирает факты с правильными значениями для известных параметров и унифицирует значения для любых неизвестных параметров. Алгоритм поиска с возвратом позволяет определить все решения для формируемых запросов.

В предыдущих работах рассматривались статические базы данных, где факты являлись частью кода программы и не могли быть изменены во время работы с программой. Принципиальным отличием, рассматриваемых ниже,

динамических баз данных является то, что во время работы с программой, из них можно удалять любые, содержащиеся в ней утверждения, а также добавлять новые. Другая важная особенность динамической базы состоит в том, что она может быть написана на диск и считана с диска в оперативную память.

Факты, принадлежащие динамическим базам данных (ДБД), обрабатываются отличным от обычных предикатов образом для того, чтобы ускорить работу с базами данных большого объема. Предикаты ДБП отличаются от обычных тем, что они описываются в отдельной секции программы - database. Манипулирование фактами в ДБД осуществляется с использованием трех стандартных предикатов.

asserta(Term) - добавляет новые факты в начало динамической базы данных.
assertz(Term) - добавляет новые факты в конец динамической базы данных,
retract(Term) - удаляет факт из динамической базы данных.

Для модификации какого-либо факта в базе данных необходимо сначала удалить его, а затем добавить в измененном виде. Стандартные предикаты:
save("file_name") - сохранение ДБД в текстовом файле с именем file_name.
consult("file_name ") - загрузка ДБД из текстового файла с именем file_name,
обеспечивают обмен между ОЗУ и ВЗУ данными, содержащимися в динамической базе данных.

Следует отметить, что иногда часть информации базы данных предпочтительно иметь в виде утверждений статической БД и заносить эти данные в динамическую БД сразу же после активизации программы. При этом предикаты статической БД должны иметь другое имя, но ту же самую структуру данных. что и предикаты динамической БД. Причем описание предикатов ДБД производится в секции database, а предикатов статической БД - в секции predicates.

2. Простейшие приемы работы с динамическими БД

В секции database декларируются предикаты, описывающие динамическую базу данных. Описание предикатов динамической БД должно предшествовать описанию всех обычных предикатов.

В приведенной ниже программе 18 рассмотрен пример использования явной динамической базы данных, структура которой соответствует отношению

ЛИЧНОСТЬ (ИМЯ, ВОЗРАСТ, ПОЛ).

На основе явной динамической БД, которая допускает модификацию во время работы, в программе с помощью правил определены еще три неявные базы данных:

МУЖЧИНА (ИМЯ, ВОЗРАСТ), ЖЕНЩИНА (ИМЯ, ВОЗРАСТ). РЕБЕНОК (ИМЯ, ВОЗРАСТ, ПОЛ)

Предикат person() можно использовать точно так же, как и любые другие предикаты. Отличие лишь в том, что для него, во время выполнения программы, возможно добавление и удаление фактов. Факты, добавляемые таким образом, сохраняются в ОЗУ ЭВМ. Используя программу 18,

рассмотрим простейшие приёмы работы с ДБД. Если загрузить в память ЭВМ

```
/* Программа 18 */
domains
name= string
age = integer
sex = m; f
/* домены m и f состоят только из одного функтора */
database
person(name,age,sex)
predicates
male(name,age) female(name.age) child(name,agc,sex)
clauses
male(Name,Age):-person(Name,Age,m).
female(Name,Age):-person(Name,Age,f).
child(Name,Age,Sex):-person(Name,Age,Sex), Age<16.
```

эту программу и сформировать к ней запрос

```
Goal: person(X,Y,Z):
```

то Турбо-Пролог ответит "нет" что вызвано отсутствием данных в ДБД. Если же теперь ввести последовательно две цели

```
Goal: asserta(person(tom,20,m))
Goal: asserta(person(mery,18,f ))
```

и ввести первоначальный запрос, то ответом на него будет:

```
X=mery     Y=18     Z=f
X=tom      Y=20     Z=m
```

Полученный ответ характеризует наличие в БД person() сведений о двух лицах, их именах, возрасте и поле, хотя никаких дополнительных фактов в текст программы не вводилось. В этом принципиальное отличие ДБД, которые организуются Турбо-Прологом в памяти ЭВМ, свободной от Пролог-программы.

Занести информацию о Томе и Мери в ДБД нам позволил предикат asserta(). Особенность его работы видна из сопоставления двух целей и результата запроса. Данные о Томе вводились первыми, а затем данные о Мери. В ответе - первыми являются данные о Мери. Это связано с тем, что asserta(person(mery,18,f)) внес данные о Мери в начало динамической БД. Добавим в БД информацию еще о двух лицах, последовательно вводя еще две цели вида:

```
Goal: assertz( person( nick, 10, m))
Goal: asserta( person( ada , 9 , f ))
```

Если теперь сформулировать запрос о выдаче всей информации, хранящейся в ДБД pcrson(), то ответ на него будет получен в следующей форме:

```
Goal: person(Name,Age,Sex),
Name=ada          Age=9     Sex=f
Name=meryAge=18    Sex-f
```

Name=tom Age==20 Sex=m
Name=nick Age=10 Sex=m
4

Из сопоставления целей добавления и результатов запроса видно, что один из добавленных фактов помещен в начало ДБД, а во второй - в конец, что связано с различием в работе предикатов asserta() и assertz().

С элементами ДБД Пролог может выполнять все операции, допустимые для фактов аналогичной структуры. Их можно искать, унифицировать, использовать в виде подцелей правил и т.д. Так, в данной программе ДБД используется в виде подцели трех правил формирования неявных БД. Используя предикаты неявных БД, можно сформировать запросы о данных по мужчинам, женщинам или детям:

Goal: male(Name,Age)
Goal: female(Name,Age)
Goal: child(N,A,S)
Name=tom Age=20
Narne=nick Age=10
2
Name=ada Age=20
Name=mery Age=10
2
N=ada A=9 S=f
N=nick A=10 S=m
2

или сформировать на их основе любые иные, простые или составные, запросы.

Если потребуется удалить из ДБД какой-либо факт, то для этого следует воспользоваться предикатом retract(). Например, для удаления сведений о Мери следует задать цель:

Goal: retract(person(mery,_,_))

При необходимости скорректировать какой-либо элемент ДБД его следует удалить из базы данных, а затем добавить в ДБД новый модифицированный элемент. В частности, для того, чтобы изменить возраст Тома на один год, надо ввести составную цель:

Goal:
person(tom,OldAge,Sex,Person), retract(person(tom,_,_)), NewAge=OldAge+1,!, asserta(person(tom,NewAge,Sex,Person)) ,

которая удалит старый факт и добавит в ДБД кодифицированный факт. Результат проведенных преобразований ДБД можно получить, как ответ на запрос ,

Goal: person(Name,Age,Sex)
Name=tom Age=21 Sex=m
Name=ada Age=9 Sex=f
Name=nick Age=l0 Scx=m

3

Динамическая база данных целиком может быть сохранена в текстовом файле с помощью вызова предиката save с именем текстового файла в качестве параметра. Например, после выполнения цели:

Goal: save("person.dba")

файл mydata.dba будет похож на обычную программу на Турбо-Прологе с фактами на каждой строке. Такой файл может быть позднее считан в память, используя:

Goal: consult("person.dba").

Заданное действие выполнится, если программа в файле не содержит ошибок. При наличии ошибок считывание из файла данных в динамическую БД не произойдет.

Задание 1.

Загрузите программу 18 и выполните все перечисленные в данном разделе действия по заполнению и модификации динамической базы данных person(), а также по запросом к явной ДБД и неявным БД. Сохраните результирующую ДБД в файле "person.dba", а программу в файле "1ab8-1.pro". Просмотрите содержимое файла "person.dba", а затем снова загрузите программный файл. Запустите программу на выполнение и введите запрос person(X,Y,Z). Какой получился результат и почему? Загрузите ДБД из файла "person.dba" и введите запрос person(X,Y,Z). Какой в этом случае получился результат и почему?

Ранее уже не раз отмечалось, что любая программа на Прологе - это набор фактов и правил, которые представляют собой своеобразную базу данных, на которой Пролог выполняет логический вывод. В свою очередь, динамические базы данных - это набор фактов, который может изменяться при работе программы.

Из этих двух посылок следует важнейший вывод: Пролог допускает изменение программ во время их выполнения! И эти изменения можно выполнять, используя динамические базы данных.

3. Связь статических и динамических баз данных

В предыдущем разделе было показано, что для того, чтобы начать работать с динамической БД, требуется либо доступный для загрузки динамической БД файл на диске, либо начальная загрузка динамической базы вручную. Вызвано это тем, что Пролог формирует динамическую базу в оперативной памяти. Выключение компьютера или просто выход из программы могут привести к потере данных. В этих условиях наиболее ценные данные можно представить фактами статических база данных непосредственно в теле программы.

При использовании такого подхода, для возможности изменения данных в процессе работы программы или добавления новых фактов, следует перезаписать данные из статической в динамическую БД и перейти к работе с ДБД.

В программе 19 используются две базы данных одинаковой структуры

АДРЕС (ОРГАНИЗАЦИЯ, УЛИЦА, ДОМ, ГРУППА_ОРГАНИЗАЦИЙ)

Одна из них является динамической и описана предикатом address() в секции database, вторая - статической и описана предикатом adres() в секции predicates. Предикат place() связывает месторасположение организации в городе с ее адресом.

Процедура p1ace() накладывает ограничение один-ко-многим при доступе к ДБД address(), что обеспечивает целостность отношений при обработке запросов.

Предикаты school() и bank() описывают две неявные БД, обеспечивающие пользовательский интерфейс к динамической БД.

Процедура load_dbd служит для занесения в ДБД address() информации из статической БД adres(). В этой процедуре используется уже знакомый нам метод отката после неудачи, который позволяет перебрать все утверждения предиката adres() и добавить соответствующие им факты в ДБД.

```
/* Программа 19 */
domains
firm, street, grup = symbol
house = integer
database
address(firm, street, house, grup)
predicates
adres( firm, street, house, grup )
place( firm. street, house, grup )
load_dbd
del(firm)
school(firm)
bank(firm)
clauses
adres("ЛТА","Институтский пер.",5,"Вуз").
adres("СП6ЭТУ","ул. проф. Попова" ,5 ,'"Вуз").
adres("ЛесПромБанк","Крапивный пер.",2,"Банк").
load_dbd:- adress(X,Av,N.G),
assertz(adress(X,Av,N,G)).
fail.
load_dbd:- !.
del(Firm) :- retract(address(Firm,_,_,_)).
place(X,St,H,G) :- bound(X),address(X,St,H,G), !.
place(X.St.H,G) :- free(X), address(X,St,H,G).
school(Firm) :- place(Firm,St,H,"Вуз").
write(St," ",H), nl.
bank(Fiim) :- place(FirmX,St,H,"Банк").
write(St," ",H), nl.
```

Предикат del(X) имеет всего один аргумент и определяется правилом, которое удаляет из ДБД факт для организации, имеющей наименование то же, что и аргумент X предиката.

Закончив описание программы 19, перейдем к работе с базами этой программы.

Загрузим программу в память и запустим ее на выполнение. Если ввести запрос:

Goal :adres(Firm,_,_,_) .

то в ответе будет список из трех организаций по числу фактов, входящих в статическую БД. Ответ на запрос:

Goal: place(Firm, _,_,_}

будет "нет", так как нет данных в ДБД, хотя в статической они заданы. Но предикат place() был определен па ДБД. Если до формирования запроса загрузить динамическую базу данных из статической, т.е. сформировать цель

Goal:(load_dbd,place(Firm,_, _,Grup)

то ответом будет

Firm=ЛТА Grup=Вуз

Firm=СПбЭТУ Grup=Вуз

Firm= ЛесПромБанк Grup=Банк

3.

При этом, операцию загрузки динамической БД следует выполнять только один раз во время сеанса работы с программой. Это обусловлено тем, что повторное обращение к предикату load_dbd вызовет добавление динамической базы данных, уже существующим в ней набором фактов. То есть произойдет дублирование данных.

Задание 2

Введите программу и выполните описанные выше запросы. Повторно введите последний запрос и убедитесь в дублировании данных. Приведите ДБД в исходное состояние, используя предикат del() и сформируйте четыре запроса об адресе ЛТА, используя четыре различных предиката (adres, address, place, school). В чем состоит различие запросов? Сформируйте запросы: "Какие организации расположены в домах с номером 5?", "Найдите адреса всех ВУЗов?", "Адрес ПромстройБанка?". Сколько имеется способов построения каждого из этих запросов и в чем их различие? Запросы и их результаты привести в отчете по работе.

Использование неявных баз данных позволяет существенно упростить запросы за счет организации пользовательского интерфейса с ДБД. Его использование не требует от пользователя знания всех атрибутов ДБД и последовательности их описания Б предикате, декларирующем ДБД.

Кроме этого, неявные БД позволяют фильтровать исходную динамическую БД в соответствии с некоторыми групповыми признаками. При этом у администратора БД остается возможность доступа непосредственно к фактам ДБД.

Процедура работы с динамической БД, обучающаяся у пользователя

Реализация последнего запроса задания 2 показывает отсутствие требуемых данных в БД и, как следствие, необходимость их добавления.

Однако добавление новых данных требует выполнения ряда операций, описанных в предыдущих разделах. Для пользователя, не знающего структуру динамической БД, это является невозможным, а для администратора БД это представляет собой нудный трудоемкий процесс.

В этих условиях встает задача построения более интеллектуальной процедуры доступа к динамической БД, которая не только обеспечивала бы ограничения "один-ко-многим", но и обучалась бы у пользователя, формируя динамическую БД.

```
/* Программа 19 */
...
place(X,St,H,G) :- bound(X),bound(G). write("введи для ",X ), nl, write("улица "), readln(St),
write ("дом "). readreal(H),
assertz( address(X,St,H,G) ),!.
place(X,St,H,_) :- bound(X),
write("введи для ",X), nl,
write("улица "), readln(St),
write ("дом "). readreal(H),
writc("группа "), readln(G),
assertz( adress(X,St,H,G) ),!.
...
```

Если в процедуру place() добавить еще два правила, то она не просто завершается неудачей при невозможности найти нужные данные в ДБД address(), а переключается на другую стратегию и получает сведения от пользователя.

При этом пользователь будет выступать в качестве альтернативного источника знаний. Процедура place() учится на своем опыте, вводя новые ответы в ДБД.

Первое правило запрашивает у пользователя адрес конкретной организации, отсутствующей в БД и относящейся к определенной группе. Это имеет место при запросах с использованием неявных баз данных, когда известен некоторый групповой признак, характерный для конкретной неявной БД.

```
введи для ПромСтройБанк улица Невский пр. дом 38
Goal:
bank("ПромСтройБанк")
Невский пр. 38
True
```

Так, если в исходном варианте процедуры place() запрос о ПромСтройБанке оканчивался неудачей (False), то добавление в процедуру p1acc() первого правила приведет к запросу со стороны программы адреса банка и добавления нового факта в динамическую БД address().

Результат выполнения этого действия с программой можно увидеть, если ввести запрос на вывод содержимого всей динамической БД:

```
Goal: address(Firm, _ ,_,Grup)
```

Firm=ЛТА Grup=Вуз
Firm=СПбЭТУ Grup=Вуз
Firm=ЛесПромБанк Grup=Банк
Firm=ПромСтройБанк Grup=Банк
4

Динамическая БД содержит четыре факта, последний из которых теперь соответствует ПромСтройБанку, у которого группа - "Банк", хотя это не вводилось. Эта группа позволяет системе относить новый факт к неявной базе bank(). В этом можно убедиться, сформировав запрос на вывод всех банков Goal:bank(Name).

Если же нас интересует информация об организации, имеющей групповой признак, для которого не определена неявная БД, то можно воспользоваться интерфейсной БД place(). Однако, ответ на запрос p1ace("СевЗапМебель",St,H,G) будет False, так как не определен признак группы и не выполняется первое из двух новых правил процедуры place().

Если же в процедуру p1ace() добавить второе правило и повторить запрос, то система потребует ввести адрес и группу, к которой относится данная организация.

Goal:
place("СевЗапМебель",St,H,G)
улица Дегтярная ул.
дом 4
группа АО
True

После ввода процедура сама добавит в True динамическую БД новый факт. Убедиться в этом можно, если сформировать запрос на вывод содержимого всей БД address(), которая теперь должна содержать данные уже по пяти организациям.

Но в текущий момент вся информация ДБД хранится R оперативной памяти и выход из программы, а тем более выключение компьютера, приведет к потере всех данных. Избежать этого можно, сохранив ДБД в файле.

Задание 3

Добавьте в процедуру p1ace() одно, а затем второе правою и выполните описанные в данном разделе действия по модификации динамической БД и запросам к явной и неявным БД. Сохраните ДБД в файле "adres.dba", а программу - в файле "1ab8-2.prg". Просмотрите содержимое файла "adres.dba", а затем снова запустите программу на выполнение и выполните ряд запросов, содержание которых и результат их выполнения привести в отчете по работе.

Расширение базы данных в файлы

Рассмотренный подход к записи и считыванию ДБД, с использованием save и consult, дает хорошие результаты при небольших размерах БД.

Связано это с тем, что факты динамической БД являются частью Пролог-программы и, следовательно ограничиваются размером свободной оперативной памяти. Для увеличения объема хранимой информации можно организовать базы данных, которые хранятся не в оперативной памяти, а в файлах на диске.

Это расширяет возможность баз данных, так как при их размещении в файлах единственным ограничением является размер свободного дискового пространства. Это означает, что базы данных Турбо-Пролога могут достигать десятков мегабайт.

Предикат readterm делает возможным доступ к фактам R файле. Он может читать любой объект, записанный в файл с помощью предиката write и имеет вид:

readterm(DomainName , TermRecord).

где DomainName - это имя области типов данных, TermRecord - это терм. который связывается с объектом чтения, при условии его согласования с описанием домена.

Факты, которые описывают предикаты базы данных, могут быть обработаны так, как если бы они были термами. Это возможно благодаря домену dbasedom, который автоматически декларируется системой Турбо-Пролог и образует ОДНУ альтернативу для каждого предиката базы данных. Он описывает каждый предикат базы данный функтором и доменами аргументов данного предиката. Например, ПУСТЬ в Пролог-программе имеются следующие декларации баз данных:

```
database
person(name, age, sex)
address(firm, street, house, grup)
```

В этом случае система Турбо-Пролог автоматически сгенерирует соответствующий этим декларациям домен dbasedom:

```
domains
dbasedom = person(name, age, sex) ; address(firm. street, house, grup)
```

который может быть использован, как и любой другой предопределенный домен, Пролог- программы.

Рассмотрим простейший пример организации базы данных с использованием файла на диске. За основу примем упрощенный вариант программы 19, в котором база данных address() размещается в файле последовательного доступа "adres.dba".

Данному примеру соответствует программа 20. в которой интерфейс с явной БД обеспечивается с использованием предиката place(), структура которого аналогична программе 19. Однако процедура, его определяющая, отличается тем, что в се правилах предусмотрено обращение к фактам дисковой БД.

```
/* Программа 20 */
/* обучающаяся запросная система к БД "adres.dba" */
domains
firm, street, grup = symbol
```

```
house = integer
file = file_bd
database
address(firm, street, house, grup)
predicates
place(firm, street, house, grup)
my_read(dbasedom )
my_append( dbasedom )
next_rec( file )
clauses
my_read(Record) :- openread( file_bd ,"adres.dba" ),
readdevice( file_bd ),
next_rec( file_bd ),
readterm( dbasedom , Record).
my_append(Record) :- openappend( file_bd, "adres.dba" }.
writedevice( file_bd ),
write(Record,"\n"),nl,
closefile(file_bd).
next_rec(_).
next_rec(File) :- not(eof(File)), next_rec(File).
place(F,S,H,G);- bound(F).my_read(R), R=address(F.S,H.G),!.
place(F.S,H,G):- free(F), my_read(R), R=address(F,S.H.G).
place(F.S.H.G):- bound(F), free(S), free(H),
readdevice(key board),
write("введи для ",F),nl,
write("улица "), readln(S),
write("дом "), readreal(H),
write("группа ").readln(G),
closefile(file_bd),
my_append( address(F,S,H,G) ).
```

Для использования в программе файлов, следует ввести файловый домен, где определяется логическое имя файла. В нашей программе таким логическим имеем будет file_bd. Это имя предикатами программы связывается с именем файла БД.

В программе 20 определены два предиката для чтения и добавления домена БЛ. Это предикаты my_read и my_append. Предикат next_rec по структуре похож ни предикат repeat и служит для перехода от одной записи файла к другой, пока не будет обнаружен конец файла.

В процедуре place два первых правила в качестве второй подцели вызывают предикат my_read(R), который связывает терм R с текущей записью файла БД.

Третья подцель этих двух правил унифицирует текущую запись со структурой address(F,S,H,G). Если унификация невозможна, происходит откат к предикату my_read(R), который связывает переменную R со

следующей записью файла. И так будет продолжаться до тех пор, пока унификация не будет успешной, или пока не будет достигнут конец файла.

Если унификация завершится успешно, то переменные F, S, H, G будут иметь те же значения, что и поля текущей записи БД, а вес правило, как и процедура в целом, будет успешно доказанным. Если унификация, как первого, так и второю правила завершилась неудачей при достижении конца файла, то система перейдет к доказательству третьего правила процедуры place().

В процедуре my read() первая подцель связывает логическое имя file_db с дисковым DOS-файлом "adres.dba" и открывает его для чтения.

Вторая подцель назначает файл с логическим именем file_db стандартным устройством ввода. Это означает, что все встречающиеся далее предикаты типа read будут выполнять ввод данных из файла, а не с клавиатуры, которая по умолчанию являлась стандартным устройством ввода

Третья подцель служит для организации повторного выполнения, следующих подцелей при откатах. Следует отметить, что предикат next_rec() истинен всегда, если не обнаружен конец (eof()) записей в дисковом файле.

Так как стандартным устройством ввода сейчас установлен дисковый файл, то именно из него четвертая подцель считывает в переменную Record терм, структура которого соответствует структуре домена базы данных. Иначе говоря, выполняется считывание в переменную Record содержимого текущей записи файла, структура которой должна соответствовать описанной в программе структуре БД.

В процедуре my append() первая подцель связывает логическое имя file_db с дисковым DOS-файлом "adres.dba" и открывает его для добавления.

Вторая подцель назначает файл с логическим именем file_db стандартным устройством вывода. Это означает, что все встречающиеся далее предикаты типа write будут выполнять вывод данных в файл, а не на экран дисплея, который является стандартным устройством вывода по умолчанию.

Третья подцель выводит (добавляет в конец файла) новую запись, значение которой передано в данную процедуру через переменную Record. Кроме этого в конце записи дополнительно -выводится символ перевода на новую строку "\n".

После выполнения этих действий четвертая подцель закрывает открытый для добавления файл.

Основным режимом данной программы является режим ответов на запросы, который требует постоянного доступа по чтению s. дисковому файлу. При этом устройство ввода переназначено с клавиатуры на файл БД. Однако при отсутствии в БД нужных данных, программа переходит в режим обучения и требует от пользователя задания новых данных.

В третьем правиле процедуры place() для того, чтобы обеспечить возможность ввода данных с клавиатуры, используется предикат readdevice(keyboard) который устанавливает в качестве стандартного устройства ввода клавиатуру. После этого обеспечивается запрос на ввод

новых данных. Закрывается рабочий файл, который был открыт для чтения. И выполняется добавление новой записи в дисковый файл.

Следует отметить, что в этом примере рассмотрен простейший случай доступа к файловым данным. Он приведен с целью иллюстрации принципа построения динамических БД и их расширения в файловые структуры.

Задание 4

Загрузите программу 20. разберитесь в ней и выполните ряд описанных в предыдущем разделе действий по работе с БД, сохранению в файле "adres.dba". Познакомьтесь по приложению 1 с предикатами для работы с файлами. Доработайте программу 20 так, чтобы в ней можно было пользоваться неявными базами данных school() и bank(), аналогично программе 19. Запустите программу и введите ряд запросов, содержание которых и результат выполнения включить в отчет по работе.

Выше использовались файлы последовательного доступа. Сложность работы с ними возникает уже при выполнении таких простых действий, как удаление данных. Эта операция потребует реорганизации исходное файла БД, что вызовет непроизводительные затраты времени. Поэтому, при построении файловых БД используются файлы прямого доступа с индексными файлами. Вместо индексных файлов, в ряде случаев, более целесообразно применять метод двоичного поиска. Эти методы значительно укоряют доступ к фактам БД.

6. Содержание отлегла по лабораторной работе

Отчет по лабораторной работе должен содержать:

Программы работы с динамическими БД, использованные при выполнении данной работы, а также описание структуры этих программ.

Содержимое файловых БД, сохраненных на диске.

Результаты выполнения заданий 1,2,3, и 4.

3.3 Лабораторная работа 9

Тема:Работа со сложноструктурированными базами данных

Цель работы:

Знакомство с организацией сложноструктурированных БД в Турбо-Прологе.

Закрепление знаний по описанию сложных структур данных.

Получение навыков составления Пролог-программ для работы с БД.

Разработка собственной информационно-запросной системы на Прологе.

1. Задание на лабораторную работу

Требуется разработать информационно-запросную систему для некоторой предметной области. В качестве примера можно использовать приведенное ниже описание логической модели данных предметной области "семья". Для решения поставленной задачи необходимо:

Сформировать и описать в программе все необходимые структуры данных.

Описать структуру динамической БД.

Описать и определить предикаты неявных БД.

Составить описания и включить в программу процедуры, реализующие ряд описанных ниже действий над базами данных.
Разработать внешний (экранный) интерфейс по работе с программой. При выполнении задания может быть рассмотрена любая другая предметная область, соответствующая по сложности структур, набору процедур и действий над БД описанному ниже примеру.

Описание логической модели данных

Данная лабораторная работа развивает навыки представления структурных объектов данных и управления ими. Она показывает, что Пролог является естественным языком запросов к базе данных и как с его помощью переходить к формированию простейших баз знаний.

Как УЖС отмечалось ранее, одним из способов представления БД на Пролом может быть представление БД в виде множества фактов. Например, в базе данных о семьях каждая семья может описываться одним фактом. При этом в составе любой семья можно выделить три ее компоненты: муж, жена и дети. Исходя из этого, в предметной области "семья" могут быть выделены три объекта, которые описываются отношением

Семья(Муж, Жена, Дети)

Поскольку количество детей в разных семьях может быть разным, то объект Дети целесообразно представить в виде списка, состоящего из произвольного числа элементов:

Дети = [Ребенок!, Ребенок2, . . . , РебенокN]

Каждого члена семьи в свою очередь можно описать структурой, состоящей из четырех компонент: имени, фамилии, даты рождения и вила деятельности. Т.е. представить в виде:

ЧленСемьи(Имя , Фамилия, ДатаРождения, Деятельность)

При этом дата рождения также представляется структурой, объединяющей в своем составе три компоненты: число, месяц и год рождения:

Дата(Число, Месяц, Год)

Особый интерес представляет информация о виде деятельности, которая может содержать сведения о том работает, учится или не работает какой-либо член семьи. Причем в том случае, если человек работает, следует иметь данные о месте его работы, должности и окладе. Если учится, то следует указать место учебы. Если не работает, то надо иметь об этом информацию. Таким образом, вид деятельности для каждого из членов семьи может быть описан одной из возможных структур:

Работает(Где, Кем, Оклад) или УчитсяГде) или НеРаботает

Информация о семье, является экземпляром отношения Семья и может быть занесена в базу данных в виде факта (утверждения):

семья (членсемьи(Иван. Ким, дата(8,март, 1949), работает(АО,шеф,500)).
членсемьи(Лена, Ким, дата(3,май, 1951), неработает),
[членсемьи(Пат,Фокс, дата(5,май,1973), неработает),
членсемьи(Саша, Ким, дата(1,июнь,1983), учится(школа))]).

Тогда БД будет состоять из последовательности фактов, подобных этому, и описывать все семьи, представляющие интерес для нашей программы.

Пролог очень удобен для извлечения необходимой информации из такой базы данных. В нем хорошо то, что можно ссылаться на объекты, не указывая в деталях всех их компонент. Можно задавать только структуру интересующих нас объектов и оставлять конкретные компоненты без точного описания или лишь с частичным описанием. Так, в запросах к БД можно ссылаться на всех Ивановых с помощью терма

```
семья(членсемьи(_, Иванов, _, _), _, _)
```

Символы подчеркивания означают различные анонимные переменные, значения которых нас не заботят. Ссылаться на все семьи с тремя детьми позволяет терм:

```
семья(_, _, [_,_, _])
```

Чтобы найти всех замужних женщин, имеющих по крайней мере троих детей, можно задать вопрос:

```
семья(__, членсемьи(Имя,Фамилия,_,_),[_,_,_ |_]).
```

Основным моментом в этих примерах является то, что указывать интересующие нас объекты можно не только по их содержимому, но и по их структуре.

Получение структурированной информации из базы данных

Можно создать набор процедур, который делал бы взаимодействие с нашей БД более удобным. Такие процедуры являлись бы частью пользовательского интерфейса. Вот некоторые полезные процедуры для рассматриваемой БД:

```
муж(X):- семья(X, _, _).
жена(X):- семья (_, X, _).
ребенок(X):- семья(_,_, Дети),принадлежите X, Дети).   /* принадлежит элемент списку */'
существует (Членсемьи):- муж(Членсемьи);       /* Любой член семьи в БД */
        жена(Членсемьи);
        ребенок (Членсемьи).
датарождения(Членсемьи(_,_Дата,_)Дата).
доход(Членсемъи(_,_,_,работает(_,S)),S):-!.          /* Доход работающего */
доход(Членсемьи(_,_,_,_), 0).                        /* Доход неработающего */
принадлежите (X, [X | L].
принадлежит (X, [Y | L]:- принадлежит(X, L).
```

Этими процедурами можно воспользоваться, например в следующих запросах в базе данных:

```
Запрос
Цель
Найти имена и фамилии всех людей из базы данных
существует(членсемьи(Имя, Фамилия, _, _)).
Найти всех детей, родившихся в 1983 году
ребенок(X), датарождения(X, дата(_,_, 1983)).
Найти всех работающих жен.
```

жена(членсемьи(Имя, Фамилия,_, работает(_,_))).
Найти имена и фамилии людей, которые не работают и родились до 1963 года
существует(членсемьи(Имя, Фамилия,дата(_,_,Год) неработает)).Год < 1963.
Найти людей, родившихся до1950 года, чей доход меньше, чем 1000
существует(Членсемьи), датрождения(Членсемьи,дата(_,_,Год)). Год < 1950.доход(Членсемьи. Доход).Доход <1000.
Найти фамилии людей, имеющих по крайней мере трех детей
семья(членсемьи(_, Фамилия,_,_), _,[_,_,_|_]).

Для подсчета общею дохода семьи полезно определить сумму доходов людей из некоторого списка в виде двухаргументного отношения:
общий(Список_Людей, Сумма_их_доходов)
для которого можно написать процедуру вида:

```
общий([], 0).                    /* Пустой список людей +/
общий([Человек 1 Список], Сумма):-
доход(Человек, S),               /* S - доход 1-ого человека */
общий(Список, Sn),               /* Sn - сумма доходов оста:1ьных */
Сумма is S + Sn.
```

Теперь общие доходы всех семей могут быть найдены с помощью запроса:
семья(Муж, Жена, Дети), общий([Муж, Жена, Дети], Доход).

Пусть отношение длина подсчитывает количество элементов списка, как это было в работе, посвященной спискам. Тогда мы можем найти все семьи, которые имеют доход на члена семьи, меньший, чем 2000, при помощи запроса:
семья(Муж, Жена, Дети), общий([Муж, Жена | Дети], Доход), длина([Муж, Жена |Дети], N), Доход/N < 2000.

Задание 1

Напишите запросы для поиска в БД: a) семей без детей: b) всех работающих детей; c) семей, где жена работает, а муж нет; d) всех детей, разница в возрасте родителей которых составляет не менее 15 лет. Запросы и результат их выполнения привести в отчете по работе.

Абстракция данных и построение баз знаний

Абстракцию данных можно рассматривать как процесс организации различных фрагментов информации в единые логические единицы. При этом каждой такой логической единице придается некоторая концептуально осмысленная форма. Любая информационная единица должна быть легко доступна в программе. В идеальном случае все детали реализации исходной структуры должна быть невидимы пользователю. И самое главное - дать ему возможность использовать информацию не думая о деталях ее действительного представления.

Одним из способов реализации этого принципа является применение неявных баз данных, а также привлечение дополнительных знаний о правилах построения этих баз на основе структуры исходных данных.

В рассматриваемом примере каждая семья - это набор некоторых фрагментов информации. Все они объединены в естественные

информационные единицы такие, как член семьи или семья, и с ними можно обращаться как с едиными объектами.

На базе естественных информационных элементов могут быть определены новые отношения (неявные БД), с помощью которых пользователь может получать доступ к конкретным компонентам семьи, не зная деталей, а также получать информацию, которая в явном виде отсутствует в исходной БД.

Для исходной БД такими новыми отношениями (неявными БД) могут являться:

Отец(ИмяРебенка,ИмяОтца), МатыИмяРебенка, ИмяМатсри), Сестра-Брат(Имя1, Имя2).

Используя эти отношения, а также знания о родственных связях между людьми в обществе, можно сформировать базу знаний о родственниках, которая, например может содержать следующий набор отношений:

Родитель(ИмяРебенка, ИмяРодителя), Бабушка(ИмяВнука, ИмяБабушки). Дедушка(ИмяВнука, ИмяДедушки),

...

Предок(Кто, Чей).

Задание 2

Добавьте в исходную программу описание новых отношений и сформируйте процедуры для их определения. В исходную БД добавьте ряд фактов для семей, чтобы введенные родственные связи существовали, и выполните ряд произвольных запросов содержание которых и результат выполнения привести в отчете по работе.

Содержание отчета по лабораторной работе

Отчет по лабораторной работе должен соответствовать требованиям основного задания на работу и включать результаты выполнения заданий 1 и

Тесты для контроля усвоения знаний

1 Каков будет результат выполнения программы:

```
mush([],0).
mush([A|B],N):-mush(B,M), N is M+1.
?-mush ([саша,игорь,лена],X).
```

- X=3;
- X=лена;
- X=саша;
- X=игорь.

2 Каков будет результат выполнения программы:

```
prin (X,[X|Y]).
prin (X,[A|Y]):-prin(X,Y).
?prin (4,[1,3,4,9]).
```

- Yes;
- No;
- True;
- False.

3 Каков будет результат выполнения программы:

```
pris ([],P,P).
pris([X|Y],P,[X|T]):-pris (Y,P,T).
? pris(L,[джим..R],[джек,бил,джим,тим,джим,боб]).
```

- L=[джек,бил].
  R=[тим,джим,боб].L=[джек,бил,джим,тим].R=[боб]
- R=[джек,бил].
  L=[тим,джим,боб].R=[джек,бил,джим,тим].L=[боб]
- L=[джек,тим].
  R=[джек,тим,джим,боб].L=[джек,тим].R=[джим]
- L=[джек]. R=[боб].L=[джек,бил,джим,тим].R=[джек,боб]

4 Каков будет результат выполнения программы:

```
max([X],X).
max([X,Y],X):-max(Y,W),X>W,!.
max([X,Y],W):-max(Y,W).
?max([1,7,6,4,3],M)
```

- M=1;
- M=7;
- M=3;
- M=6.

5 Каков будет результат выполнения программы:

```
Clauses
Man(“Агамемнон”). Man(“Аид”).
Man(“Атлант”).Man(“Гелиос”).
Woman(“Автоноя”).Woman(“Агава”).Woman(“Антигона”).Woman(“
Афродита”).Woman(“Галатея”).
```

Parent(“Агамемнон”,“Аид”).Parent(“Автоноя”,“Аид”).Parent(“Гелиос”,“Атлант”).Parent(“Галатея”,“Атлант”).Parent(“Атлант”,“Афродита”).Parent(“Антигона”,“Афродита”).
Mother(X,Y):-Parent(X,Y),Woman(X).
Father(X,Y):-Parent(X,Y),Man(X).
Daughter(X,Y):-Parent(X,Y),Woman(Y).
Sun(X,Y):-Parent(X,Y),Man(Y).
Sun(X,Y):-Parent(X,Y),Man(Y).Predok(X,Y):-parent(X,Y).
Predok(X,Y):-parent(Z,Y),Predok(X,Z).
?Father(“Гелиос”,”Аид”).

- Yes
- No
- Гелиос
- Аид

6 Каков будет результат выполнения программы:

Clauses
Man(“Агамемнон”). Man(“Аид”).
Man(“Атлант”).Man(“Гелиос”).
Woman(“Автоноя”).Woman(“Агава”).Woman(“Антигона”).Woman(“Афродита”).Woman(“Галатея”).
Parent(“Агамемнон”,“Аид”).Parent(“Автоноя”,“Аид”).Parent(“Гелиос”,“Атлант”).Parent(“Галатея”,“Атлант”).Parent(“Атлант”,“Афродита”).Parent(“Антигона”,“Афродита”).
Mother(X,Y):-Parent(X,Y),Woman(X).
Father(X,Y):-Parent(X,Y),Man(X).
Daughter(X,Y):-Parent(X,Y),Woman(Y).
Sun(X,Y):-Parent(X,Y),Man(Y).
Sun(X,Y):-Parent(X,Y),Man(Y).Predok(X,Y):-parent(X,Y).
Predok(X,Y):-parent(Z,Y),Predok(X,Z).
?Mother(X,” Афродита“)

- X=Антигона
- X=Атлант
- X=Гелиос
- X=Галатея

7 Являются переменными в Прологе

- _4711
- x_123;
- 123;
- Результат;

8 Правило, определяющее отношение `ребенок` через отношение `отец`, запишется следующим образом

- ребенок(X, Y) :- отец(Y, X)
- ребенок(X, Y) :- отец(X, Y)
- отец(Y, X) :- ребенок(Y, X)

- отец(Y, X) :- ребенок(X, Y)

9 Из перечисленного: 1) простота; 2) исходный текст программ менее подвержен влиянию машинно-зависимых особенностей, чем исходные тексты программ, написанных на других языках; 3) нетребовательность к ресурсам; 4) наличие развитой среды программирования; 5) тенденция к единообразию в различных версиях языка – достоинствами языка Пролог являются

- 1, 2, 5
- 1, 3, 5
- 3, 4, 5
- 1,3

10 В Прологе встроенные предикаты не могут

- являться головой рекурсивного правила
- появляться в рекурсивном правиле
- появляться врекурсивном правиле и являться головой рекурсивного правила
- являться головой правила и появляться в факте

11 В Прологе отношение "X равно Y" описывается с помощью предиката

- X = Y
- X =:= Y
- X == Y
- X =\= Y

12 Если программа на Прологе содержит набор фактов и правил, то ее называют

- процедурой
- базой знаний
- запросом
- базой данных

13 Правило в Прологе состоит из

- предиката и тела
- предиката и списка аргументов
- имени и списка аргументов
- тела и имени

14 Из перечисленного: 1) атомы; 2) числа; 3) переменные; 4) списки; 5) выражения – в Прологе существуют виды термов

- 1, 2, 3
- 1, 2
- 3, 4, 5
- 1, 4, 5

15 Если программа на Прологе содержит набор фактов и правил, то ее называют

- базой знаний
- процедурой
- базой данных
- запросом

16 В Прологе переменные, которым не было присвоено значение, называются

- анонимными
- свободными
- формальными
- пустыми

17 Из перечисленных правил: 1) имя факта начинается со строчной буквы; 2) запись каждого факта заканчивается точкой; 3) факт должен содержать хотя бы одну переменную; 4) факт должен ссылаться на правило – в Прологе при записи фактов необходимо выполнять

- 3, 4
- 2, 3
- 1, 4
- 1, 2

18 Из перечисленного: 1) <---->; 2) x_123; 3) 123; 4) 'Это атом или нет?'; 5) Результат; 6) лето – являются корректными атомами в Прологе

- 2, 3, 5, 6
- 1, 2, 4, 6
- 2, 4, 5, 6
- 1, 3, 4, 5

19 Из перечисленного: 1) _4711; 2) x_123; 3) 123; 4) _; 5) Результат; 6) лето – являются переменными в Прологе

- 2, 5, 6
- 1, 4, 5
- 1, 3, 4
- 2, 4, 6
- 3.

20 В Прологе голова и тело правила разделены знаком

- ->
- :
- |
- :-

21 Предикат Пролога записывается в виде

- имя_предиката=аргументы
- имя_предиката(аргументы)
- имя_предиката аргументы
- имя_предиката:-аргументы

22 Механизм решения задачи при помощи языка Пролог называется
- обработчиком знаний
- интерпретатором
- генератором запросов
- компилятором

23 Из перечисленного: 1) _4711; 2) x_123; 3) 123; 4) _; 5) Результат; 6) лето – являются переменными в Прологе
- 1, 4, 5
- 2, 4, 6
- 1, 3, 4
- 2, 5, 6

24 В Прологе отношение "X равно Y" описывается с помощью предиката
- X = Y
- X =\= Y
- X == Y
- X =:= Y

25 Переменная в Прологе рассматривается как
- отдельный объект
- выделенный участок памяти
- глобальное имя для некоторого объекта
- локальное имя для некоторого объекта

26 Теоретической основой Пролога является
- лямбда-исчисление Черча
- исчисление высказываний
- фреймовое представление
- исчисление предикатов

27 В языке Пролог факт -это
- Неопровержимое доказательство
- Истинное происшествие
- Предикат с аргументами-констатнтами
- Правило, которое выполняется всегда

28 В языке Пролог правило - это
- хорновские фразы с заголовком и одной или несколькими подцелями
- предикаты, носящие приказывающий характер
- факты, в которых содержится условие
- алгоритм действия

29 Вопрос - это
- отправная точка логического ввода, происходящего при выполнении программы
- отправная точка логического вывода, происходящего при выполнении программы
- отправная точка логического вывода свободных переменных

– запрос программы на соплоставление переменных.

30 Имя – это

– последовательность букв и цифр, начиающаяся со строчной буквы
– последовательность букв и цифр, начиающаяся с прописной буквы
– конструкция, состоящая из имени и заключенного в круглые скобки списка его аргументов, разделенных запятыми
– объединение элементов произвольных видов, разделенных запятыми и заключенных в квадратные скобки

31 Переменная - это

– последовательность букв и цифр, начинающаяся со строчной буквы
– последовательность букв и цифр, начинающаяся с прописной буквы
– конструкция, состоящая из имени и заключенного в круглые скобки списка его аргументов, разделенных запятыми
– объединение элементов произвольных видов, разделенных запятыми и заключенных в квадратные скобки

32 Вопрос называется общим, если:

– все переменные, которые он содержит,- свободные;
– хотя бы одна переменная, которую он содержит, - свободная
– он не содержит переменных
– все переменные, которые он содержит, - связанные

33 Вопрос называется частным, если

– все переменные, которые он содержит,- свободные;
– хотя бы одна переменная, которую он содержит, - свободная
– он содержит переменные
– все переменные, которые он содержит, - связанные

34 Программа на Прологе является

– алгоритмом действия операторов на переменные;
– записью условия задачи на языке формальной логики
– процедурным описанием алгоритма
– функциональным описанием алгоритма

35 Чем в языке Пролог заканчивается строка программы?

– :
– :-
– ;
– .

36 Какая операция в языке Пролог является основной?

– присваивание;
– сопоставление;
– отсекание;

- редуцирование

37 Как в языке Пролог выглядит запрос:

- ?
- /Help;
- Zapros;
- say

38 Что в языке Пролог будет являться именем:

- Name;
- name;
- $name;
- #name.

39 Что в языке Пролог будет являться переменной:

- Name;
- name;
- $name;
- #name.

40 Переменная, используемая в качестве аргумента предиката,когда конкретное значение переменной несущественно,-это переменная

- свободная;
- связанная;
- анонимная;
- декларативная.

41 Переменная, которая еще не получила конкретного значения в результате сопоставления с константами в фактах,- это

- свободная;
- связанная;
- анонимная;
- декларативная.

42 Переменная, которая приняла конкретное значение, называется:

- свободная;
- связанная;
- анонимная;
- декларативная.

43 Переменные служат:

- хранилищем информации;
- частью процесса сопоставления;
- отправной точкой логического вывода;
- заменой констант.

44 Правило, определяющее отношение ребенок/2 через отношение отец/2, запишется следующим образом

- отец(Y, X) :- ребенок(X, Y)
- ребенок(X, Y) :- отец(X, Y)
- отец(Y, X) :- ребенок(Y, X)

- ребенок(X, Y) :- отец(Y, X)

45 В Прологе в результате унификации переменной с атомом переменная
- принимает значение данного атома
- сохраняет свое значение
- уничтожается
- становится анонимной

46 В Прологе выражение X is ([1,2,3] +5) имеет значение
- 8
- 5
- 11
- 6

47 В Прологе при выполнении запроса первым выводится то из решений, которое
- стоит первым после сортировки полученных решений
- находится ближе к началу базы данных
- является наиболее правдоподобным
- находится ближе к концу базы данных

48 Хвостом списка [1, 2, 3] в Прологе является
- [2, 3]
- [1, 2, 3]
- [3]
- пустой список

49 Хвостом списка [1, 2, 3] в Прологе является
- пустой список
- [3]
- [2, 3]
- [1, 2, 3]

50 В Прологе встроенные предикаты не могут
- появляться рекурсивном правиле и являться головой рекурсивного правила
- появляться в рекурсивном правиле
- являться головой правила и появляться в факте

являться головой рекурсивного правила

51 Рекурсивная процедура в Прологе должна включать компоненты
- нерекурсивную фразу и рекурсивное правило
- оператор цикла и счетчик уровней рекурсии
- рекурсивное правило и счетчик уровней рекурсии
- нерекурсивную фразу и оператор цикла

52 Предикат вычисления факториала натурального числа n выглядит так:
- factorial(1,1). factorial(N,X):-factorial(N-1,Y),X is Y*N;
- factorial(1,1). factorial(N,X):-factorial(N,Y),X is Y*N;
- factorial(0,1). factorial(N,X):-factorial(N,X),X is N*(N-1);
- factorial(0,1). factorial(N,Y):-factorial(N-1,X),X is Y*(N-1);

53 В Прологе составной терм унифицируется с другим составным термом, если совпадают

- имена
- количество аргументов
- имена и количество аргументов, а аргументы поддаются унификации

имена и количество аргументов

54 Структура -это

- объединение элементов произвольных видов, разделенных запятыми и заключенных в квадратные скобки;
- конструкция, состоящая из имени структуры и ее свойств, разделенных запятыми;
- последовательность букв и цифр, начинающихся со строчной буквы
- конструкция, состоящая из имени структуры и заключенного в скобки списка ее аргументов, разделенные запятыми

55Список - это

- объединение элементов произвольного вида, разделенных запятыми и заключенных в квадрвтные скобки
- конструкция, состоящая из имени структуры и ее свойств, разделенных запятыми
- последовательность букв и цифр, начиающаяся со строчной буквы
- конструкция, состоящая из имени структуры и заключенного в скобки списка ее аргументов, разделенные запятыми

56 Рекурсивная процедура в Прологе должна включать компоненты

- нерекурсивную фразу и рекурсивное правило
- оператор цикла и счетчик уровней рекурсии
- нерекурсивную фразу и оператор цикла
- рекурсивное правило и счетчик уровней рекурсии

57 В языке Пролог выход из рекурсии обеспечивается:

- Halt;
- Break;
- Stop;
- ! (отсечение).

58 В языке Пролог списком будет:

- [голова|хвост]
- p^next;
- set of item;
- rray.

59 Для вычисления арифметических выражений в Прологе используется оператор

- is
- =
- is on

– ==

60 В Прологе при выполнении запроса первым выводится то из решений, которое

- является наиболее правдоподобным
- находится ближе к началу базы данных
- находится ближе к концу базы данных
- стоит первым после сортировки полученных решений

61 В Прологе элементы списка разделяют

- запятыми
- точками с запятой
- точками
- пробелами

62 Правило в Прологе состоит из

- имени и списка аргументов
- предиката и тела
- предиката и списка аргументов
- тела и имени

63 Из перечисленных правил при записи фактов необходимо выполнять:

- имя факта начинается со строчной буквы
- запись каждого факта заканчивается точкой
- факт должен содержать хотя бы одну переменную
- факт должен ссылаться на правило

Таблица 1–Ответы на тесты

№	1	2	3	4
1	X			
2	X			
3	X			
4		X		
5		X		
6	X			
7	X			X
8	X			
9	X			
10		X		
11	X			
12		X		
13	X			
14	X			
15	X			
16	X			
17				X
18			X	
19		X		
20				X
21		X		

22		X		
23	X			

Продолжение таблицы 1

№	1	2	3	4
24	X			
25			X	
26				X
27			X	
28	X			
29		X		
30	X			
31		X		
32			X	
33			X	
34		X		
35				X
36		X		
37	X			
38		X		
39	X			
40			X	
41	X			
42		X		
43		X		
44				X
45				X
46				
47		X		
48				X
49			X	
50			X	
51	X			
52	X			
53			X	
54				X
55	X			
56	X			
57				X
58	X			
59	X			
60		X		
61		X		
62		X		
63	X	X		

Список использованных источников

1. Братко И. Программирование на языке искусственного интеллекта: Пер. с англ./Братко И.-Мир, 1990.
2. Янсон А. Турбо-Пролог в сжатом изложении: пер.с нем.- М. [Текст]:Мир,1991.
3. Практикум по информатике: Учеб. Пособие для студ.высш.учеб.заведений/Могилёв, Н.И.Пак, Е.К. Хеннер; Под ред Е.К.Хённера.-М.: Издательский центр «Академия», 2002.-608с.
4. Хабаров С.П. Использование Турбо-Пролога при проектировании баз данных и баз знаний[Электронный ресурс]: учебное пособие. –Режим доступа: http://firm.trade.spb.ru/serp/book_prolog/prolog1.html

Приложение А (справочное)–Краткий список встроенных предикатов

Полный список приведен в файле prolog.hlp)

Предикаты ввода

readln(StringVariable) - читает строку.
readint(IntgVariable) - читает целое число.
readreal(RealVariable) - читает действительное число.
eadchar(CharVariable) - читает символ.
file_str(DosFileName,StringVariable) - читает строку из файла.
inkey(CharVariable) - читает ключ (символ).
keypressed - проверяет, нажата ли клавиша.

Предикаты вывода

write(Variable|Constant *) - производит запись на текущее устройство вывода.
nl - перевод строки.

Предикаты для работы с файлами

openread(SymbolicFileName,DosFileName) - открывает файл для чтения.
openwrite(SymbolicFileName,DosFileName) - открывает файл для записи.
openappend(SymbolicFileName,DosFileName) - открывает файл для добавления.
openmodify(SymbolicFileName,DosFileName) - открывает файл для чтения/записи.
readdevice(SymbolicFileName) - определяет или считывает символическое имя файла
устройства ввода.
writedevice(SymbolicFileName) - определяет или считывает символическое имя файла устройства вывода.
filemode(SymbolicFileName,FileMode) - установка или чтение типа файла.
closefile(SymbolicFileName) - закрывает файл.
filepos(SymbolicFileName,FilePosition,Mode) - установка или чтение позиции указателя.
eof(SymbolicFileName) - проверка на конец файла
flush(SymbolicFileName) - очищает содержимое буфера.
existfile(DosFileName) - проверка существования файла.
deletefile(DosFileName) - удаляет файл.
renamefile(OldDosFileName,NewDosFileName) - переименовывает файл.
disk(DosPath) - устанавливает или показывает накопитель или путь.

Предикаты экрана

scr_attr(Row,Column,Attr) - устанавливает или считывает атрибут.
field_str(Row,Column,Length,String) - записывает или читает строку.
field_attr(Row,Column,Length,Attr) - устанавливает или читает атрибут поля экрана.
cursor(Row,Column) - считывает или устанавливает позицию курсора.
cursorform(Startline,Endline), 0<Startline<14, 0<Endline<14- считывает или устанавливает форму курсора.
attribute(Attr) - считывает или устанавливает цвет фона текущего окна.

Предикаты работы с окнами

makewindow(WindowNo,ScrAtt,FrameAtt,Framestr,Row,Column,Height,Width) - создает окно.
shiftwindow(WindowNo) - меняет текущее окно или считывает номер текущего окна.
gotowindow(WindowNo)- -активизирует окно с заданным номером.
existwindow(WindowNo) - проверяет существование окна.
removewindow - удаляет текущее окно.
clearwindow - чистка окна.
window_str(ScreenString) - записывает (или считывает) строку в текущее окно
window_attr(Attribute) - определяет атрибуты текущего окна
scroll(NoOfRows,NoOfCols) - сдвиг содержимого текущего окна.

Предикаты для работы со строками

frontchar(String,FrontChar,RestString) - разделяет заданную строку на первый символ и оставшуюся часть.
fronttoken(String,Token,RestString) - разделяет строку на лексему и остаток.
frontstr(Lenght,Inpstring,StartString,RestString) - разделяет строку на две части, количество первой части равно Lenght.
concat(String1,String2,String3)- String3 = String1 + String2.
str_len(String,Length) - определяет длину строки.

Предикаты преобразования данных

char_int(CharParam,IntgParam) - преобразует символ в целое число или наоборот.
str_int(StringParam,IntgParam) - преобразует строку в целое число или наоборот.
str_char(StringParam,CharParam) - преобразует строку в символ или наоборот.
str_real(StringParam,RealParam) - преобразует строку в действительное число или наоборот.

upper_lower(StringInUpperCase,StringInLowerCase) - преобразует прописные буквы в строчные и наоборот.

Предикаты баз данных

consult(DosFileName) - загружает или добавляет текстовый файл базы данных в ОП.
consult(DosFileName,InternalDatabaseName) - загружает в ОП группу поименованную группу фактов.
save(DosFileName) - записывает на диск все факты динамической БД.
save(DosFileName,InternalDatabaseName) - записывает на диск поименованную группу фактов БД.
assert(Term) - добавляет факт БД в ОП.
retractall(Term) - удаляет все факты с указанным термом.
retractall(_, InternalDbaseName) - удаляет все факты поименованной группы.

Предикаты для работы с редактором

display(String) -) - показывает в текущем окне строку(до 64Кбайт).
edit(InputString,OutputString) - вызов редактора.
editmsg(InputString,OutputString,Headstr,Headstr2,Msg,Pos,Helpfilename,RetStatus) - вызов редактора с дополнительными возможностями.

Системные предикаты

system(DosCommandString) - выполнение команд DOS.
dir(Path,Filespec,Filename) - выводит текущий каталог.
comline(LineBuffer) - читает параметры командной строки.
port_byte(PortNo,Value) - посылает байт в порт или читает его из порта.
ptr_dword(8086Ptr,Segment,Offset) - читает строку или адрес строки.
memword(Segment,Offset,Word) - запоминает или считывает слово по заданному адресу.
membyte(Segment,Offset,Byte) - запоминает или считывает байт.
bios(Interruptno,reg(AXi,BXi,CXi,DXi,SIi,DIi,DSi,ESi), reg(AXo,BXo,CXo,DXo,SIo,DIo,DSo,ESo)) - объявляет прерывания для вызова процедур BIOS
exit - выход из программы.
storage(StackSize,HeapSize,TrailSize) - определяет размер имеющейся памяти.
sound(Duration,Frequency) - звуковой сигнал с параметрами.
beep - звуковой сигнал.
date(Year,Month,Day) - установка или считывание даты.
time(Hours,Minutes,Seconds,Hundredths) - устанавливает или считывает системное время.
findall(Variable, Atom, ListVariable) - собирает значения, возникающие в процессе бектрегинга, в список.

free(Variable) - проверяет свободная ли переменная.
bound(Variable) - проверяет связана ли переменная.

Арифметические операции

+, -, *, /, mod, div

Операции отношения

>, <, =, >=, <=, <>, ><

Логические операции

not(Atom) - отрицание.
and , or

Функции

sin, cos, tan, arctan, ln, log, exp, sqrt, round, trunc, abs

Приложение Б (обязательное)–Варианты индивидуальных заданий к лабораторной работе 1

Для углубленного изучения предлагается выполнить пример практической реализации по следующему алгоритму. Сначала необходимо познакомиться со средой Турбо-Пролог. Далее, в соответствии с выбранным вариантом задания необходимо ввести программу. Далее добавить к ней три факта относящихся к соответствующей области знаний. В отчет записать номер варианта, текст программы, с введенными дополнительными фактами, запросы и их результаты, трассы этих запросов.

Вариант 1 Музыка

```
Predicates
songster(symbol,symbol).
Clauses
songster(“sadness”, “enigma”).
songster(“mea culpa”, “enigma”).
songster(“principles of lust”, “enigma”).
songster(“white dove”, “scorpions”).
songster(“still loving you”, “scorpions”).
 songster(“nothing else matters”, “metallica”).
```

Вариант 2 Жанры кино

```
Predicates movie(symbol,symbol).
Clauses
movie(“man in black”,”comedy”).
movie(“man in black-2”,”comedy”).
movie(“King Kong”,”Adventure”).
movie(“Indiana Jones”, “ Adventure”).
movie(“The Lion King”,  “Cartoon”).
movie(“Ice Age”, “Cartoon”).
```

Вариант 3 Расширени алов

```
 predicates
extension(symbol,symbol).
clauses
extension( “picture”, “*.jpg”).
extension( “picture”, “*.tif”).
extension( “picture”, “*gif”).
extension( “video”, “*.mp4”).
```

extension(“video”, “*.mov”).
extension(“text”, “*.txt”).

Вариант 4 Авторы книг

predicates
 book(symbol,symbol).
clauses
book(“the fellowship of the ring”, “tolkien”).
book(“hobbit”, “tolkien”).
book(“hit or myth”, “asprin”).
book(“myth inc.”, “asprin”).
book(“karrie”, “king”).
book(“red rose”, “king”).

Вариант 5 Времена года

predicates
season(symbol,integer).
clauses
season(“january”, “winter”).
season(“march”, “spring”).
season(“april”, “spring”).
season(“may”, “spring”).
season(“june”, “summer”).
season(“august”, “summer”).

Вариант 6 Количество дней в месяце

predicates
days (symbol,symbol).
clauses
days (“janury”, 31).
days (“march”, 31).
days (“april”, 30).
days (“may”, 31).
days (“june”, 30).
days (“august”, 31).

Вариант 7 Типы ПО

predicates
class (symbol,symbol).
clauses
class (“Windows”, “Operation System”).

class (“DOS”, “Operation System”).
class (“Unix”, “Operation System”).
class (“Word”, “Text Editor”).
class (“Notepad”, “Text Editor”).
class (“Doom”, “Game”).

Вариант 8 Фирма – производитель ПО

predicates
manufacturer (symbol,symbol).
clauses
manufacturer (“Windows”, “Microsoft”).
manufacturer (“Office”, “Microsoft”).
manufacturer (“Photoshop”, “Adobe”).
manufacturer (“Illustrator”, “Adobe”).
manufacturer (“Flash”, “Macromedia”).

Вариант 9 География

predicates
city (symbol,symbol).
clauses
city(“Ryn”, “Russia”).
city(“Moscow”, “Russia”).
city(“Vologda”, “Russia”).
city(“New York”, “USA”).
city(“Chicgo”, “USA”).
city(“London”, “Great Britain”).

Вариант 10 Цвета

predicates
color (symbol,symbol).
clauses
color (“Corn”, “Yellow”).
color (“Banana”, “Yellow”).
color (“Cheese”, “Yellow”).
color (“Apple”, “Red”).
color (“Raspberry”, “Red”).
color (“Kiwi”, “Green”).

Вариант 11 Овощи-фрукты

predicates
eat (symbol,symbol).

clauses
eat(“apple”, “fruit”).
eat(“orange”, “fruit”).
eat(“kiwi”, “fruit”).
eat(“potatoes”, “vegetables”).
eat(“onion”, “vegetables”).
eat(“cabbage”, “vegetables”).

Вариант 12 Дисциплины

predicates
courses (symbol,symbol).
clauses
courses(“microelectronics”, “technical”).
courses(“SII”, “technical”).
courses(“Programming”, “technical”).
courses(“psychology”, “humanitarian”).
courses(“for_lang”, “humanitarian”).

Приложение В (обязательное)–Варианты индивидуальных заданий к лабораторной работе 2

Аксиоматизируйте предложенные области знаний. Формируемая вами система аксиом должна быть достаточной для ответа на вопросы, список которых вы должны составить заранее. Вопросы должны быть такими, чтобы для ответа на них требовалась цепочка шагов вывода.

Вариант 1

Если бы я был богат, то автомобиль купил бы. Если бы я был бесчестен, то я украл бы таковой.

Вариант 2

Если девушка почувствует горошину, то она – настоящая принцесса. Эта девушка – настоящая принцесса.

Вариант 3

Все Моржи – водные млекопитающие. Все ушастые тюлени - водные млекопитающие. Все настоящие тюлени –водные млекопитающие. Моржи, ушастые тюлени, настоящие тюлени представляют семейство ластоногих.

Вариант 4

Земля вращается вокруг Солнца по эллиптической орбите. Марс вращается вокруг Солнца по эллиптической орбите. Юпитер вращается вокруг Солнца по эллиптической орбите. Сатурн вращается вокруг Солнца по эллиптической орбите. Плутон вращается вокруг Солнца по эллиптической орбите. Венера вращается вокруг Солнца по эллиптической орбите. Уран вращается вокруг Солнца по эллиптической орбите. Нептун вращается вокруг Солнца по эллиптической орбите. Меркурий вращается вокруг Солнца по эллиптической орбите. Земля, Марс, Юпитер, Сатурн, Плутон, Венера, Уран, Нептун, Меркурий –планеты Солнечной системы.

Вариант 5

Если ты опоздаешь хоть на одну минуту, то карета сделается тыквой, лошади – мышами, лакеи –ящерицами, а пышный наряд –стареньким, залатанным платьицем.

Вариант 6

Всем людям для жизнедеятельности нужен кислород. Всем растениям для жизни нужна влага. Всем организмам необходимы витамины. Всем растениям для фотосинтеза нужен солнечный свет. Курение вредно для здоровья. Некоторые лекарственные растения помогают ликвидировать воспалительный процесс.

Вариант 7

Валин синтезируется на рибосомах. Глицин синтезируется на рибосомах. Лизин синтезируется на рибосомах. Глютаминовая кислота синтезируется на рибосомах. Все белки синтезируются на рибосомах. Тирозин – белок.

Вариант 8

Если наступает наводнение, то уровень воды поднимается. Если уровень воды поднимается, то это может привести к затоплению домов.

Приложение Г (обязательное)–Варианты индивидуальных заданий к лабораторным работам 3-4

Запишите на Прологе правила, являющиеся решением следующих заданий:

Вариант 1
Даны два числа a и b, получите их сумму, разность и произведение.

Вариант 2
Дана длина ребра куба, найдите объём куба и площадь его боковой поверхности.

Вариант 3
Дан радиус основания r и высота цилиндра h, найдите его объём и площадь боковой поверхности.

Вариант 4
Даны стороны a и b параллелограмма, а также угол между ними, найдите диагонали параллелограмма и его площадь.

Вариант 5
Вычислите значение выражения 2x+3y+4

Вариант 6
Вычислите значение выражения (2x+8y+4)/2

Вариант 7
Вычислите значение выражения y-x^2

Вариант 8
Вычислите значение выражения x^2+xy+ y^2

Вариант 9
Вычислите значение выражения x/2+5y

Вариант 10
Вычислите значение выражения 5(34x-y)

Вариант 11
Вычислите значение выражения x^2+3y^2

Приложение Д (обязательное)–Варианты индивидуальных заданий к лабораторной работе 5-6

Напишите программы, выполняющие операции над списками:

Вариант 1
Объедините два списка, найдите MAX и удалите его.

Вариант 2

Удалите из списка элемент, найдите длину оставшегося списка

Вариант 3

Добавьте к списку элемент, вычислите среднее арифметическое его элементов

Вариант 4

Обратите список, найдите последний и предпоследний элементы

Вариант 5

Исключите из списка заданный элемент во всех вхождениях, кроме первого, найдите длину оставшегося списка

Вариант 6
Проверьте, имеются ли в списке повторяющиеся элементы, и все их удалите

Вариант 7

Удалите из списка все элементы, равные последнему, найдите длину оставшегося списка

Вариант 8

Объединить два списка, найти MIN и удалить его

Вариант 10

Обратить список, найти MAX и удалить его

Вариант 11
К списку добавить обращенный 2-й список, найти длину результата

Вариант 12
Отсортировать список

Приложение Е (Обязательное)–Задания для самостоятельной работы

1. Опишите на Прологе свою родословную, определите бабушек, дедушек, прабабушек, прадедушек.
2. Опишите на Прологе телефонную книгу.
3. Опишите на Прологе районы вашего города, республики, области, укажите численность их населения, местные достопримечательности.
4. Опишите на Прологе европейские государства(население, площадь ит.д.)
5. Опишите на Прологе таблицу дат и событий русской истории.
6. Опишите на Прологе небольшой словарь для перевода с русского языка на иностранный язык, который вы изучаете.
7. Опишите на Прологе успеваемость вашей группы, дайте определение отличника.
8. Опишите на Прологе каталог книг в библиотеке.
9. Напишите программу, решающую задачу о волке, козе и капусте.
10. Напишите программу, определяющую как разлить 10 л молока по 5 л, пользуясь бидонами на 3, 7 и 10 л.
11. Определите предикат сестра и найдите всех сестёр конкретного лица.
12. Определите предикат брат и найдите всех братьев конкретного лица.
13. Определите предикат сын и найдите всех сыновей конкретного лица.
14. Определите предикат дочь и найдите всех дочерей конкретного лица.