Лабораторная работа №1

Основные сведении о языке Пролог и системе программирован. Visual Prolog

Цель работы. Изучение основных возможностей языка Пролог и системы программирования Visual Prolog. Изучение механизмов управления в программах на языке Пролог.

Краткие теоретические сведения.

1.1. Начало работы с системой Visual Prolog

Программы, разрабатываемые в системе Visual Prolog (VIP), оформляются как проекты. Они могут строиться на различных платформах (например, MS DOS или Windows). От этого зависит уровень возможностей программы. В данной работе рассматривается создание программы на платформе MS-DOS.

Начальные действия по созданию новой программы в среде VIP выполняются в следующем порядке.

1. Запустить систему VIP.
2. Из меню Project выбрать команду New Project.
3. В появившемся на экране окне Application Expert выбрать закладку

General (обычно она оказывается выбранной по умолчанию). В поле Project Name указать имя проекта (например, LAB1). В поле Name of .VPR file достаточно просто щелкнуть мышью; будет указано имя файла проекта, совпадающее с заданным именем проекта (в данном примере – LAB1.VPR). В поле Base Directory указывается имя папки, в которой должны сохраняться создаваемые файлы программы. Например, если их требуется сохранять на диске E: в папке ES, то следует указать E:\ES\

4. Перейти на закладку Target. В поле Platform выбрать DOS, в поле UI

Strategy – TextMode, в поле Target Type – exe, в поле Main Program – Prolog.

Нажать кнопку Create.

5. На экран выводится окно, в котором отображается структура проекта.

Для начала набора текста программы следует выделить имя программы (файл с расширением .PRO) и нажать кнопку Edit, или просто дважды щелкнуть мышью по имени программы.

Если проект уже имеется на диске, то для его загрузки следует воспользоваться командой Open Project из меню Project.

1.2. Подготовка и запуск программы в системе Visual Prolog

После выполнения действий, описанных выше, на экране появляется “шаблон” программы. Он содержит разделы predicates, clauses и goal, имеющиеся в любой программе на языке Пролог. Назначение этих разделов будет рассмотрено ниже. Кроме того, на экран выводятся комментарии; из них можно видеть, что в Прологе начало комментария обозначается символами /*, а конец - */. Выводятся еще некоторые команды; для рассматриваемого примера их можно удалить.

В качестве примера рассмотрим подготовку и запуск программы, запрашивающей у пользователя два числа и выводящей на экран результат их умножения. Текст программы приведен ниже.

```
predicates
nondeterm obrab
nondeterm umnoz (real, real, real)
clauses
obrab:- write ("Введите 1-е число: "), readreal (X1),
write ("Введите 2-е число: "), readreal (X2),
umnoz (X1, X2, Y), write ("Результат: ",Y).
umnoz (A, B, C):- C=A*B.
goal
obrab.
```

Смысл конструкций, использованных в программе, будет рассмотрен ниже. Однако уже из этого примера видно, что для вывода данных на экран в Прологе используется команда write, а для ввода *вещественных* чисел – readreal. Здесь же можно видеть, что арифметические операции (например, умножение) записываются в Прологе так же, как в других языках.

Для сохранения программы следует использовать команду File – Save, или нажать клавишу F2.

Для запуска программы на выполнение следует из меню Project выбрать команду Run. На экран сначала выводится окно сообщений (Messages), где выводится информация о ходе компиляции программы.

Затем на экран выводится окно MS-DOS. В нем сначала выводится информация о версии используемой системы VIP и сообщение Press any key. Для продолжения работы необходимо нажать любую клавишу. Затем выполняется запрос исходных данных и вывод результатов. После этого окно MS-DOS следует закрыть. Происходит возврат в окно сообщений. Чтобы снова перейти к программе, следует из меню Window выбрать окно файла программы (с расширением .PRO) или файла проекта (.VPR).

В случае ошибок в программе на экран выводится окно сообщений (Message) с текстом сообщений об ошибках. Чтобы перейти к тому месту в программе, где была обнаружена ошибка, следует щелкнуть мышью по сообщению об ошибке.

1.3. Основные конструкции языка Пролог. Понятие предиката

Пролог - язык логического программирования, предназначенный для представления и обработки знаний о некоторой предметной области. В Прологе реализован так называемый декларативный подход, при котором задача описывается с помощью набора утверждений о некоторых объектах и правил обработки этих утверждений.

Основной конструкцией языка Пролог является *предикат*. Предикат – это логическая функция от некоторого набора аргументов; при этом аргументы могут иметь любой вид, а функция принимает значение "ложь" или "истина". В языке Пролог имеется три вида предикатов: предикаты-факты, предикаты-правила и стандартные предикаты.

Предикаты-факты. Аргументами предикатов-фактов являются константы. Предикаты-факты предназначены для записи некоторых утверждений, которые при выполнении программы считаются истинными. Пусть, например, требуется записать в программе утверждение: "Иван – отец Петра". В программе на Прологе его можно выразить фактом:

father ("Иван","Петр"). Точка в конце факта обязательна. Здесь father – имя предиката (оно может быть любым другим). Этот предикат имеет два аргумента, оба - строковые.

Приведем еще несколько примеров фактов. Пусть требуется записать в программе на Прологе следующую информацию: “Сотрудник Иванов работает над проектом с шифром П20. Этот проект разрабатывается по заказу завода “Рубин”, и стоимость контракта на его разработку составляет 700 тыс ден. ед.”. В программе на Прологе это можно записать в виде двух предикатов-фактов:

rabota (“Иванов”, “П20”) и proekt (“П20”, “Рубин”, 700).

Предикат rabota имеет два аргумента (оба – строковые), предикат proekt – три аргумента (два строковых и один целочисленный).

Набор фактов, имеющихся в программе на Прологе, называется базой данных.

Предикаты-правила. Они записываются в виде: <Предикат1> истинен, если истинны <Предикат2>, <Предикат3>,...,<ПредикатN>. Здесь <Предикат1>
называется головным (или заголовком), остальные - телом правила.

Приведем пример. Пусть в программе есть факты, задающие отношение "отец" (см. пример выше), а требуется составить правило, позволяющее найти по этим данным брата конкретного человека (или проверить, являются ли два конкретных человека братьями). Это правило можно сформулировать так:

"Один человек - брат другого, если эти люди разные и у них один и тот же отец". На Прологе это можно записать так:

brother (X, Y):- father (Z, X), father (Z, Y), X<>Y.

Здесь предикат brother - заголовок правила, остальные три предиката составляют его тело. Символы :- обозначают "если", запятая - союз "И" (отметим также, что точка с запятой обозначает союз "ИЛИ").

Приведем еще один пример. Пусть в программе имеются факты rabota, описывающие работу сотрудников над проектами, и факты proekt, описывающие заказчиков проектов (см. пример выше). Требуется составить правило, позволяющее найти заказчиков, для которых работает конкретный сотрудник. Такое правило можно записать следующим образом

rab_zak (Fam, Zak):- rabota (Fam, Shifr), proekt (Shifr, Zak, Stoim).

Одно правило может иметь несколько вариантов (альтернатив).

Рассмотрим пример такого правила. Пусть требуется вычислить функцию

Y=2*X, если X<10, и Y=X/2, если X≥10.

На Прологе правило для такого расчета можно записать так:

funct (X, Y):- X<10, Y=2*X.

funct (X, Y):- X >= 10, Y=X/2.

Можно записать то же самое короче:

funct(X, Y):- X<10, Y=2*X, !.

funct(X, Y):- Y=X/2.

Смысл символа "!" будет показан позже.

Правила и факты называют также клозами (предложениями). Предикат

brother (см. выше) состоит из одного клоза, предикат funct - из двух. Если предикат состоит из нескольких клозов, то они должны быть расположены вместе (между ними не должно быть клозов других предикатов).

Стандартные предикаты входят в состав самого языка Пролог. Простейшие из них: write (X) - вывод значения X на экран (по умолчанию) или на другое устройство, где X - переменная или константа; nl - переход в следующую строку на экране; readln (X) - ввод строковой переменной;

readint (X) - ввод целочисленной переменной; readreal (X) - ввод вещественной переменной.

1.4. Пример программы на Прологе. Структура программы

Пример. Имеются факты rabota, описывающие работу сотрудников над проектами, и факты proekt, описывающие заказчиков проектов (см. пример выше). Требуется составить программу, которая будет выводить шифры и заказчиков проектов, над которыми работает указанный сотрудник, если стоимость этих проектов превышает некоторую величину. Фамилия сотрудника минимальная стоимость проекта запрашиваются у пользователя.

```
predicates
nondeterm vyvod
nondeterm poisk (string, integer)
nondeterm rabota (string, string)
nondeterm proekt (string, string, integer)
goal
vyvod.
clauses
vyvod:- write ("Фамилия: "), readln (F),
write ("Стоимость: "), readint (S),
poisk (F, S).
poisk (Fam, St):- rabota (Fam, Shifr),
proekt (Shifr, Zak, Stoim),
Stoim >= St,
write (Shifr," ",Zak).
rabota ("Антонов", "П20").
rabota ("Иванов", "П70").
rabota ("Иванов", "П100").
rabota ("Петров", "П100").
rabota ("Васильев", "П70").
rabota ("Иванов", "П120").
```

rabota ("Васильев", "П120").
proekt ("П20", "Рубин", 700).
proekt ("П100", "Рубин", 1400).
proekt ("П70", "Горизонт", 500).
proekt ("П120", "Кристалл", 1100).

Здесь показаны три основных раздела программы на Прологе:
• predicates - список имен предикатов и типы их аргументов;
• goal – целевой предикат (цель);
• clauses - перечень клозов, т.е. правил и фактов.

Еще раз следует обратить внимание, что имена переменных в программах на Прологе должны начинаться с *заглавной* буквы.

1.5. Принцип работы программ на Прологе

Выполнение программы на Прологе заключается в доказательстве целевого предиката. Этот предикат обычно является правилом. Чтобы доказать правило (любое, а не только цель), требуется доказать все предикаты, составляющие его тело, т.е. найти факты, соответствующие этим предикатам. Для этого происходит согласование предиката с другим одноименным предикатом, т.е. сопоставление (*унификация*) соответствующих аргументов этих предикатов. Согласование предикатов выполняется успешно, если успешна унификация всех аргументов предикатов. При унификации аргументов возможны следующие основные случаи:
• сопоставление двух констант - заканчивается успешно, если они равны, и неудачно, если они не равны;
• сопоставление константы и переменной, еще не имеющей значения (свободной) - заканчивается успешно, и переменная получает значение константы (становится связанной);
• сопоставление двух связанных (т.е. имеющих значения) переменных –заканчивается успешно, если значения переменных равны, и неудачно, если они не равны;
• сопоставление двух переменных, одна из которых связана, а другая свободна (т.е. еще не получила значение) – заканчивается успешно, и свободная переменная принимает то же значение, что и связанная;

• сопоставление двух свободных переменных – заканчивается успешно; если впоследствии одна из переменных получает значение, то и другой переменной присваивается то же значение.

Если согласование предикатов закончилось неудачно, то делается попытка согласования данного предиката с другим одноименным клозом. Если таких клозов нет, то происходит возврат (*бэктрекинг*) к ближайшей "развилке", т.е. к точке программы, в которой было возможно другое согласование предикатов.

Рассмотрим принцип работы программ на Прологе на приведенном примере. Выполнение этой программы состоит в доказательстве целевого предиката vyvod. Для этого требуется доказать предикат poisk (предикаты write, readln и readint, также имеющиеся в теле предиката poisk, являются стандартными, и их доказательство состоит в выполнении соответствующих операций вывода и ввода).

Пусть на запрос о фамилии введено “Иванов”, а на запрос о стоимости – 1000. Таким образом, переменная F связана со значением "Иванов", а переменная S – со значением 1000. Для доказательства предиката poisk выполняется его согласование с заголовком предиката-правила poisk. Выполняется успешная унификация переменных Fam и F, St и S. Чтобы доказать предикат-правило poisk, требуется доказать предикаты, входящие в его тело. Сначала доказывается предикат rabota (Fam, Shifr), где переменная Fam связана со значением “Иванов”, а переменная Shifr – пока свободна. Сопоставление этого предиката с фактом rabota ("Антонов", "П20") завершается неудачно, так как не совпадают первые аргументы (фамилии). Поэтому происходит сопоставление предиката rabota (Fam, Shifr) со следующим фактом:

rabota ("Иванов", "П70"). Это сопоставление успешно, и переменная Shifr связывается со значением “П70”. Таким образом, предикат rabota (Fam, Shifr) доказан.

Доказывается следующий предикат в теле правила poisk: proekt (Shifr,Zak, Stoim), где переменная Shifr связана со значением “П70”, а переменные Zak и Stoim пока свободны. Сопоставление этого предиката с фактами proekt ("П20", "Рубин", 700) и proekt ("П100", "Рубин", 1400) завершается неудачей из-за несовпадения первого аргумента. Сопоставление с фактом proekt ("П70", Горизонт", 500) завершается успешно; переменная Zak связывается со значением “Горизонт”, а Stoim – со значением 500.

Доказывается предикат Stoim>St. Так как Stoim=500, а St=1000, этот предикат имеет значение “ложь”. В результате происходит возврат. Ближайшая "развилка" - предикат proekt (Shifr, Zak, Stoim), так как он еще не сопоставлялся с предикатом proekt ("П120", "Кристалл", 1100). При возврате происходит освобождение переменных Zak и Stoim (т.е. они снова не связаны ни с каким значением). Сопоставление предикатов proekt (Shifr, Zak,

Stoim) и proekt ("П120","Кристалл", 1100) завершается неудачей из-за несовпадения первых аргументов (так как Shifr=”П70”). Поэтому снова происходит возврат. Теперь ближайшая “развилка” – предикат rabota (Fam, Shifr), так как он еще не сопоставлялся с пятью фактами rabota. Переменная Shifr снова становится свободной (она освобождается при возврате), переменная Fam связана со значением “Иванов”. Сопоставление предикатов rabota (Fam, Shifr) и rabota ("Иванов", "П100") выполняется успешно, и переменная Shifr связывается со значением “П100”. Таким образом, предикат rabota (Fam, Shifr) доказан.

Снова доказывается предикат proekt (Shifr, Zak, Stoim), где переменная Shifr связана со значением “П100”, а переменные Zak и Stoim свободны. Сопоставление этого предиката с фактом proekt ("П20", "Рубин", 700) заканчивается неудачей (из-за несовпадения первого аргумента), а сопоставление с фактом proekt ("П100", "Рубин", 1400) - завершается успешно. Переменная Zak связывается со значением “Рубин”, а Stoim – со значением 1400. Снова доказывается предикат Stoim>St. Так как Stoim=1400, а St=1000, этот предикат доказывается успешно. Поэтому выполняется следующий предикат write, выводящий на экран значения переменных Shifr (“П100”) и Zak(“Рубин”). Таким образом, доказан предикат poisk, так как доказаны все предикаты, составляющие его тело. Предикат poisk является последним в целевом предикате vyvod, поэтому предикат vyvod также доказан. Выполнение программы на этом завершается.

Можно сказать, что предикат vyvod в этой программе был основной целью, а предикаты poisk, rabota и другие - временными целями.Следует отметить, что программа вывела не все проекты стоимостью свыше 1000 ден. ед., над которыми работает указанный сотрудник (Иванов). Из набора фактов легко видеть, что он работает также над проектом П120стоимостью 1100 ден. ед. Однако программа вывела только проект П100, указанный первым. Устранение этого недостатка будет рассмотрено ниже.

1.6. Механизмы управления в программах на Прологе

1.6.1. Искусственный возврат (fail)

Изменим рассмотренную выше программу таким образом, чтобы на экран выводился список всех проектов, соответствующих заданным условиям. Для этого достаточно записать предикат poisk в следующем виде:

```
poisk (Fam, St):- rabota (Fam, Shifr),
proekt (Shifr, Zak, Stoim),
Stoim >= St,
write (Shifr," ",Zak), nl, fail.
```

poisk (_, _).

Стандартный предикат fail вызывает искусственный возврат (состояние неудачи).

Программа с измененным предикатом poisk выполняется сначала так же,как и рассмотренная выше. После того, как на экран выводятся значения переменных Shifr (“П100”) и Zak (“Рубин”), возникает состояние искусственной неудачи (fail). Происходит возврат к ближайшей "развилке" proekt (Shifr, Zak, Stoim), так как он еще не сопоставлялся с двумя фактами proekt. При этом Shifr=”П100”, а переменные Zak и Stoim при возврате становятся свободными. Сопоставление предиката proekt (Shifr, Zak, Stoim) с фактами proekt ("П70", "Горизонт", 500) и proekt ("П120", "Кристалл", 1100) заканчивается неудачей из-за несовпадения первого аргумента. Происходит возврат к следующей развилке: rabota (Fam, Shifr), где Fam=”Иванов”, а переменная Shifr освобождается при возврате. Предикат rabota (Fam, Shifr) еще не был сопоставлен с четырьмя последними фактами rabota. Сопоставление этого предиката с фактами rabota ("Петров", "П100") и rabota ("Васильев", "П70") заканчивается неудачей. Сопоставление с фактом rabota ("Иванов", "П120") выполняется успешно, и переменная Shifr связывается со значением "П120". Таким образом, предикат rabota (Fam, Shifr) доказан.

Снова доказывается предикат proekt (Shifr, Zak, Stoim), где переменная Shifr связана со значением “П120”, а переменные Zak и Stoim – пока свободны. Сопоставление этого предиката с фактами proekt ("П20", "Рубин", 700), proekt ("П100", "Рубин", 1400) и proekt ("П70", "Горизонт", 500) заканчивается неудачей. Сопоставление с фактом proekt ("П120", "Кристалл", 1200) выполняется успешно. Переменная Zak связывается со значением “Кристалл”, а Stoim – со значением 1100.

Снова доказывается предикат Stoim>St. Так как Stoim=1100, а St=1000, этот предикат доказывается успешно. Поэтому выполняется предикат write, выводящий на экран значения переменных Shifr (“П120”) и Zak (“Кристалл”).

После этого снова возникает состояние искусственной неудачи, создаваемое стандартным предикатом fail. Ближайшей “развилкой” теперь оказывается предикат rabota (Fam, Shifr), так как он еще не был сопоставлен с предикатом rabota ("Васильев", "П120"). При этом Fam=”Иванов”, а Shifr – свободна (освобождается при возврате). Сопоставление предикатов rabota (Fam, Shifr) и rabota ("Васильев", "П120") выполняется неудачно из-за несовпадения первого аргумента. Таким образом, доказательство первого клоза предиката poisk завершилось неудачей. Поэтому происходит возврат к следующей "развилке" -ко второму клозу предиката poisk. В нем никаких действий не требуется, поэтому он доказывается успешно. Предикат poisk - последний в целевом предикате vyvod. Поэтому целевой предикат также доказан. Использованные во втором клозе предиката poisk

символы “_” (знак подчеркивания) используются для заполнения позиции аргумента, конкретное значение которого не требуется.

Стандартный предикат nl предназначен для перехода в следующую строку при выводе данных на экран.

1.6.2. Повторение

Усовершенствуем рассмотренную выше программу таким образом, чтобы после вывода ответа на экран выводился запрос “Продолжить ?”. При положительном ответе (при вводе буквы “д”) программа должна снова запрашивать у пользователя фамилию сотрудника и стоимость проектов, а затем выводить информацию о проектах, как показано выше. При отрицательном ответе (“н”) программа должна заканчивать работу. Программа при этом будет иметь следующий вид.

```
predicates
nondeterm vyvod
nondeterm poisk (string, integer)
nondeterm rabota (string, string)
nondeterm proekt (string, string, integer)
nondeterm repeat
goal
vyvod.
clauses
vyvod:- repeat,
clearwindow, write ("Фамилия: "), readln (F),
write ("Стоимость: "), readint (S),
poisk (F, S),
nl, write (“Продолжить ? ”), readchar(Prod), Prod=’н’.
poisk (Fam, St):- rabota (Fam, Shifr),
proekt (Shifr, Zak, Stoim),
Stoim >= St,
write (Shifr," ",Zak), nl, fail.
poisk (_, _).
repeat.
repeat:-repeat.
```

База данных (набор фактов) остается такой же, как в предыдущих примерах. Многократное повторение работы программы обеспечивается с помощью предиката repeat. Этот предикат работает следующим образом. При первом обращении к предикату repeat доказывается его первый клоз (repeat.), а второй (repeat:-repeat.) остается в качестве неиспользованной альтернативы. Если на запрос "Продолжить?" ввести любой ответ, отличный от “н”, то сравнение Prod='н' заканчивается неудачей и происходит возврат к ближайшей “развилке”.

В данном случае такая “развилка” – предикат repeat, так как у него имеется неиспользованный клоз (repeat:- repeat.). Происходит обращение к этому клозу. Предикат repeat вызывает сам себя и успешно доказывается в *первом* клозе, а *второй* клоз снова остается в качестве неиспользованной альтернативы. После доказательства предиката repeat повторяются запросы фамилии и стоимости, и выводится информация о проектах. Если затем на запрос "Продолжить ?" снова ввести ответ, отличный от “н”, то повторяется возврат к предикату repeat. Такое выполнение программы повторяется, пока на запрос "Продолжить ?" не будет введен ответ “н”. В этом случае сравнение Prod=’н’ завершается успешно, целевой предикат доказывается, и выполнение программы завершается.

Важно понимать, что предикат repeat не является стандартным. Поэтому его объявление в разделе predicates и описание в разделе clauses является обязательным. Использование названия repeat (по-английски – “повторение”) для организации повторения программ на Прологе стало традиционным, однако можно использовать и любое другое название этого предиката, например,povtor.

Использованный в данной программе стандартный предикат clearwindow предназначен для очистки экрана (точнее, для очистки текущего окна). Стандартный предикат readchar предназначен для ввода данных типа char(одиночные символы). Из приведенного примера также видно, что данные типа char в программе на Прологе заключаются в *одиночные* кавычки.

1.6.3. Отсечение

Как показано выше, если предикат имеет несколько клозов, то при доказательстве одного из них остальные сохраняются в памяти ЭВМ как нерассмотренные. В случае неудачи при доказательстве какого-либо предиката, происходит возврат к таким нерассмотренным клозам. Если требуется исключить такую возможность, то применяется стандартный предикат отсечения (!).

Пример. Имеются факты, описывающие работу сотрудников над проектами и характеристики проектов (см. примеры выше). Требуется составить программу, которая

будет запрашивать фамилию сотрудника и название заказчика; если указанный сотрудник работает над каким-либо проектом данного заказчика, то программа должна выводить ответ “да”, в противном случае – “нет”.

```
predicates
nondeterm vyvod
nondeterm otvet (string, string)
nondeterm rabota (string, string)
nondeterm proekt (string, string, integer)
nondeterm repeat
goal
vyvod.
clauses
vyvod:- repeat,
clearwindow, write ("Сотрудник: "), readln (F),
write ("Заказчик: "), readint (Z),
otvet (F, Z),
nl, write (“Продолжить ? ”), readchar(Prod), Prod=’н’.
otvet (Fam, Zak):- rabota (Fam, Shifr),
proekt (Shifr, Zak, _),
write ("Да"), nl, !.
otvet (_, _):- write ("Нет"), nl.
repeat.
repeat:-repeat.
```

Пусть, например, введена фамилия сотрудника “Иванов”, заказчик –“Горизонт”. Доказывается предикат otvet (F, Z), где F=”Иванов”, Z=”Горизонт”. Предикат rabota (Fam, Shifr) успешно сопоставляется с предикатом-фактом rabota (“Иванов”, “П70”), а предикат proekt (Shifr, Zak, _) – с предикатом- фактом proekt ("П70", "Горизонт", 500). Выполняется следующий предикат (write), и на экран выводится ответ “Да”. Предикат отсечения (!) удаляет из памяти второй клоз предиката otvet, где предусмотрен вывод ответа “Нет”.
Если затем на запрос "Продолжить ? " ответить “д”, то произойдет возврат к ближайшей “развилке”. В данном случае такой “развилкой” оказывается предикат repeat. Он обеспечивает повторение выполнения программы, как показано в п.1.6.2.

Если не указать в этой программе предикат !, то ответ “Да”все равно будет выведен, но второй клоз предиката otvet сохранится в качестве неиспользованной альтернативы. Затем будет задан вопрос "Продолжить ? ". При ответе “д” произойдет возврат, и ближайшей “развилкой” окажется неиспользованный второй клоз предиката otvet. На экран будет выведено сообщение “Нет” (хотя это не требуется), и снова задан вопрос "Продолжить ? ".

ВАРИАНТЫ ЗАДАНИЙ.

(варианты указаны номерами)

1. Написать программу для выбора меню (первое, второе, третье) с учетом имеющихся у вас денег и стоимости блюд. Должно быть в базе несколько первых блюд, несколько вторых и несколько третьих. Если имеющейся у вас суммы не хватает, программа должна предложить усеченное меню (из первого и второго или только из первого).
2. Написать программу для выбора автомобиля из базы данных по различным признакам (например, по марке, цене, стране-производителю, году). Таким образом, программа должна распознавать вопрос (что спрашивается) и выводить все подходящие автомобили.
3. Написать программу, подбирающую лекарство с учетом симптомов и противопоказаний. Каждое лекарство, представленное в базе данных, должно лечить некоторый симптом (например, головную боль или температуру или высокое давление или резь в животе и т.п.). У лекарства есть противопоказание (например, нельзя употреблять при болезнях сердца, нельзя давать больным младше 16 лет, нельзя давать при склонности к аллергии и т.п.). Самостоятельно составить базу лекарств и реализовать обработку запросов.
4. Написать программу-переводчик с русского на английский и наоборот. Список слов в базе данных порядка 10-15. Программа должна распознавать, на какой язык выполнять перевод, например, если вводим bread , то ответом должно быть хлеб, а если вводим хлеб – то ответ суть bread.
5. Написать программу, позволяющую узнать любую информацию о работнике – год рождения, адрес, образование, отдел. В базе данных хранить сведения о работниках (например, о 10-15 работниках). Программа должна распознавать запрос (что нужно найти – год рождения, отдел или адрес и т.п.).

6. Написать программу, дающую предсказания вероятности дождя по симптомам. В качестве симптомов использовать – влажность (высокая, средняя, низкая), ветер (сильный, умеренный, слабый), давление (высокое, умеренное, пониженное). Например, программа требует ввести симптомы – влажность, ветреность, давление. Примером ответа может служить – *вероятность дождя не высокая*. Знания в базе знаний составить по своему представлению, например, типа

дождь(“вероятность дождя не высокая”,“умеренная”,“сильный”,“высокое”).

Здесь второй аргумент – влажность=*умеренная*, третий аргумент – ветер=*сильный*, четвертый аргумент- давление=*высокое*.

7. Написать программу для обслуживания в библиотеке. В базе данных хранить информацию о книгах (автор, название, жанр, наличие в библиотеке). Пользователь может интересоваться книгами определенного автора, а также определенного жанра,а также по названию. Система должна распознавать, что спрашивает пользователь и выдавать подходящие книги, если они есть в библиотеке.