МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ЕЛЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ И.А.БУНИНА»

Е.В. Игонина

ОСНОВЫ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ И РЕАЛИЗАЦИЯ ПРОГРАММ НА ЯЗЫКЕ ПРОЛОГ

Учебное пособие

Елец – 2018

УДК 512.64
ББК 22.143
И 26

Печатается по решению редакционно-издательского совета
Елецкого государственного университета им. И.А.Бунина
от 29.01.2018, протокол № 1

Рецензенты:

М.А. Крутиков, кандидат педагогических наук, доценткафедры информатики, информационных технологий и защиты информации ЛГПУ им. П.П. Семенова-Тян-Шанского.
Д.В. Кузнецов, кандидат физико-математических наук, доцент кафедры физики, радиотехники и электроникиЕГУ им. И.А. Бунина.

Е.В. Игонина
И 26 Основы логического программирования и реализация программ на языке Пролог: учебное пособие. – Елец: ЕГУ им. И.А. Бунина, 2018. – 92 с.

Пособие состоит из двух частей: теоретические основы логического программирования на языке Пролог, включающие структуру программы, синтаксис языка, сопоставление, арифметические операции в Прологе, семантику Пролог-программ, работу со списками и лабораторного практикума, представленного тремя уроками-практикумами и девятью лабораторными работами. Уроки-практикумы позволяют студенту разобраться с механизмом работы в системе TurboProlog 2.0, освоить команды редактирования и команды главного меню. В лабораторных работах рассматриваются возможности применения языка Пролог для решения логических задач, при создании экспертных систем, интерфейсов на естественном языке.

Учебное пособие адресовано студентам, обучающимся по направлению подготовки «090301 – Информатика и вычислительная техника». Также оно может быть рекомендовано студентам других направлений подготовки, изучающим логическое программирование, аспирантам, преподавателям высшей школы, программистам.

УДК 512.64
ББК 22.143

© Елецкий государственный
университет им. И.А. Бунина, 2018

ВВЕДЕНИЕ

Язык Пролог является представителем семейства языков логического программирования и в сравнении с традиционными языками программирования, предназначенными для записи алгоритмов, такими как Бейсик, Фортран, Паскаль, Си, обладает существенными особенностями:

• программа на Прологе не является алгоритмом, а представляет собой запись условия задачи на языке формальной логики (т.е. это дескриптивный, описательный язык программирования);

• язык Пролог предназначен не для решения вычислительных или графических задач, а для решения логических задач, для моделирования процесса логического умозаключения человека; вычисления же и графические построения выполняются в Прологе как побочный продукт логического вывода;

• Пролог требует особого стиля мышления программиста, что затрудняет изучение его теми, кто уже привык к процедурному программированию, поэтому, так называемые, практические программисты не стремятся переходить на этот язык, что мешает росту популярности Пролога. Однако во многих странах (Японии, Англии, Франции, Германии, Израиле и т.д.) расширяется практика применения Пролога в образовании как первого изучаемого языка программирования.

Все вышеперечисленное позволяет отнести Пролог в существующем делении языков программирования на языки низкого и высокого уровня к языкам сверхвысокого уровня. В японском проекте создания в 90-х годах XX века компьютеров 5-го поколения (обладающих искусственным интеллектом) Пролог положен в основу аппаратной организации и разработки программного обеспечения.

Изучению языка Пролог очень способствует предшествующее изучение математической логики, понятийной системой которой он пользуется.

Целью настоящего пособия является изучение теоретических аспектов и практическое освоение приемов логического программирования студентами ВУЗа, обучающимся по направлению подготовки «090301 – Информатика и вычислительная техника». Учебное пособие может представлять интерес для студентов других направлений подготовки, изучающих логическое программирование, а также для аспирантов, преподавателей высшей школы, программистов.

Пособие «Основы логического программирования и реализация программ на языке Пролог» состоит из двух частей. В первой части дана общая характеристика теоретических основ программирования на языке Пролог: структура программы, синтаксис языка, сопоставление, арифметические операции в Прологе, семантика Пролог-программ, работа со списками. Вторая часть представлена тремя уроками-практикумами и девятью лабораторными работами. Уроки-практикумы позволяют студенту разобраться с механизмом работы в системе Turbo Prolog 2.0, освоить команды редактирования и ко-

манды главного меню. В лабораторных работах рассматриваются основные понятия и предикаты языка, а также возможности применения языка Пролог для решения логических задач, при создании экспертных систем, интерфейсов на естественном языке. Изложение сопровождается большим количеством примеров, оформленных в соответствии с правилами и синтаксисом языка. В конце каждой лабораторной работы размещены задания для самостоятельного выполнения и структура соответствующего отчета. Представленные примеры и предлагаемые задания ориентированы на использования системы программирования Turbo Prolog 2.0.

Так как системы программирования на Прологе для компьютеров допускают использование лишь латинских строчных и прописных букв: a ... z, A ... Z, то использование русских строчных и прописных букв: а ... я, А … Я не допускается. Для того чтобы смысл имен оставался понятным, в ходе практической работы с интерпретатором рекомендуется использовать в качестве имен запись русских слов латинскими буквами. В задачах настоящего учебного пособия все отношения и имена записаны русскими буквами, такой прием использован для того, чтобы сделать смысл программ наиболее понятным. При запуске этих программ в «англоязычных» системах программирования нужно заменять русские буквы в именах на латинские, соответствующие приемы приведены в примерах раздела 2.

Автор выражает искреннюю признательность своим студентам (Андроповой О., Полякову А., Ирицян А., Бородкину Д., Ермаковой Е.), вопросы и замечания которых в немалой степени способствовали появлению предлагаемого пособия. Не меньшую признательность автор выражает рецензентам – кандидату педагогических наук, доценту кафедры информатики, информационных технологий и защиты информации ЛГПУ им. П.П. Семенова-Тян-Шанского Крутикову М.А. и кандидату физико-математических наук, доценту кафедры физики, радиотехники и электроники ЕГУ им. И.А. Бунина Кузнецову Д.В.

Часть 1. Общая характеристика теоретических основ программирования на Прологе

§ 1. Краткий обзор языка Пролог

Пролог – это язык программирования, используемый для решения задач, в которых действуют объекты и отношения между этими объектами.

Программа на прологе состоит из *предложений*, которые могут быть *фактами*, *правилами* или *вопросами*.

Пример программы: родственные отношения.

Рассмотрим дерево родственных отношений (рис.1):

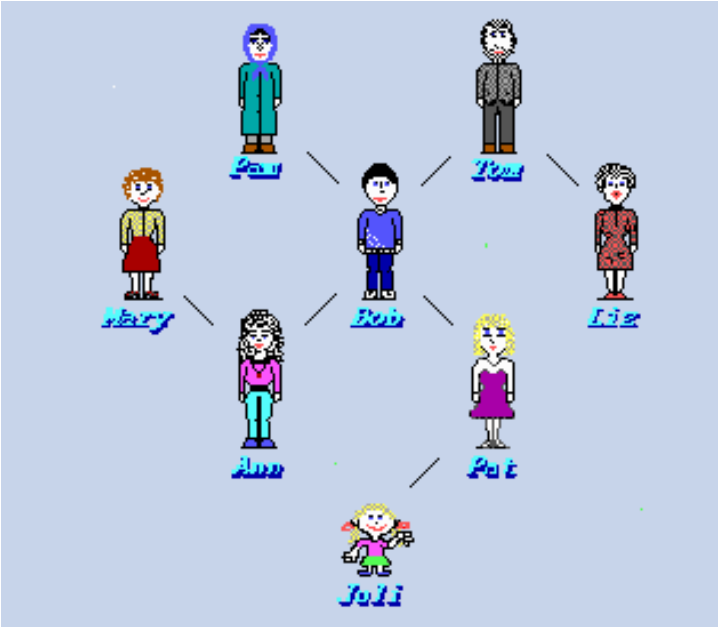


Рис. 1. Дерево родственных отношений

Факты. Введем отношение родитель (parent) между объектами: parent (tom, bob). Это факт, определяющий, что Том является родителем Боба.

Parent – имя отношения, tom, bob – его аргументы. Теперь можно записать программу, описывающую все дерево родственных отношений (рис. 2).

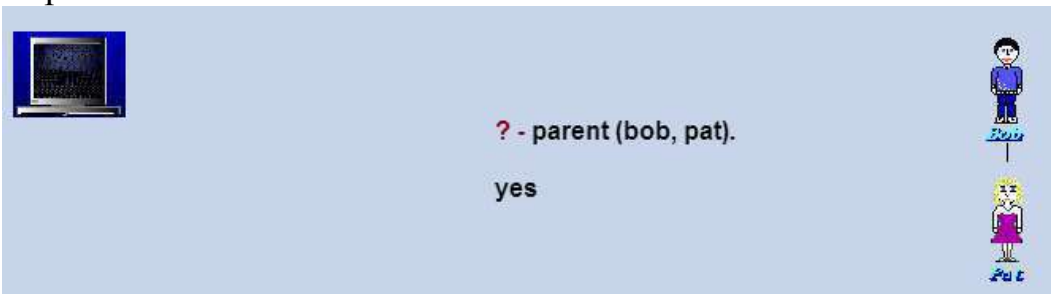
```
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(mary, ann).
parent(pat, juli).
```

Эта программа состоит из семи предложений (утверждений), *clause* (клоз). Каждый клоз записан фактом в виде отношения parent. При записи фактов надо соблюдать следующие правила:

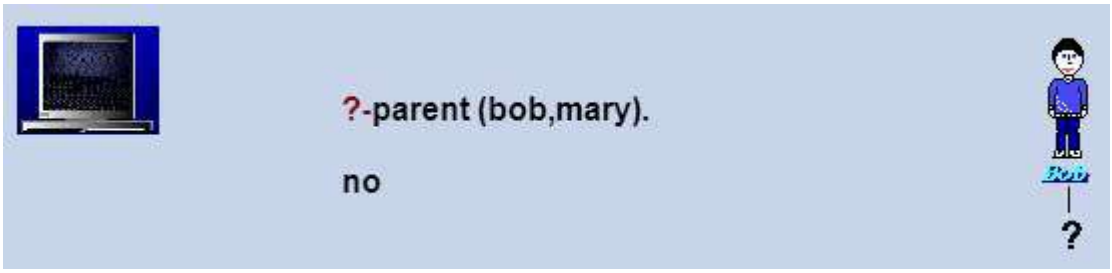
- имена всех отношений и объектов с маленькой буквы.
- сначала записывается имя отношения, затем в круглых скобках через запятую объекты.
- в конце ставится точка.

Рассмотрим еще один приме факта: like (bob, pam), где like –нравиться. Совокупность фактов в прологе называют *базой данных*.

Вопросы. К составленной базе данных можно задать *вопросы.* Вопрос в обычном прологе начинается с **?.** Вопрос записывается также, как и факт. Например:

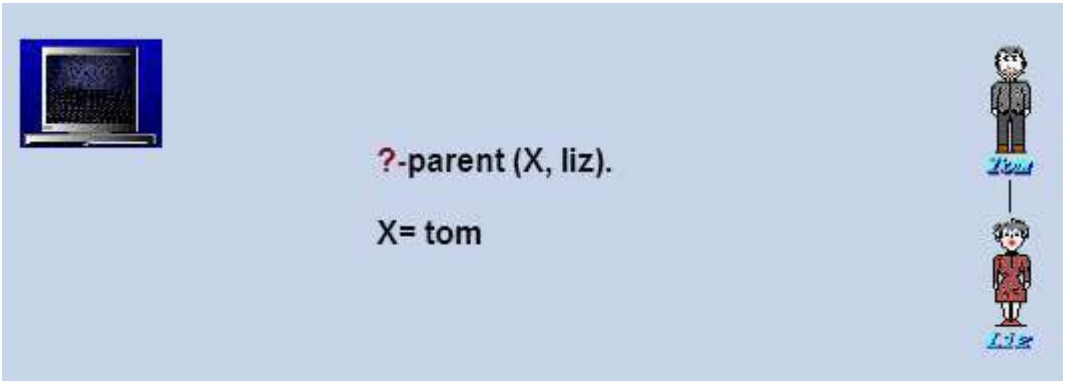


Когда пролог получает вопрос, он пытается сопоставить его с базой данных. Такой факт находится, ответ: **да (yes).** На вопрос



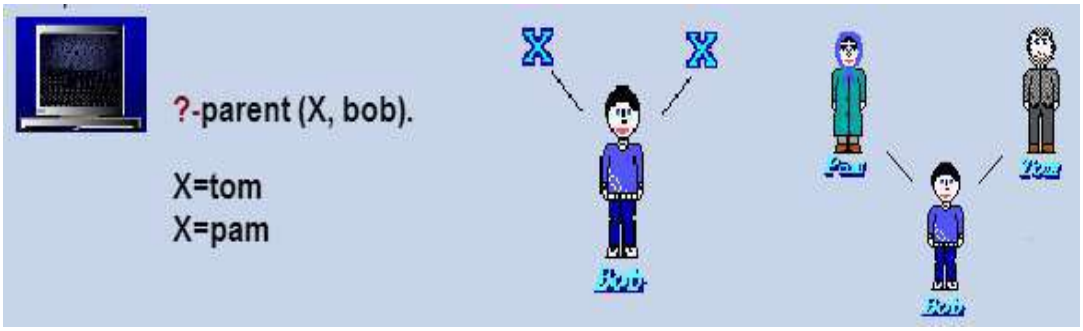
Ответ будет **нет (no)**, так как такого факта в базе данных нет.

Переменные. Можно задать вопрос и узнать кто родитель **liz**:

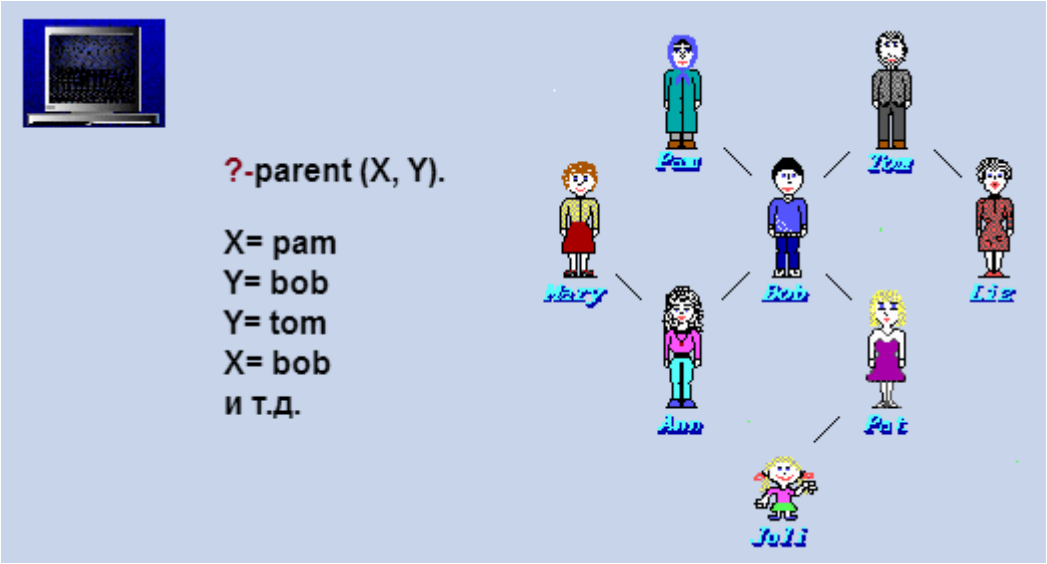


Здесь **X** – *переменная*. Ее величина неизвестна, и она может принимать значения. В данном случае ее значением будет объект, для которого это утверждение истинно.

Вопрос



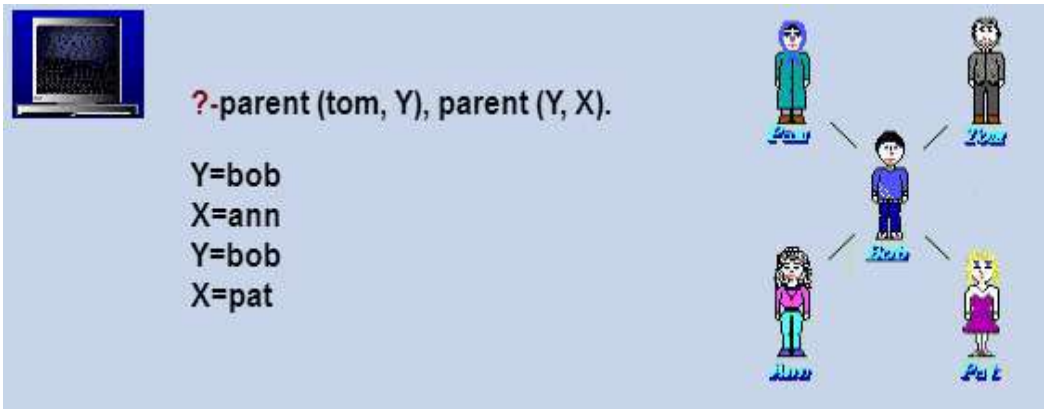
Можно задать вопрос, кто является чьим родителем. Или найти такие **X** и **Y**, что **X** является родителем **Y**.



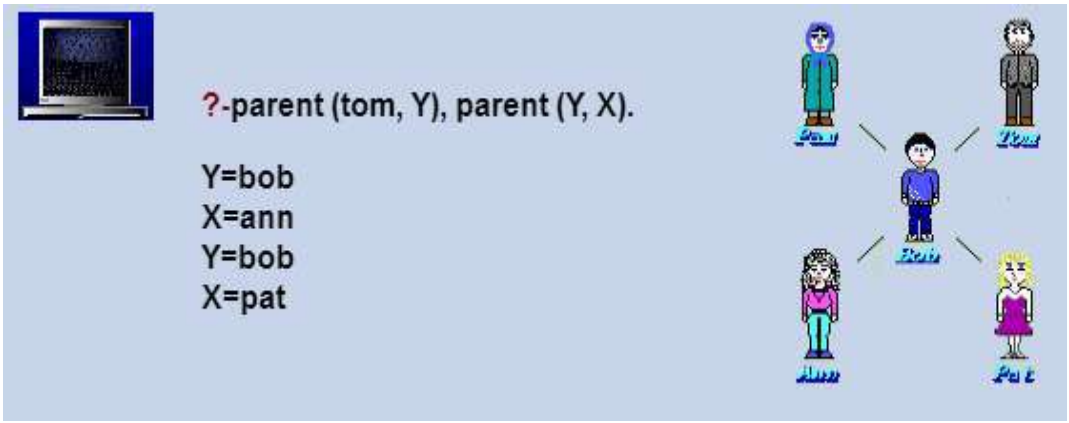
Конъюнкция целей. Можно задать более общий вопрос: Кто является родителем родителя juli. Так как нет отношения grandparent (бабушка, дедушка), то можно разбить на два вопроса:

1. кто родитель juli. Предположим – **Y**.
2. кто родитель Y. Предположим – **X**.

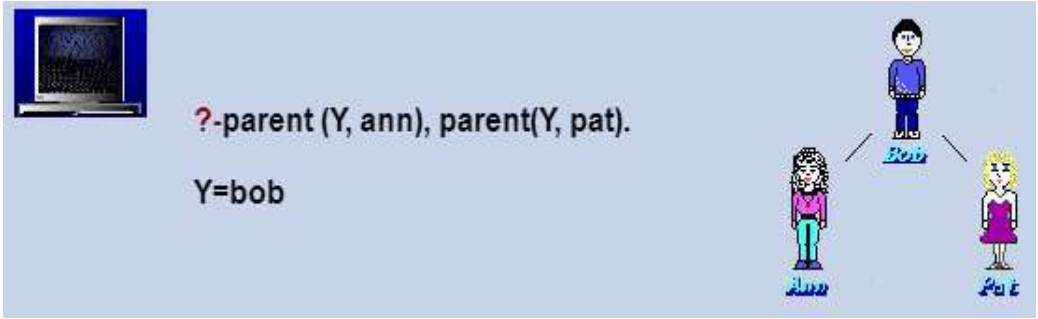
Тогда составной вопрос:



При поиске решения сначала находится **Y** , а затем по второму условию **X**. **Вопрос:** Кто внуки тома?:



Вопрос, есть ли у **ann** и **pat** общий родитель?



Правила. Введем отношение ребенок **child**, обратное к **parent** "родитель". Можно было бы определить аналогично: **child (liz, tom).** Но можно использовать, что отношение **child** обратно к **parent** и записать в виде утверждения – *правила*: **child(Y, X):-parent (X, Y).** Правило читается так: д**ля всех X и Y Y -child X, если X -parent Y.**

Правило отличается от факта тем, что факт всегда истина, а правило описывает утверждение, которое будет истинной, если выполнено некоторое условие. Поэтому в правиле выделяют: заключение условие

```
child(Y, X) :- parent (X, Y).
      голова                          тело

       head                           body
```

Если условие **parent (X, Y)**. выполняется, то логическим следствием из него будет утверждение **child(Y, X)**.

Рассмотрим, как правило используется прологом. Зададим вопрос

```
?-child(liz, tom).
```

В программе нет данных о **child**.Но есть правило, которое верно для всех **X Y**, в том числе для **liz** и **tom**. Мы должны применить правило для этих значений. Для этого надо подставить в правило вместо **X- tom**, а вместо **Y - liz**. Говорят, что переменные будут связаны, а операция будет называться *подстановкой.* Получаем конкретный случай для правила **child(liz, tom):-parent (tom, liz).** Условная часть приняла вид **parent (tom, liz).** Теперь надо выяснить выполняется ли это условие. Исходная цель **child(liz,tom)** заменяется подцелью **parent (tom, liz).**, которая выполняется, поэтому пролог ответит **"yes".**

Конъюнкция в правилах. Добавим еще одно отношение в базу данных, унарное, определяющее пол (male).

male(tom).
male(bob).
male(jim).
female(liz).
female(pam).
female(pat).
female(ann).

Теперь определим отношение **mother (мама)**. Оно описывается следующим образом: **Для всех X Y X -mother Y, if X- parent Y и X -female.**

Таким образом, правило будет **mother(X, Y):-parent(X, Y), female(X).**

Можно записать

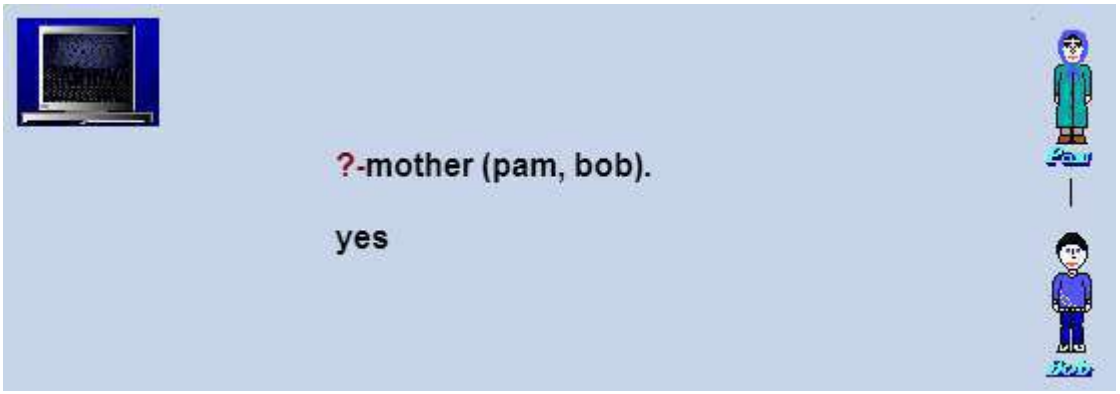
```
mother(X, Y):-parent(X, Y),
                              female(X).
```

Или

```
mother(X, Y):-
                    parent(X, Y),
                    female(X).
```

Запятая между двумя условиями означает конъюнкцию целей, т.е. два условия должны быть выполнены одновременно.

Как система ответит на вопрос?



Находится правило **mother**, производится подстановка

X=pam

Y=bob

Получаем правило **mother(pam, bob):- parent(pam, bob), female(pam).**

Сначала удовлетворяются **parent** , а затем **female**

Пролог отвечает: **yes.**

Вопрос:

Переменные в теле правила. Определим отношение **sister** (сестра): **Для любых X и Y X sister Y, if y X и Y есть общий родитель, и X female.** Запишем правило на прологе.

```
sister (X, Y):- parent(Z,X),
                parent(Z,Y),
                female(X).
```

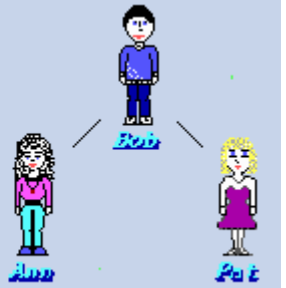
Здесь **Z**-общий родитель. **Z**-некоторый, любой. Можно спросить

```
?-sister(ann, pat).

yes

?-sister(pat, pat).

yes
```



Ответ будет "**yes**". Так как мы не потребовали, чтобы **X** и **Y** были разные.

Добавим отношение **different (X, Y)**, которое указывает, что **X** и **Y** разные.

```
sister (X, Y):- parent(Z,X),
                parent(Z,X),
                female(X),
                different (X, Y).
```

Структура программы Пролог представляется следующим образом.

```
предложение  -  факт,   правило,   вопрос

clause          fact,   rule,      goal

состав          head    head

                .       ..         ?

                        |          |

                        body       body
```

§ 2. Синтаксис Пролога. Арифметика. Сопоставление

Синтаксис Пролога. Программа на прологе состоит из *предложений* трех видов: *факты, правила, вопросы*. Все предложения строятся из термов. *Терм* является синтаксической единицей (см. рис.1).

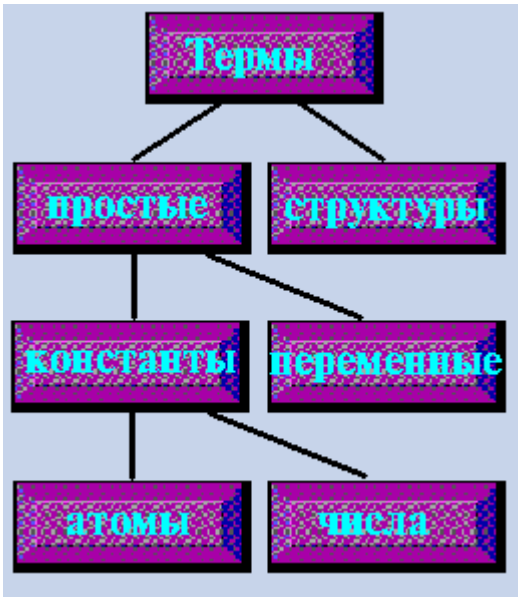


Рис.1. Виды термов

Константы – это поименованные конкретные объекты или отношения. Атомы могут задаваться:

1) цепочкой букв, цифр и символом подчеркивания '_', начиная со строчной буквы.

a
sister
x_23

2) специальными символами*: ====>

*Числа: ц*елые (диапазон -32768 32767), действительные (диапазон 1E-307 1E+308).

Переменные служат для обозначения объектов, значения которых меняются в ходе выполнения программы. Имена переменных могут начинаться или с прописной буквы, или с символа подчеркивания

X
Y
Result
_result

Если значение переменной не интересует, то можно использовать *анонимные переменные* в виде символа подчеркивания '_'. Например, **haschild(X):-parent(X, Y).** Здесь значение **Y** не интересует, поэтому можно записать **haschald(X):-parent(X, _).** Значение анонимной переменной не вы-

водится на печать. Если несколько анонимных переменных, то они все разные. Использование анонимных переменных позволяет не выдумывать имена переменных, когда не надо.

Пусть задано отношение parents для двух родителей:

parents(ann, tom, bob).

Тогда в правиле: **child(X):-parents(_, _, X). Y, Z** обе анонимные переменные разные.

Комментарий: область действия переменных – одно предложение.

Одноименные переменные в разных предложениях могут иметь разные значения.

Структура – это единый объект, состоящий из совокупности других объектов, называемых компонентами. Компоненты в свою очередь могут быть также структурами. Структура **data**. Название структуры стоит перед скобками, а компоненты внутри скобок через запятую (рис. 2). Название структуры – ФУНКТОР.

Рис. 2. Строение структуры

Структуры можно изображать в виде деревьев:

data

/ | \

27 april 1992

Корень дерева – главный функтор.

Структура с компонентами – структурами **owns(bob, book(moby_dick, mell_will)).**

Структуры можно использовать для представления геометрических фигур.

Объекты:

точка P1=point(1, 1)

точка P2=point(3, 3)

отрезок **seg(P1, P2)**

или

seg(point(1, 1), point(3, 3))

треугольник

triangle(point(2, 5), point(2, 8), point(5, 8))

Если дана точка трехмерного пространства: **point3(X, Y, Z),** то можно записать: **point(X, Y, Z).** Получается: **point(X, Y, Z) и point(X, Y)**.Это разные термы, т.к. каждый функтор различается двумя параметрами:

- именем,
- арностью (числом атомов).

Поэтому пишут: **point/2** или **point/3**. Например, если написано: Отношение point/2 задано .... . Это означает, что задано point(_,_) , а не point/2(_,_). Это типичная ошибка.

Операторы как функторы. Некоторые функторы удобнее записывать, как операторы. Например, можно записать **+(1, 2)** или

```
  +
 / \
1   2
```

Удобнее записать **1+2** , т.е. в виде оператора. Причем надо понимать, что это не операция сложения, а операторная запись структуры. Такие операторы называются *инфиксными***.** Аналогично операторная запись **2*a+b*c** может быть представлена в виде структуры:

+(*(2, a), *(b, c))

Это и производит пролог при трансляции операторных выражений. Надо четко понимать, что операторы – это другая форма записи структуры.

Арифметика. В прологе выполняются следующие операции: **+**, **-**, *****, **/**, **mod –** остаток от целочисленного деления. Если записать:

```
?-X = 2+1.
X=2+1,
```

то получим просто сопоставление переменной и структуры. Чтобы арифметическое выражение рассчитывалось, необходимо использовать встроенный оператор **is**, который заставляет выполнять арифметические операции.

```
?-X is 2+1.
X=3
?-Y is 2*(5+6).
Y=22
```

Могут быть переменные, но они должны иметь значение к моменту вычисления: **f(X, Y, Z):-Z is X*X + Y*Y** и задать вопрос

```
?- f(2, 3, R).
R=13
```

Если определить: **sum(X1, X2, X):-X is X1 + X2.**

?-sum(2, 3, X).
X=5
?- sum(1, 1, X1), sum(X1, X1, X).
4.

Операции сравнения используются при сравнении чисел, в Прологе есть операции **X > Y, X < Y, X >= Y, X =< Y** , равенство для любых термов **X = Y, X \= Y***(с побочным эффектом)*. Например, задачи сравнения

?- 100 > 4.
yes

или задача на ограничение по возрасту (**age)**

age(mary, 20).
age(ann , 23).
age(bob , 25).

?- age(X , Y), Y>21.
X=ann;
X=bob;
?- age(X , Y), Y>21, Y<=23.
X=ann ;

Определим отношение "выше" **higher/2,** используя данные рис.3.

higher(point(X1, Y1), point(X2, Y2):-Y1>Y2.

?- higher(point(1, 1), point(2, 2).
no

Если наоборот - то будет **yes**.

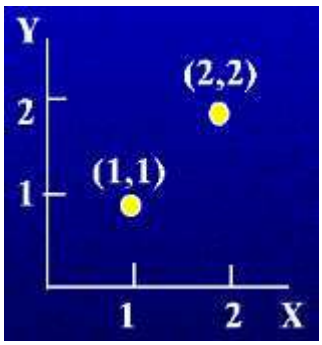


Рис. 3. Расположение точек

Сопоставление. Главной операцией в процессе выполнения Пролог-программы, является *сопоставление* (согласование, унификация) термов. Например, **parent(pam, bob).**

?- parent(pam, bob).
Yes

Цель согласуется.

?-parent(pam, X).
X=bob

Также согласуются, однако в последней программе **X** конкретизируется и принимает значение **bob**, но если взять два составных терма – структуры **a(b, C, d(e, F, g(h, i, J)))** и **a(B, c, d(E, f, g(H, I, j)))**. Согласуются ли они? Что будет с переменными после согласования? Как они будут конкретизированы?

Сопоставление – это процесс, на вход которого подаются два терма, а он проверяет, соответствуют ли эти термы друг другу. Если термы не сопоставимы, значит сопоставление терпит неудачу. Если термы сопоставимы, тогда процесс сопоставления находит конкретизацию переменных, делающих эти термы тождественными, и завершается удачей (рис. 4.).

Рис. 4. Процесс сопоставления

Какие же правила определяют сопоставимость двух термов **S** и **T**?

1. Если **S** и **T** константы , то **S** и **T** сопоставимы только, если они являются одним и тем же объектом, т.е. **2** сопоставляется с **2, bob** сопоставляется с **bob**.
2. Если **S** переменная, а **T** – произвольный объект, то ни сопоставимы и **S** приписывается значение **T**. Наоборот, если **T** – переменная, а **S** – произвольный объект, то **T** приписывается значение **S**. Говорят, что **T** конкретизируется значением **S**, т.е.:

   data(M, D, 1992).
   data(may, 3, Y).

   Сопоставимы, переменные конкретизируются:
   M=may
   D=3
   Y=1992
   в обоих термах.
3. Если **S** и **T** – структуры, то они сопоставимы, если
   a) **S** и **T** имеют одинаковый главный функтор
   и
   b) все их соответствующие компоненты сопоставимы.

Результирующая конкретизация определяется сопоставлением компонент.

Если сопоставить структуры
a(b, C, d(e, F, g(h, i, J)))
a(B, c, d(E, f, g(H, I, j))),
то получим
B=b
C=c
E=e
F=f
H=h
I=i
J=j
Рассмотрим более сложный пример:
triangle(point(2, 5), A, point(B, 8))
triangle(X, point(2, 8), point(5, 8))
После конкретизации
X=point(2, 5)
A=point(2, 8)
B=5
Еще один пример:
конкретизированная переменная X=2
!
triangle(point(X, 5), point(X, 8), point(5, Z))
неконкретизированная переменная Y=2
!
triangle(point(2, 5), point(Y, 8), point(5, A))

Если **Y** представляет собой не конкретизированную переменную, а переменная **X** конкретизированную, то **X** и **Y** согласуются и принимает значение **X**, т.е.

X=2
Y=2

Переменные **Z** и **A** обе не конкретизированы. Они согласуются и становятся ***сцепленными***. Если две переменные сцеплены, то при конкретизации одной из них, второй переменной автоматически будет присвоено тоже самое конкретное значение, что и первой. Как это было с **X** и **Y**.

Другое значение операции = в Прологе. В Прологе операция = , кроме сравнения выполняет сопоставление двух термов, с конкретизацией переменных. Если термы согласуются, то результат истина.

?-a(B,c(x,D))=a(d,c(X,3)).
Yes
B=d
X=x
D=3

Аналогично, операция /= дает истину, если термы не согласуются.

?-a(x)/=a(d).
Yes

Примеры сопоставления структур. Рассмотрим структуры, описывающие отрезки (см. рис.5). Два факта, описывающие свойство вертикальность
и горизонтальность.

vertical(seg(point(X,Y),point(X,Y1))).
horiz(seg(point(X,Y),point(X1,Y))).

?-vertical(seg(point(2,2),point(2,5))).
Yes
?- horiz(seg(point(1,4),point(2,4))).
Yes
?-horiz(seg(point(1,4),point(2,X))).
X=4

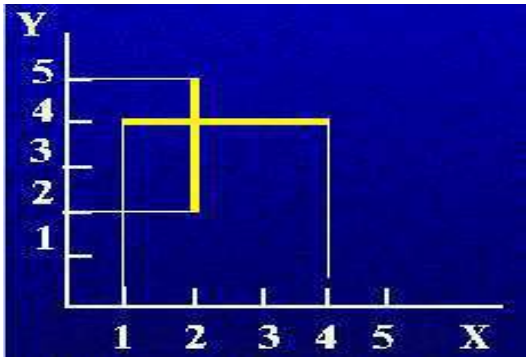


Рис. 5.

§ 3. Семантика Пролог-программ

Различают *декларативную* и *процедурную семантику* (смысл, понимание) пролог-программ. Рассмотрим декларативный смысл более подробно.

Декларативный смысл, касается только отношений, определенных в программе. Декларативная семантика определяет, что должно быть результатом работы программы, не вдаваясь в подробности, как это достигается.
Пусть задано **P:-Q, R,** где **P, Q, R –** термы (см. рис. 1).

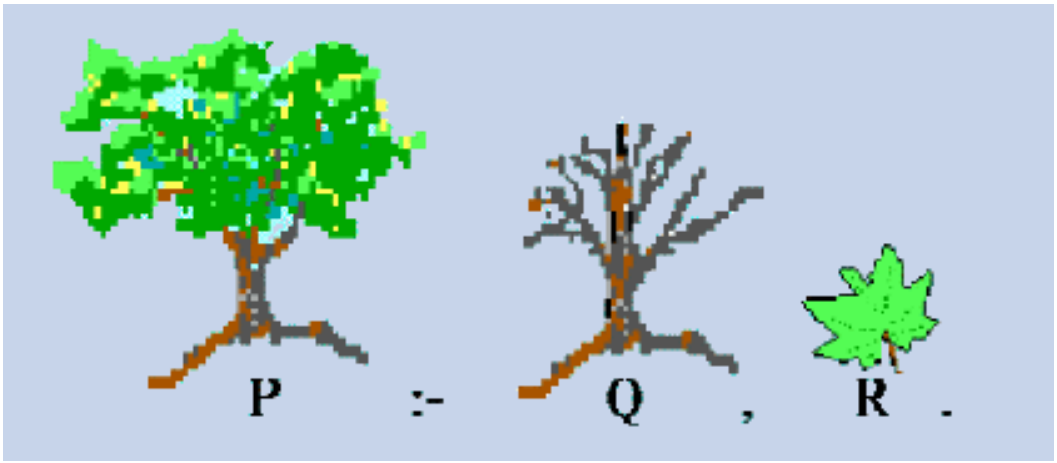


Рис. 1.

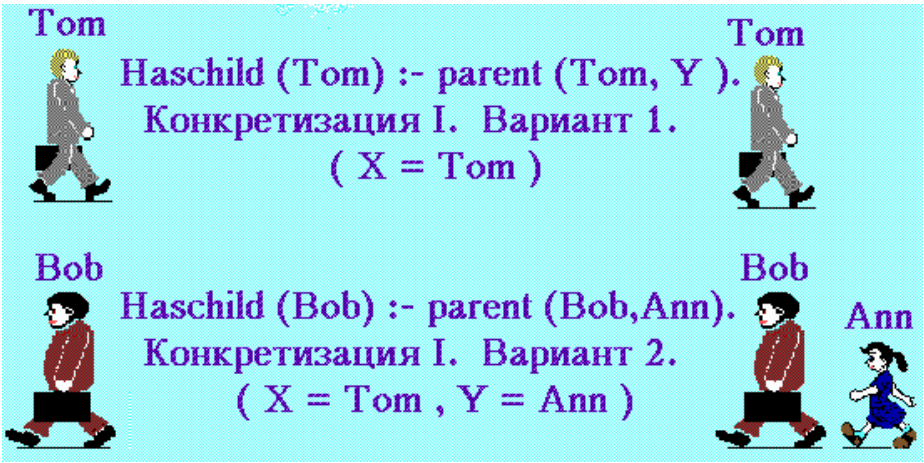
Тогда с точки зрения декларативного смысла это предложение читается:**" P-истиино, если Q и R истинны"** Или **" Из Q и R следует P**", т.е. определяются логические связи между головой предложения и целями в его теле. Таким образом, декларативный смысл программы определяет, является ли данная цель истинной (достижимой), и если – да, то при каких значениях переменных она достигается.

Конкретизацией I предложения C называется результат подстановки в него на место каждой переменной некоторого терма. Заметим, что это отличается от конкретизации переменной.

Пример 1.

haschild(X):-parent(X ,Y).

Предложение С.



Определение. Пусть дана некоторая программа и цель **G**, тогда в соответствии с декларативной семантикой, можно утверждать (см. рис. 2), что цель **G** истинна (т.е. достижима или логически следует из программы) тогда и только тогда, когда в программе существует предложение **C** такое, что существует такая его **(C)** конкретизация I, что

a) голова I совпадает с **G** и

b) все цели в теле **I** истинны.

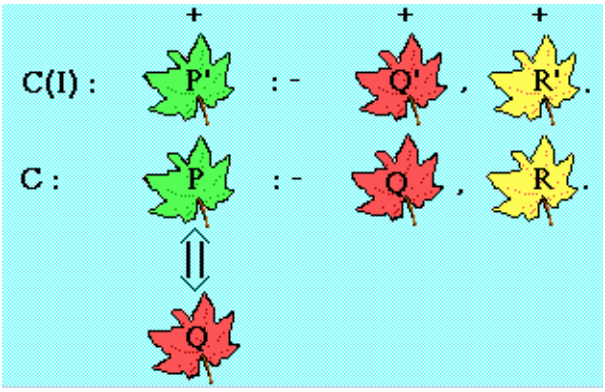


Рис. 2.

Пример 2.
female(ann).
C(I): parent(ann, bob).
C:
mother(ann):-parent(ann, Y), female(ann).
mother(X) :-parent(X, Y), female(X).
?- mother(ann).

Это определение можно распространить на вопросы следующим образом. В общем случае вопрос – список целей, разделенных запятыми. *Список целей называется истинным (достижимым),* если все цели в этом списке истинны, достижимы, при одинаковых конкретизациях переменных. Запятая между целями означает конъюнкцию целей, и они должны быть все истинны.

Дизъюнкция целей. Возможна дизъюнкция целей: истинна должна быть по крайней мере одна из целей. Дизъюнкция обозначается точкой с запятой ";". Например **P:-Q;R.** Читается: **P** истина, если **Q** – истина или **R** – истинна, т.е. это то же самое, что **P:-R** и **P:-Q.** Запятая связывает цели сильнее, чем точка с запятой. Таким образом предложение **P:-Q, R;S, T, U** понимается как **P:-(Q, R);(S, T, U)** и имеет смысл

P:-Q, R.
P:-S, T, U.

Процедурная семантика. Процедурная семантика (процедурный смысл) пролог-программы определяет, как пролог-программа отвечает на вопросы. Ответить на вопрос: «Что значит удовлетворить цели?». Поэтому процедурная семантика пролога – это процедура вычисления списка целей с учетом программы (см. рис.3).

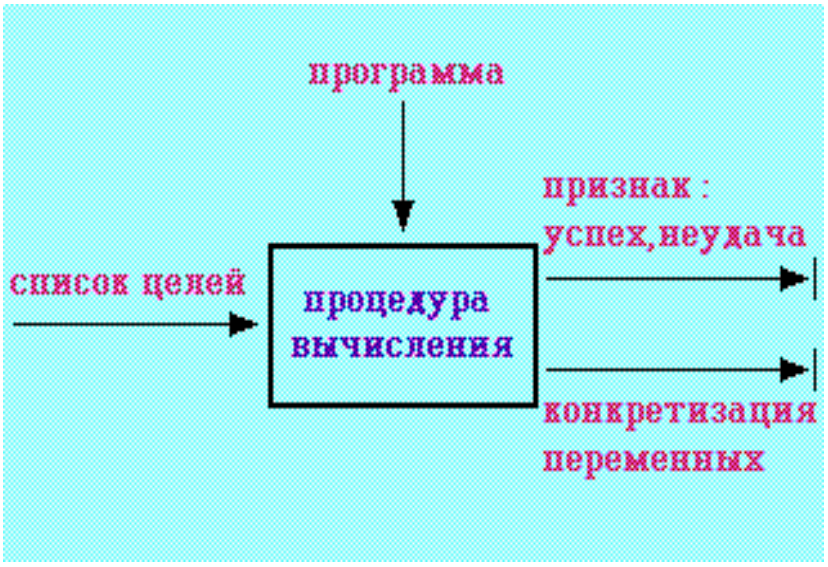


Рис. 3.

Пример вычисления. Рассмотрим программу и на ее примере – процедуру вычисления списка целей.

Программа.
1.большой(медведь).
2.большой(слон).
3.маленький (кот).
4.бурый(медведь).
5.черный(кот).
6.серый(слон).
7.темный(Z):-черный(Z).
7.1
8.темный(Z):-бурый(Z).
8.1
9.?-темный(X),большой(X).
9.1 9.2

Предложения пронумерованы для удобства. Алгоритм решения задачи представлен на рис.4.

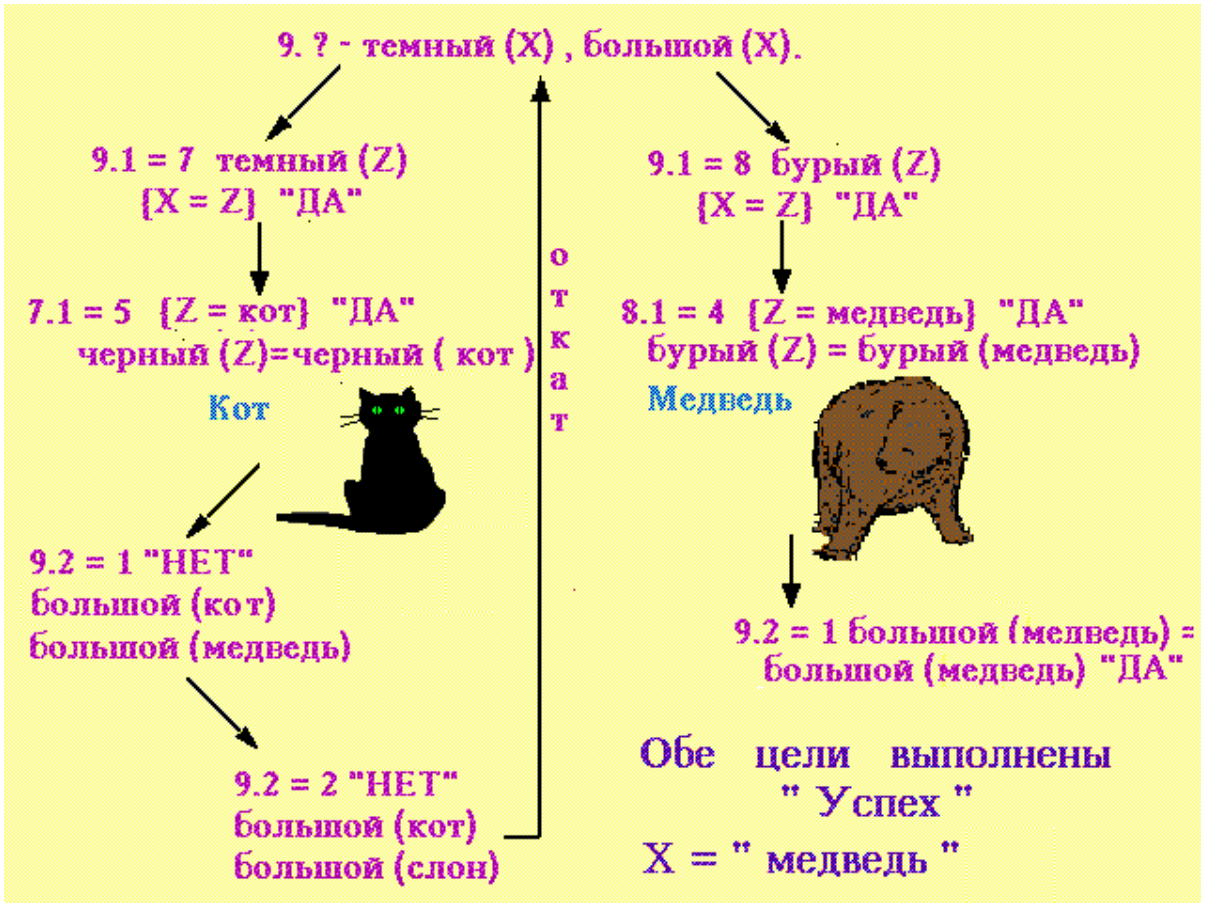


Рис. 4. Алгоритм рассуждения

Таким образом, для вычисления целей потребовалось 7 сопоставлений и один откат.

Формальное описание процедуры вычисления целей. Пусть дан список целей

G1	G2	. . .	Gm

1. Если список целей пуст, вычисление дает успех, если нет, то выполнятся пункт 2.

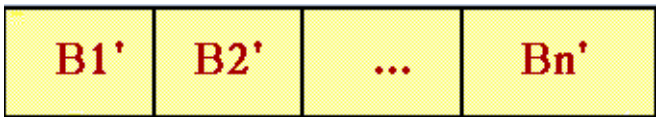
2. Берется первая цель **G1** из списка. Пролог выбирает в базе данных, просматривая сначала, первое предложение **C**,

C: H :- B1, B2, ..., Bn.

голова которого, сопоставляется с целью **G1**.

Если такого предложения нет, то неудача.

Если есть, то переменные конкретизируются и цель **G1** заменяется на список целей с конкретизированными значениями переменных.



3. Рассматривается рекурсивно через п.2 новый список целей.
4.

Если **C** – факт, то новый список короче на одну цель (n=0).

Если вычисление нового списка оканчивается успешно, то и исходный список целей выполняется успешно.

Если нет, то новый список целей отбрасывается, снимается конкретизация переменных и происходит возврат к просмотру программы, но начиная с предложения следующего за предложением C.

Описанный процесс возврата называется *бэктрекинг (backtracking).*

Соотношение между процедурным и декларативным смыслом. Создавая пролог программы всегда надо помнить о процедурном и декларативном смысле. *Декларативный смысл* касается отношений, определенных в программе, т.е. декларативный смысл определяет, что должно быть результатом программы. С другой стороны, *процедурный смысл* определяет, как этот результат может быть достигнут, т.е. как реально отношения обрабатываются прологом.

§ 4. Списки. Встроенные предикаты

Списки. *Списки* – такая же важная структура данных в Прологе, как и в Лиспе. Список в Лиспе **(a b c d) (1 2 (3))** записывается на Прологе **[a,b,c,d] [1,2,[3]]**, т.е. элементы записываются в квадратных скобках через запятую. Элементами списка могут быть любые термы. Пустой список - не **nil** , а **[ ]**.

Представление списка диаграммой. Список в Лиспе можно представить через функцию **cons** (см. рис. 1)

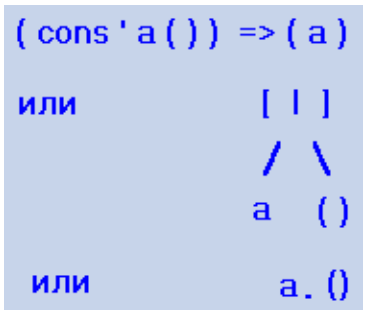


Рис. 1.

В Прологе функции **cons** соответствует функтор "." (точка), записи **(a,[])** соответствует **[a]** , это другая форма записи или

Сответственно список **[a,b,c]** представляется как структура .**(a ,.(b,.(c,[])** или в виде дерева, или диаграммой "виноградная лоза" (рис. 2).

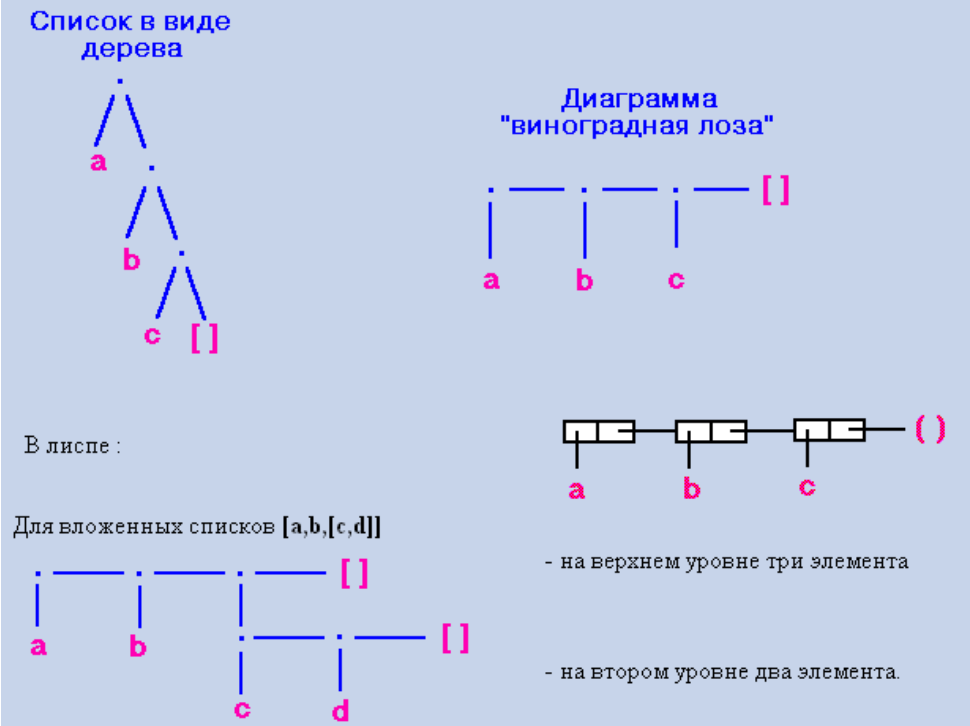
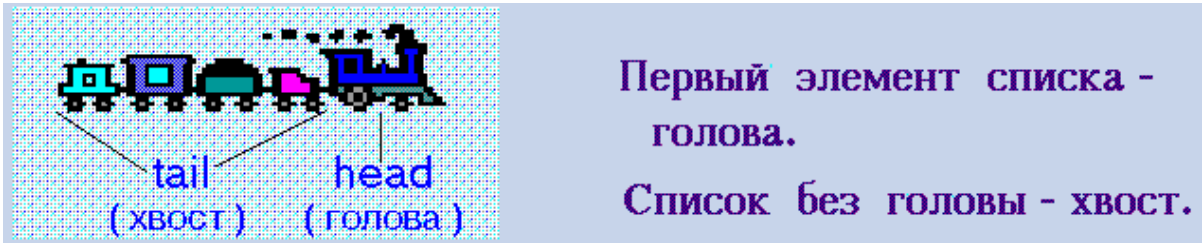
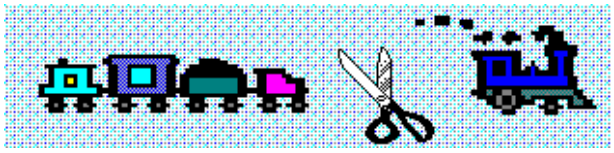


Рис. 2. Диаграмма "виноградная лоза"

Выделение головы и хвоста списка.



Главной операцией при работе со списками является расщепление списка на голову и хвост.



В Лиспе для этого используются функции car и cdr. В Прологе имеется специальная форма представления списка, называемая **cons-формой** записи:

[Head|Tail] или [H|T] [a|[]] = [a]

При конкретизации формы списком **H** сопоставляется с головой списка, а **T** - с хвостом. Например,

```
p([a,b,c]).
?-p([X|Y]).
 yes
 X=a
 Y=[b,c]
```

Таким образом, выделяются одновременно голова списка и хвост.

Рассмотрим сопоставление двух списков рис.3:

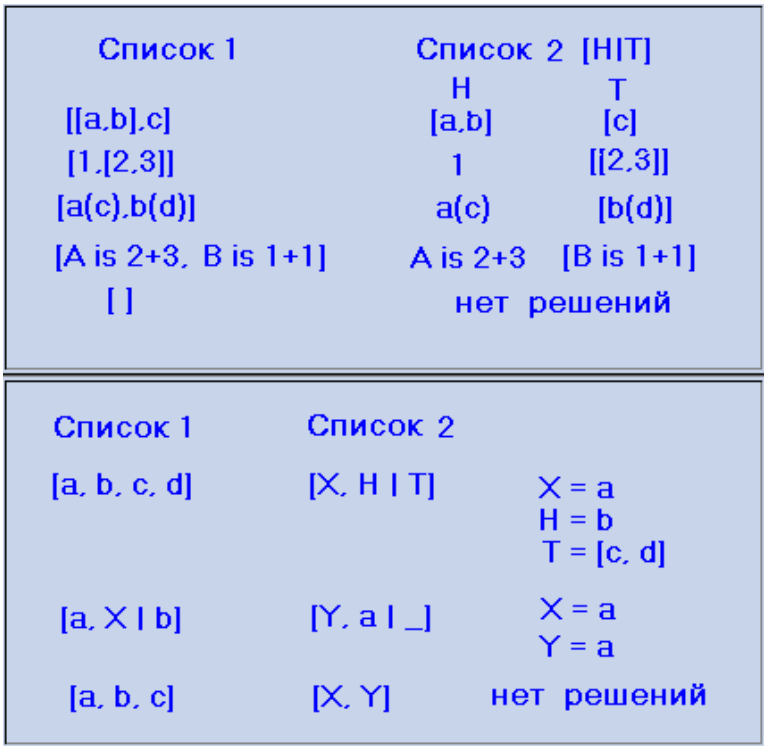


Рис. 3.

Шаблоны списков. *Шаблон (образец) списка* – это форма описания множества (семейства)списков, обладающих определенными свойствами. Например:

Шаблон списка [X|Y] описывает любой список, состоящий не менее чем из одного элемента.

Шаблон [X,Y|Z] – список, состоящий не менее чем из двух элементов.

Шаблон [b|Z] – список, первым элементом которого является b.

Шаблон [Y,X,Z] – список из трех элементов.

Шаблоны списка используются при описании процедур работы со списками.

Определения отношений через cons форму списка.

Задача 1. Определить отношение **replace_first**, которое заменяет первый элемент списка новым

Например
?-replace_first([a,b,c],w,X).
X=[w,b,c]
Это отношение:
replace_first([H|T],A,[A|T]). или
replace_first([_|T],A,[A|T]).

Процедуры обработки списков. *Процедура в прологе* – это совокупность предложений с головами, представленными одинаковыми термами.
Для обработки списков используются типовые процедуры, аналогичные функциям Лиспа:

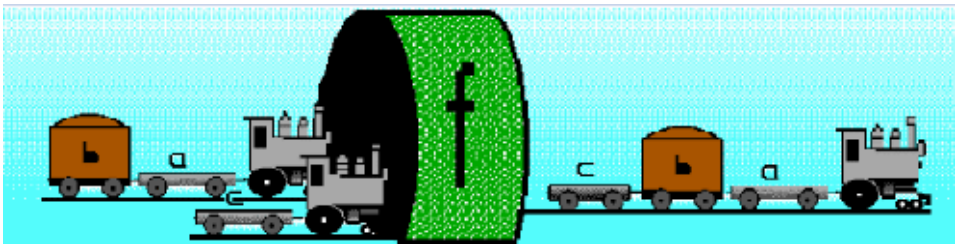
- **member** проверяет принадлежность элемента списку.

member(X, L)
Если **X** принадлежит **L**, то истина и ложь в противном случае.
С точки зрения декларативного смысла:
1) **X** принадлежит списку, если **X** совпадает с головой списка.
2) **X** принадлежит списку, если **X** принадлежит хвосту списка.
Можно записать:
member(X, [X|T]).
member(X, [H|T]) :-member(X, T).
С точки зрения процедурного смысла – это *рекурсивная процедура.* Первое предложение терминальное условие. Когда хвост будет равен **[]** проверка остановиться. Второе предложение рекурсивное. Сокращение списка происходит за счет взятия хвоста **(cdr-рекурсия)**.
Примеры применения
?-member(a, [a, b, c]).
Yes
?-member(X, [a, b, c]).
Yes
X = a
X = b
X = c

Ответьте на вопрос:
?-member(a, X).

- **append и**спользуется для соединения двух списков. т.е

append (L1, L2, L3)
L1 и **L2** - списки, а **L3** - их соединение.
?-append ([a, b], [c], [a,b,c]).
Yes

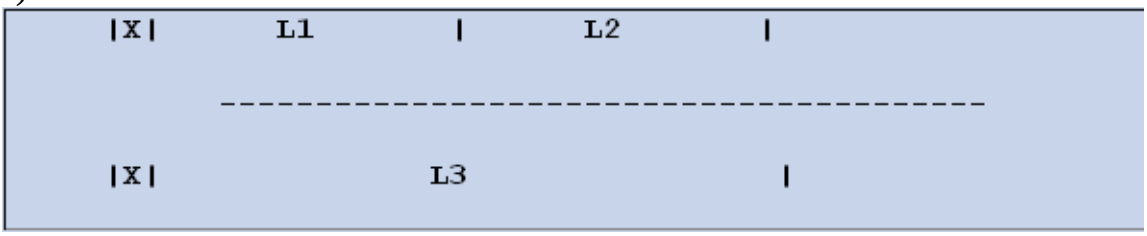


Для определения процедуры append используем два предложения:

1) если присоединить пустой список **[]** к списку **L**, то получим список **L**.
   append([], L, L).
   append([X|L1], L2, [X|L3]):- append(L1, L2, L3).

2) если присоединить не пустой список **[X|L1]** к списку **L2**, то результатом будет список **[X|L3]**, где **L3** получается соединением **L1** к **L2** :

3)



С точки зрения процедурной семантики: первое предложение – **терминальное условие**, второе – **рекурсивное с хвостовой рекурсией**.

Рассмотрим примеры применения

?-append([a], [b, c], L).
L=[a, b, c]
?-append([a], L, [a, b, c]).
L=[b, c]

?-append(L, [b, c], [a, b, c]).
L=[a]

Можно использовать для разбиения

?-append(L1, L2, [a, b, c]).
L1=[]
L2=[a, b, c];
L1=[a]
L2=[b, c];
L1=[a, b]
L2=[c];
L1=[a, b, c]
L2=[]

Процедуру **append** можно использовать для поиска комбинаций элементов. Например, можно выделить списки слева и справа от элемента

?-append(L, [3|R], [1, 2, 3, 4, 5]).
L=[1, 2]
R=[4, 5]

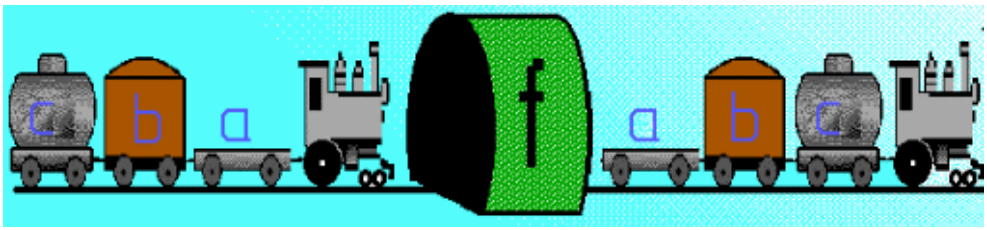
Можно удалить все, что следует за данным элементом и этот элемент тоже:

?-L1=[a, b, c, d, e], append(L2, [c|_], L1).
L1=[a, b, c, d, e]
L2=[a, b]

Можно определить процедуру выделяющую последний элемент в списке:

last(X, L):-append(_, [X], L).

- **reverse –** эта процедура обращает список.



1)Пустой список после обращения – пустой.
reverse([], []).
2)Обратить список **[X|L1]** и получить список **L2** можно, если обратить список **L1** в **L3** и в хвост ему добавить X
reverse([X|L1], L2):-reverse(L1, L3),
append(L3, [X], L2).

reverse([X|L1], L2):-reverse(L1, L3),
append(L3, [X], L2).

Длина списка. Можно, используя рекурсивный вызов, легко посчитать длину списка:

length([], 0).
length([X|L], N):-length(L, M), N is M+1.
?-length([a, b, c], N).
N=3
Встроенные предикаты. До настоящего момента рассматривали полученные ответы на задачу в форме предлагаемой прологом, а именно:

1) печатались значения переменных присутствующих в вопросе;

2) вопросы полностью записывались после запроса "?-".

Однако можно задавать вопросы и получать ответы в произвольной форме. Для этого достаточно использовать т.н. встроенные предикаты. *Встроенные предикаты* – предикаты, исходно определенные в прологе, для которых не существует процедур в базе данных. Когда интерпретатор встречает цель, которая сравнивается с встроенным предикатом, он вызывает встроенную процедуру. Встроенные предикаты обычно выполняют функции не связанные с логическим выводом. При сопоставлении встроенные предикаты обычно дают побочный эффект, который не устраняется при бэктрекинге (поиск с возвратом (англ. backtracking), общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве).

Встроенные предикаты обеспечивают возможности ввода-вывода информации:

- **write/1** – этот предикат всегда успешен. Когда вызывается, то побочным эффектом будет вывод значения аргумента на экран. При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект.
- **nl/0** – этот предикат всегда успешен. Когда вызывается, то побочным эффектом будет перевод на следующую строку. При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект.
- **tab/1** – этот предикат всегда успешен. Когда вызывается, то побочным эффектом будет печать количество пробелов заданное аргументом. Аргумент должен быть целым. При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект.
- **read/1** – этот предикат читает терм , который вводится с клавиатуры и заканчивается точкой. Этот терм сопоставляется с аргументом. При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект.

Например,

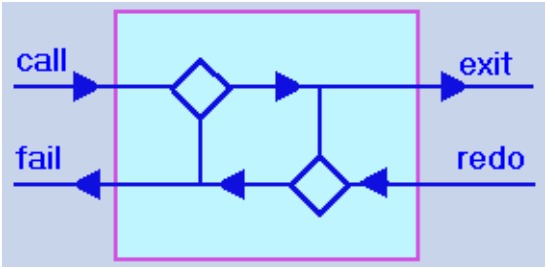
pr1:- read(X),nl,write('X='),tab(2),write(X).

При вызове

?-pr1.

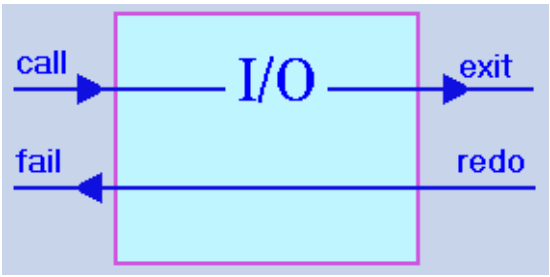
последовательность термов читает значение X, переводит строку, печатает 'X=',пропускает два пробела и печатает значение X.

Процедурный смысл встроенных предикатов ввода-вывода. Определяя встроенные предикаты, мы писали: "этот предикат всегда успешен. Когда вызывается, то побочным эффектом будет... При бэктрекинге предикат дает неудачу. Бэктрекинг не сбрасывает побочный эффект." Обычно в Прологе вход в цель возможен через **"call"** при вызове и через **"redo"** при бэктрекинге. Цель имеет внутреннюю структуру:



Первый ромб – решение при вызове **call.** Второй ромб – решение при сбросе **redo.**

Для встроенных предикатов нет внутренних точек решения внутри цели. Она представляется в виде:



Таким образом, проход через встроенный предикат будет или **call - exit,** или **redo - fail.**

Ввод-вывод списков. Для ввода-вывода списков возможны следующие два способа.

1. Ввод-вывод списка как терма. При этом способе список рассматривается как один терм. Например, процедура:

```
pr:-write('Введите список L:'),nl,
 read(L),nl,
 write('Список L='),
 tab(2), write(L),nl.
```

При вызове цели **?-pr.** она будет выполняться следующим образом:

```
 Введите список L:
 [a,b,c,d].
 Список L= [a,b,c,d]
```

2. Поэлементный ввод-вывод списка. Данный способ может быть организован с помощью рекурсивно определенных процедур.

```
read_list([X|T]):-write('Введите элемент: '),
read(X),
 X\==end,!,
 read_list(T).
```

где **end** – терм , означающий конец списка.

```
read_list([]).
```

```
write_list([]):-nl.
write_list([H|T]):-write(H),
tab(2),
write_list(T).
```

Тогда после вызова цели:

```
?-read_list(L),nl,write('Список='),nl, write_list(L).
```

Возникает следующий диалог:

```
Введите элемент: a.
Введите элемент: b.
Введите элемент: c.
Введите элемент: d.
Введите элемент: end.
Список= a b c d
```

Часть 2. Лабораторный практикум

УРОК-ПРАКТИКУМ № 1.
Запуск на DOS BOX

Цель урока-практикума: установка и отработка запуска программы в системе Turbo Prolog ver 2.0.

Для работы с программой Turbo Prolog ver 2.0 первым этапом воспользуемся интернет-ресурсами и скачаем программу DOS BOX (DOS-BOX http://www.softportal.com/software-41.), которая позволяет запустить саму программу Turbo Prolog ver.2.0. Ярлык программы см. на рис.1.

Рис.1. Ярлык DOS BOX

Затем скачаем саму программу Turbo Prolog ver 2.0 (Turbo Prolog ver 2 http://www.fayloobmennik.net/2175830). Заметим, что все программы, предложенные в лабораторных работах настоящего пособия, написаны именно для версии Turbo Prolog ver 2.0. Если в вашем распоряжении будет Пролог более новой версии, то некоторые программы не будут работать, необходима будет их корректировка с учетом синтаксиса используемой версии.

Далее отправляем папку Turbo Prolog в корневой каталог, лучше на диск С. Запускаем DOS BOX и пишем команду **moont c** указанием пути к папке, содержащей пролог как на рис.2, теперь можем запустить папку.

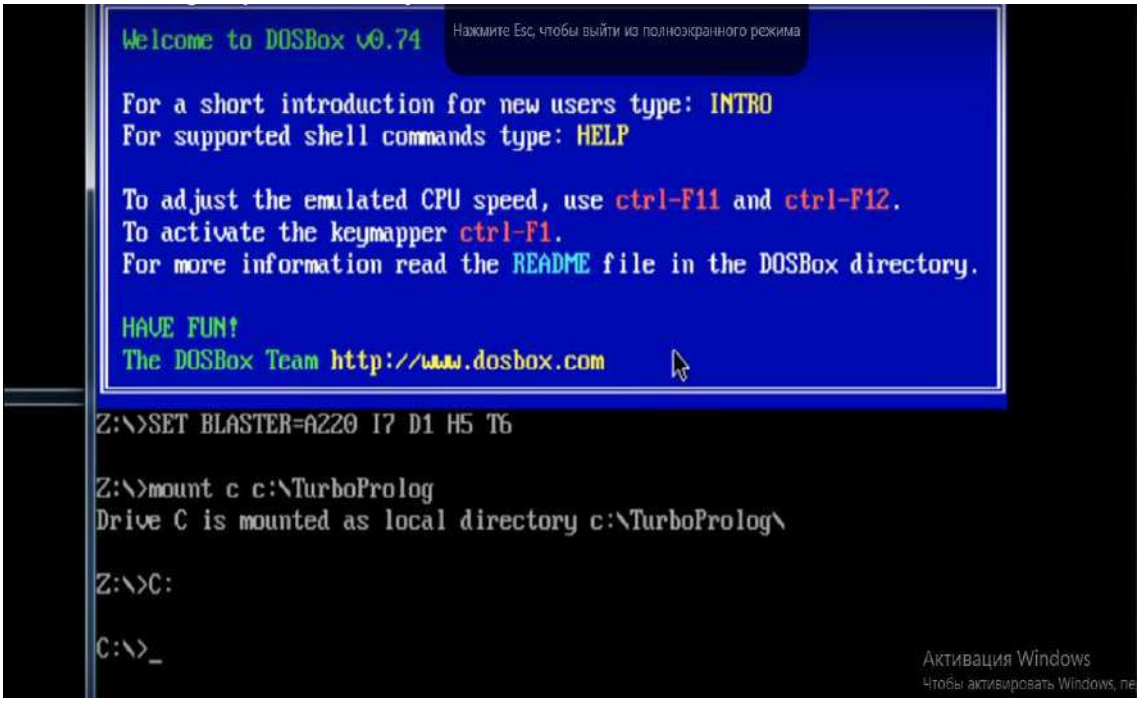


Рис. 2. Переход с помощью DOS BOX к папке Turbo Prolog

Для запуска Turbo Prolog вводим название файла PROLOG.EXE в строке с указанием диска C (см. рис.3 последняя строка).

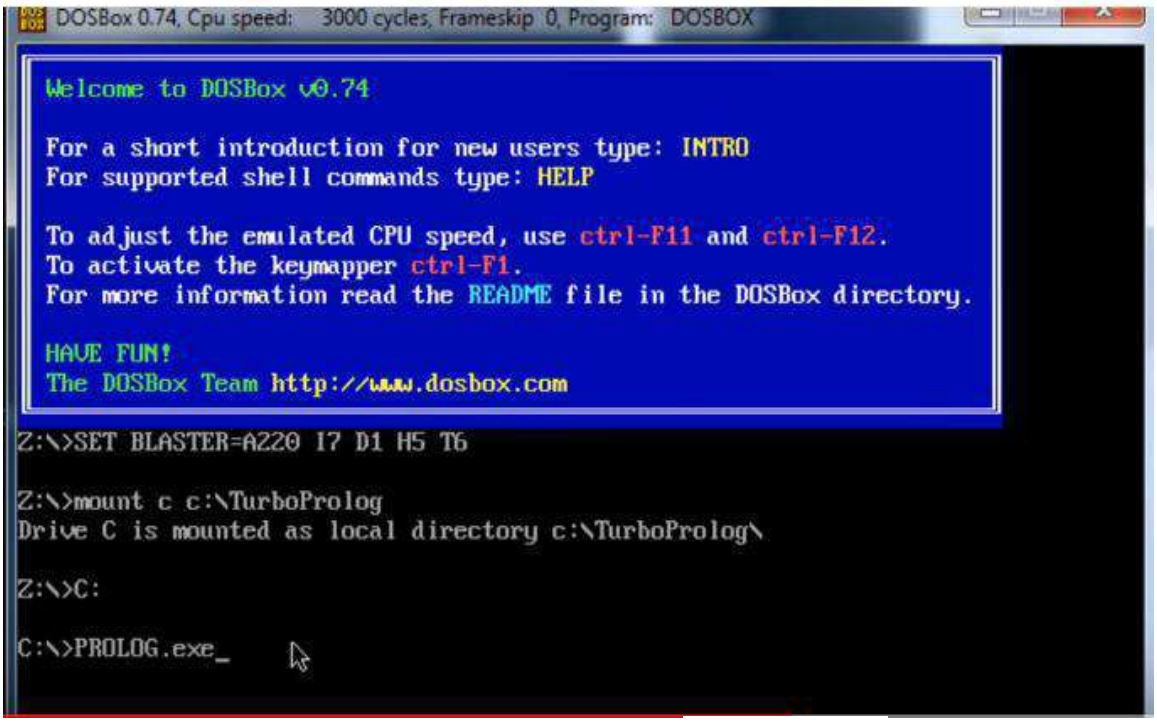


Рис. 3. Путь к папке Turbo Prolog

Используем клавишу Enter, и перед нами открывается рабочее окно программы Turbo Prolog 2.0 (см. рис. 4, рис. 5).

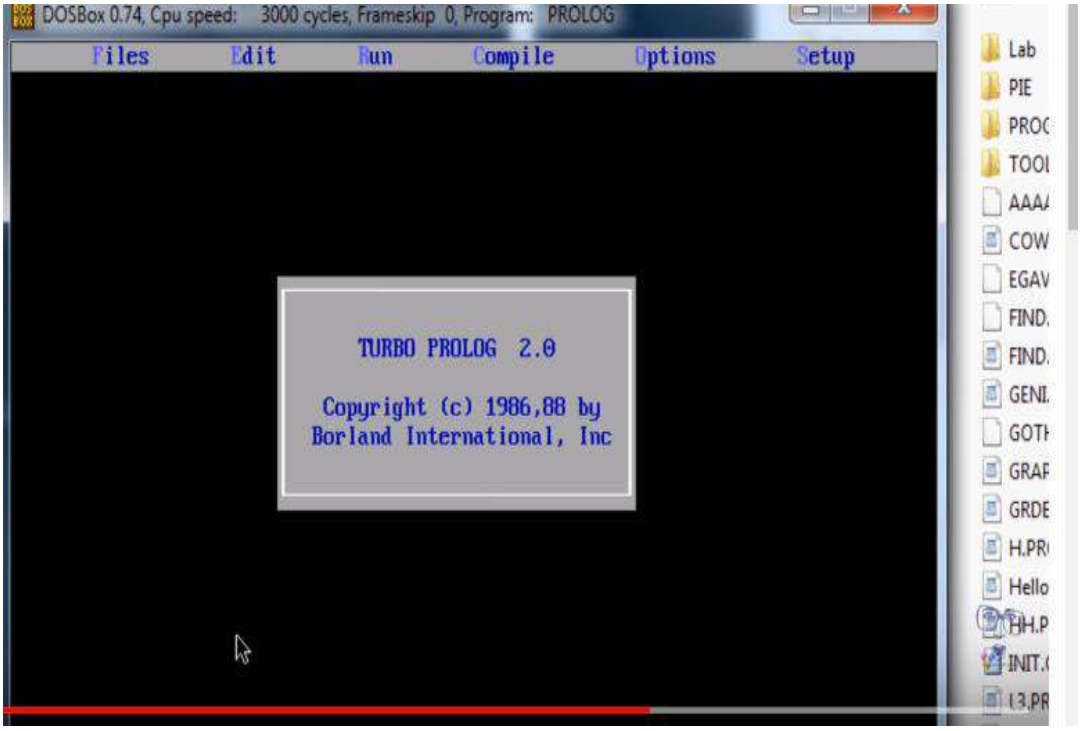


Рис. 4. Подтверждение запуска Turbo Prolog 2.0.

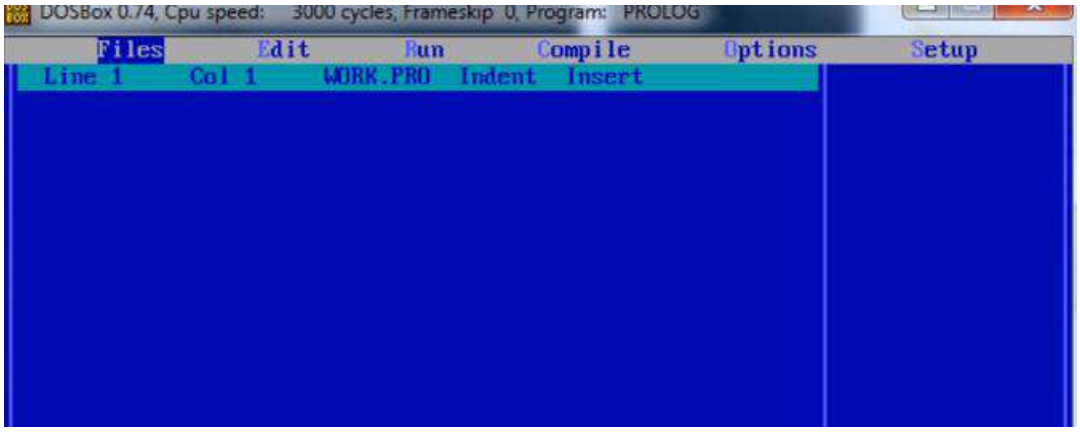


Рис. 5. Рабочее окно Turbo Prolog 2.0

Для удобства работы необходимо все созданные программы лабораторных работ, предварительно сохраненные с расширением **pro**, разместить в папке Turbo Prolog, т.к. при загрузке файлов с программой нужно будет прописывать путь до этого файла. Если же он будет находиться не на диске C, а например, на D, то не сможем получить до него доступ. В нашем случае, используя команду поиска **load** (см. рис.6), можно быстро получить доступ к нужному файлу.

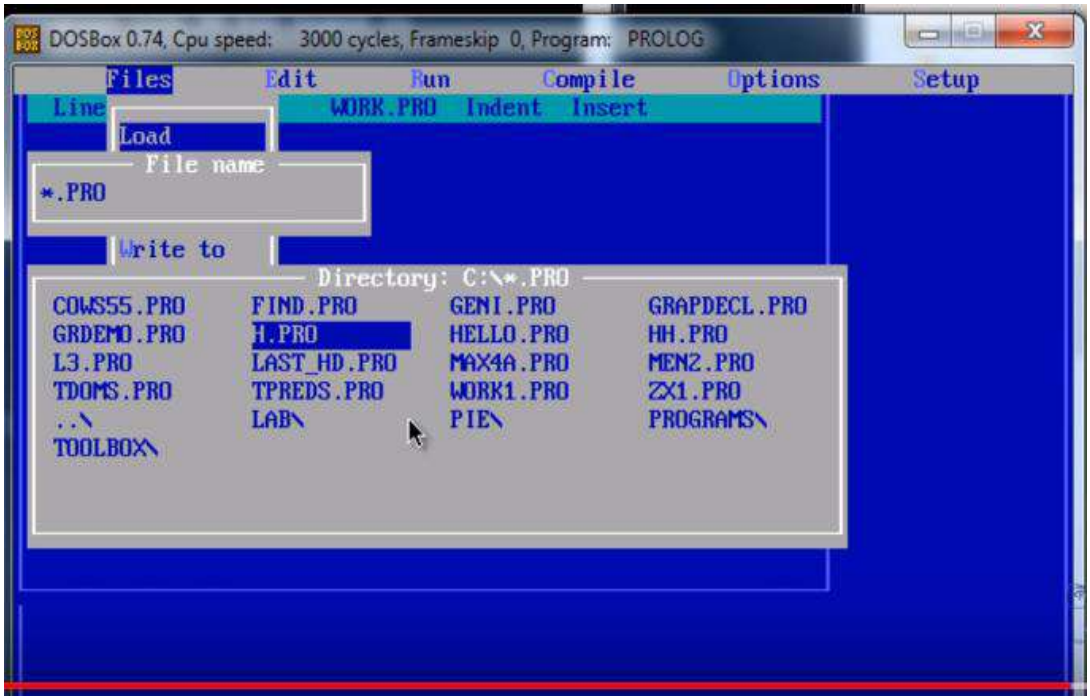


Рис. 6. Поиск нужного файла с программным кодом

УРОК-ПРАКТИКУМ № 2
Создание первой программы

Цель урока-практикума: отработка поэтапного создания простейшей программы на языке Пролог.

Рассмотрим подробное описание поэтапного создания программы для решения задачи «Brother» (брат), определяющей по условиям задачи кто чей брат, на языке Пролог. Большинство программ на прологе состоят из трех подпунктов-разделов: predicates (предикаты), clauses (классы), goal (вывод). В некоторых случаях вывод может отсутствовать, как, например, в программе «Brother». Для удобства работы с кодом программы создадим его в Notepad (блокнот), а затем пересохраним файл с расширением pro (см. урок-практикум № 1).

Предикаты – это неопределенное множество, в нашем понимании – это переменные, включающие в себя множества (но не обязательно). Например, предикат «parent», который содержит в себе две переменные, указывается в предикатах тип этой переменной, то есть «String».

Переменная пола (male) содержит один параметр. Так же предикат женщина (female) , тоже один параметр – это «String» , т.е. указываем строковую переменную. У параметра «Brother» тоже два параметра String, т.е. кто-то приходиться кому-то братом (см. рис.1).

Все вышеперечисленные предикаты должны использоваться в классах (clauses), т.к. может возникнуть синтаксическая ошибка.

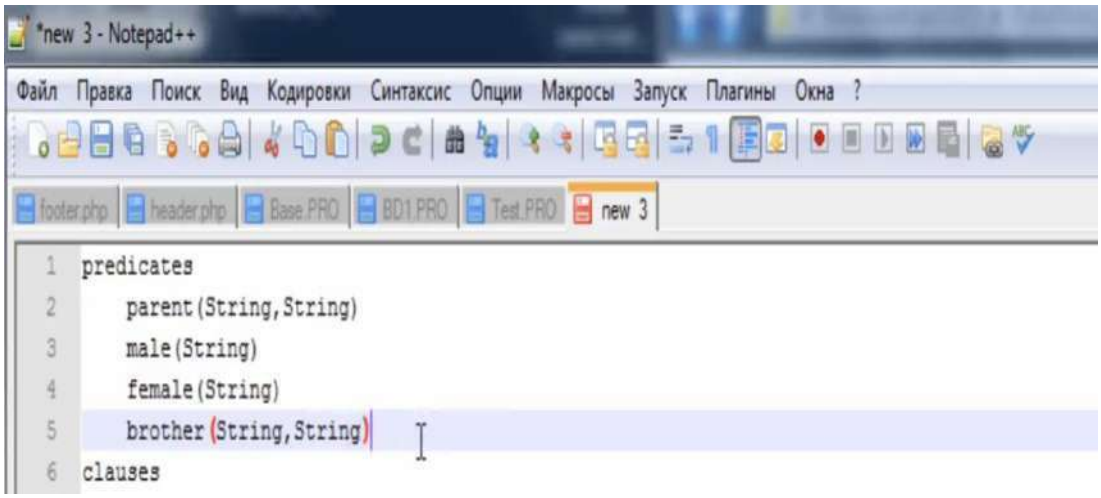


Рис. 1. Характеристика предикатов

Зададим в классах конкретное значение предикатов, т.е. родителей, например: Том родитель Джека и так далее. В конце каждой строки программы ставится точка, а не точка с запятой, как например в «C++».

Предикаты “parent”, “male”, ”female” известны программе TurboProlog’у, например, предикат «parent» – родитель читается как Том родитель Джейка, Джана тоже его родитель и так далее.

Предикаты «male», «female» тоже известны, и поэтому они задаются без каких либо проблем (см. рис.2).

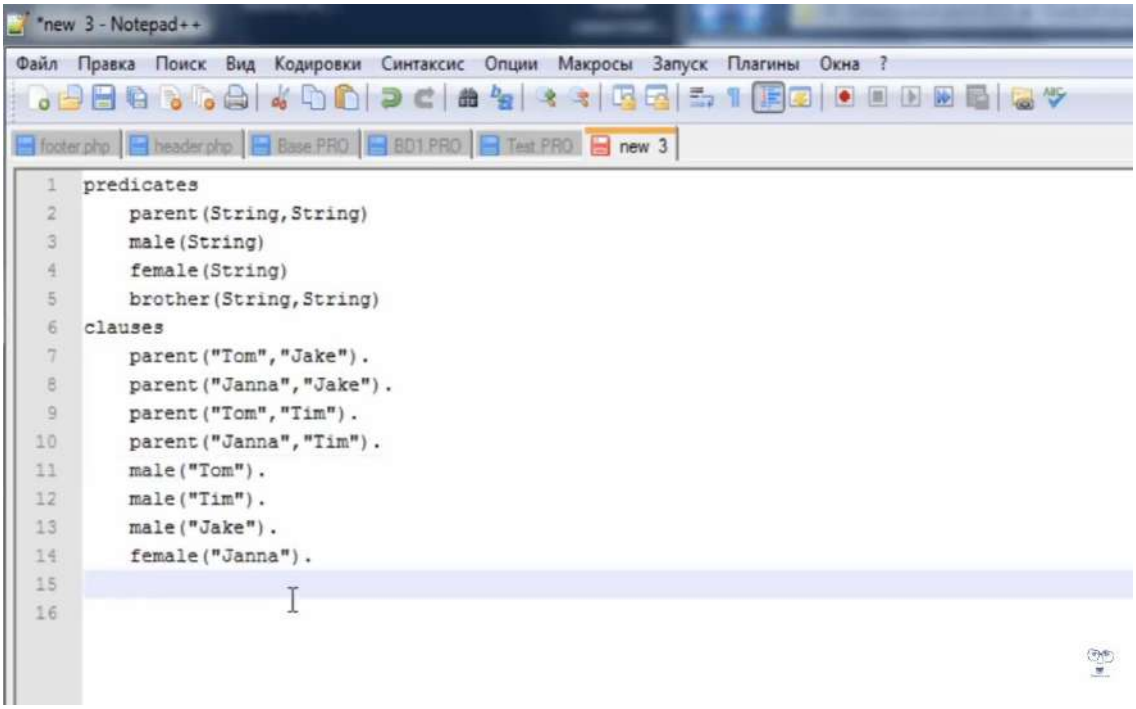


Рис. 2. Заполнение раздела clauses

Для определения предиката «brother» запишем следующую строчку, где задаем две неизвестных переменных (X,Y) , то есть X будет братом Y, ставим двоеточие и тире означающее команду если …то. Пишем условие: для того чтобы найти брата нужно задать общего родителя, т.е. у брата и сестры должен быть общий родитель, и брат X должен быть мужчиной, поэтому запишем «parent», введем переменную родителя Z (причем он должен быть родителем X); после ставим запятую, которая в прологе обозначает логическую операцию «И», далее «parent», где Z будет родителем Y (см. рис.3).

Далее запишем «Male», т.е. мужчина X, указываем, что «X<>Y» (что означает X не равно Y), т.к. брат не может быть братом сам себе. Эта распространенная ошибка в прологе, которая выводит, что человек сам себе брат.

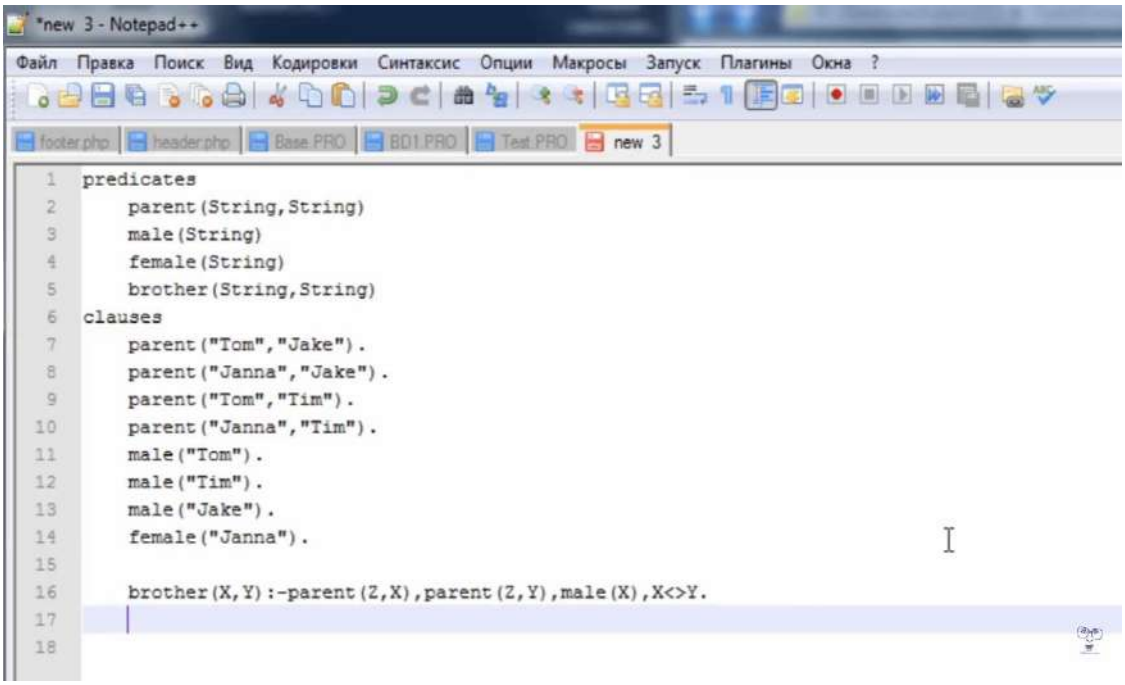


Рис. 3. Запись условия

Для вывода ответа на запрос в автоматическом режиме запишем следующую операцию «print», но изначально ее нужно определить в предикатах без параметров. Задаем переменную вывода: «brother» с параметрами X,Y. Далее пишем «write»(печать) и в этой строке задаем параметры X. После чего прописываем, что X брат Y. Ставим переход на следующую строку «nl» и «fail» (вывод до тех пор, пока не будет сообщения об ошибке, что братьев больше нет) (см. рис.4 строка 18).

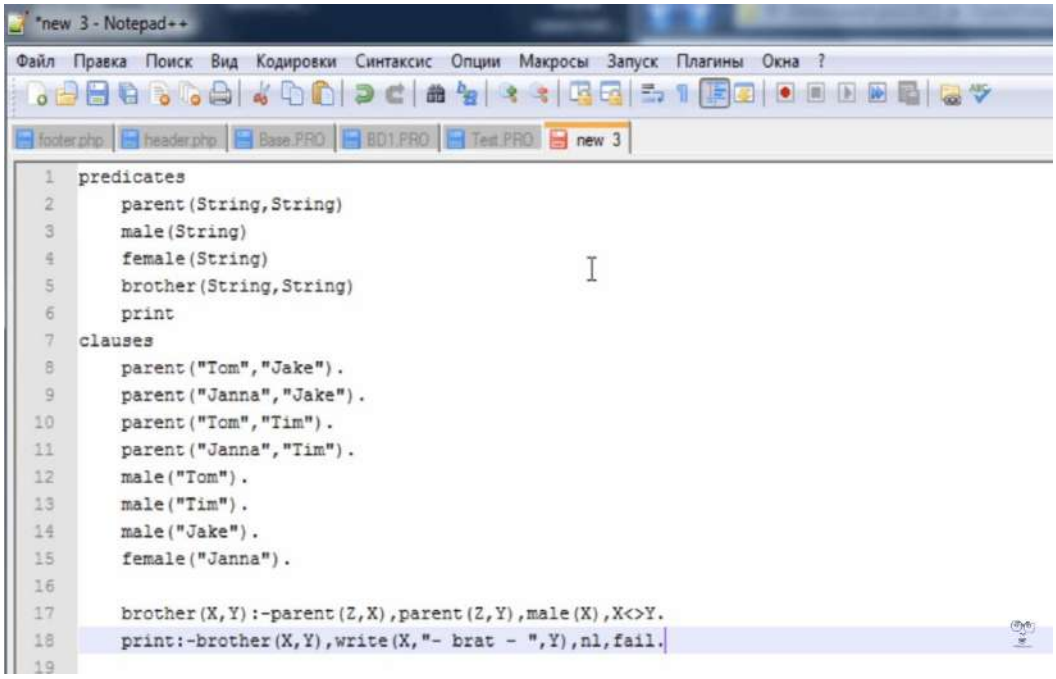


Рис. 4. Операция «print»

Программа об определении брата готова, теперь ее нужно сохранить в папке Пролога под именем «Brother» с расширением «PRO», т. к. с другим расширением программа работать не будет (см. рис. 5).

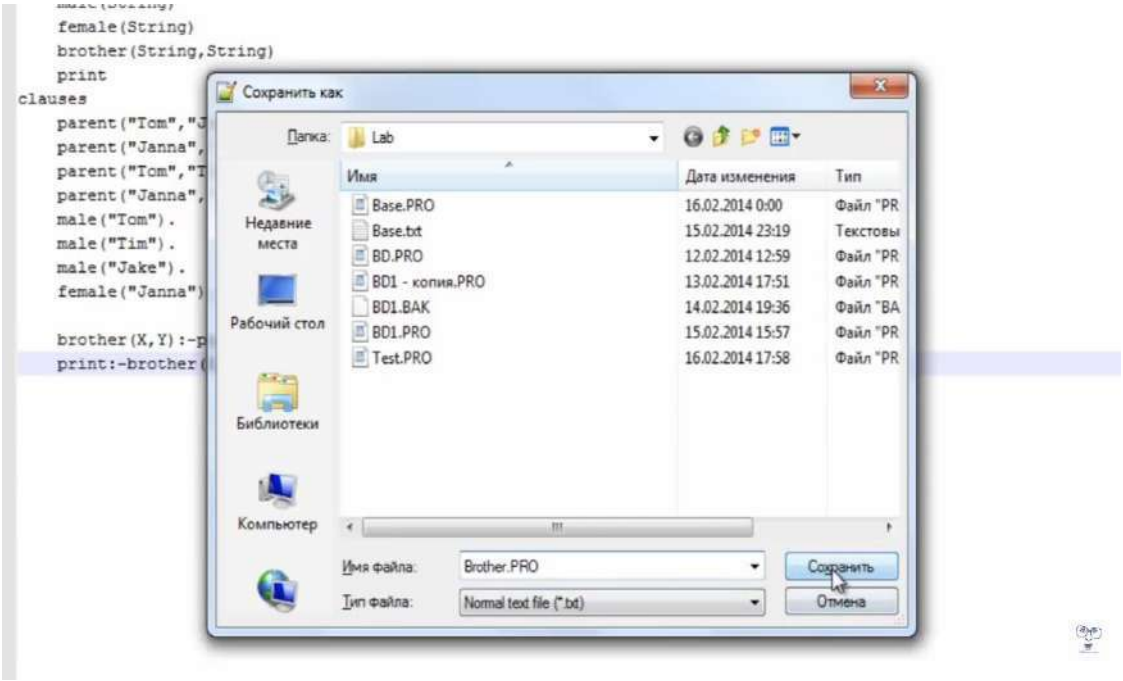


Рис. 5. Сохранение программы «Brother»

УРОК-ПРАКТИКУМ № 3
Компиляция и выполнение программы

Цель урока-практикума: открытие созданной программы решения задачи «Brother» в системе TurboProlog, выполнение компиляции, запуск программы.

Выполним поэтапно следующие действия: открываем «DOSBOX» (см. рис.1), т.к. последующие версии операционной системы Windows (после Windows XP) не поддерживают на прямую программу «Пролог», то будем использовать программу, написанную на языке «C++» и создающую DOS-окружение, необходимое для запуска старых программ.

Рис. 1. Открытие программы «DOSBOX»

В открывшемся окне требуется указать путь к программе пролог, для этого пропишем: mount c c:\TurboProlog. Нажимаем клавишу «Enter». Далее диск C, PROLOG.exe (см. рис.2.).

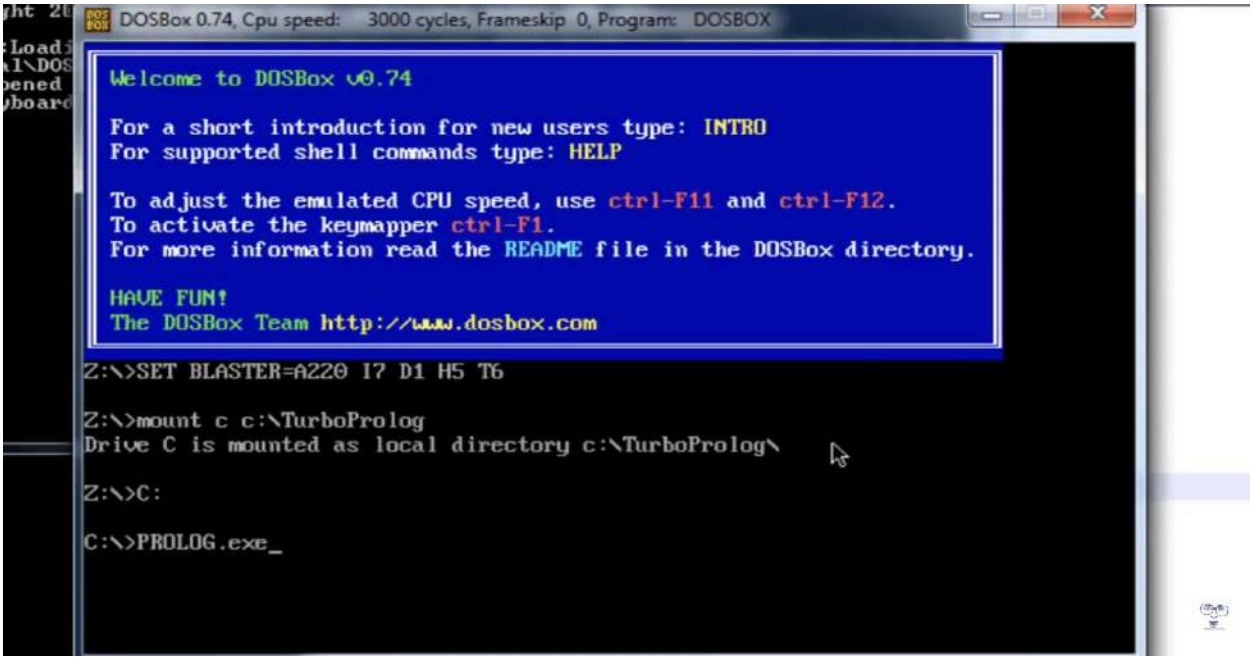


Рис. 2. Указание пути к программе Пролог

Заходим в контекстное меню Files\Load. Необходимо очистить поле ввода и задать свой адрес, т.е. адрес нашего файла: Brother.PRO (см. рис.3).

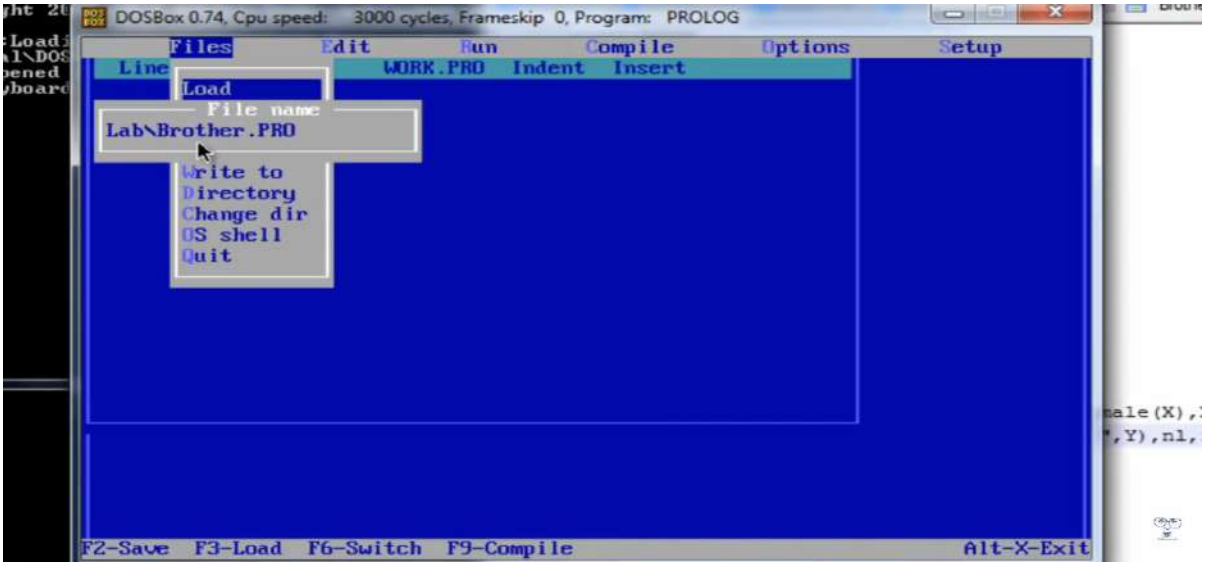


Рис. 3. Вид окна с адресной строкой

Далее для компиляции требуется нажать клавишу «F9». Фраза «Compilation successful» означает, компиляция прошла успешно и подтвердилась.

Затем переходим на вкладку «Run», нажимаем «Enter». Вводится «Goal», т.е. выбирается раздел ввод для выполнения запроса задачи (см. рис 4).

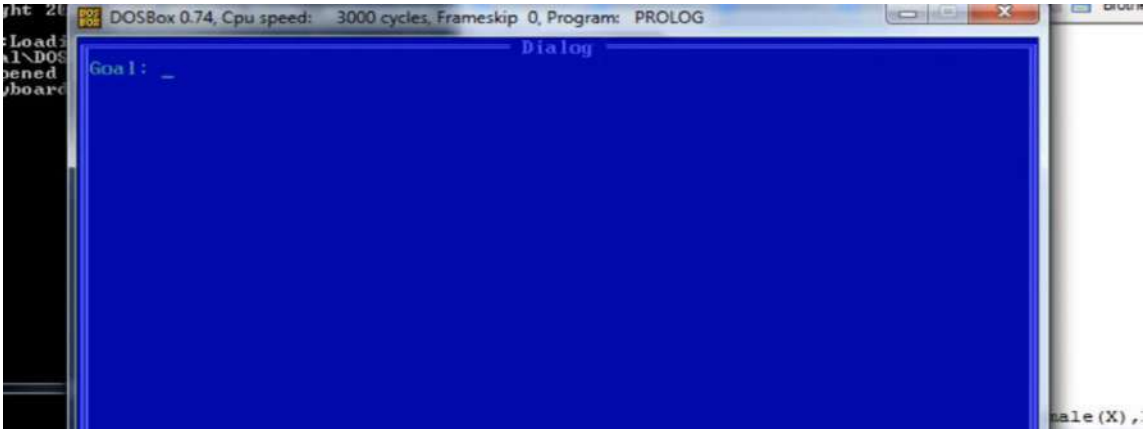


Рис. 4. Окно ввода

Для того чтобы вывести всех братьев задачи «Brother» используется операция «print», т.е. печать (см. рис. 5 строка 18).

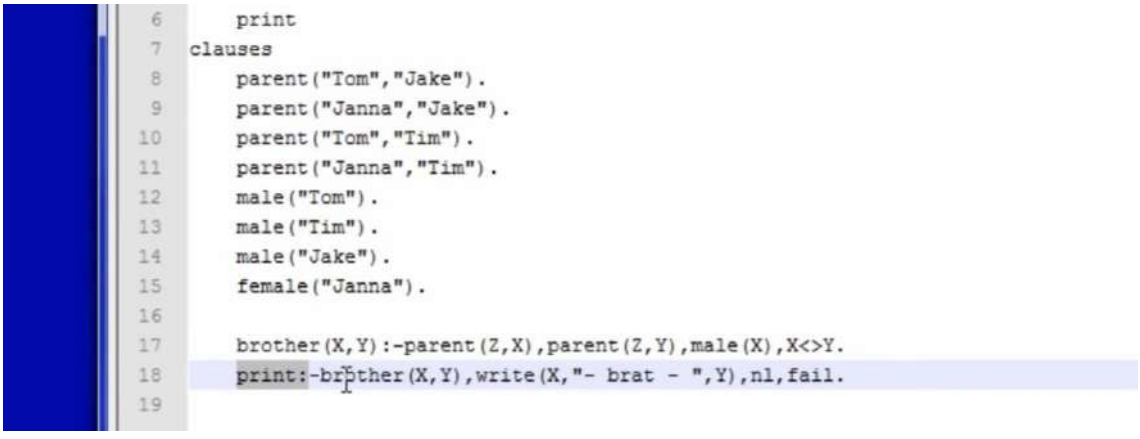


Рис. 5. Вопрос задачи «Brother»

Вводим эту операцию: пишем команду «print», получаем ответ, что «Джейк брат Тима» (см. рис. 6).

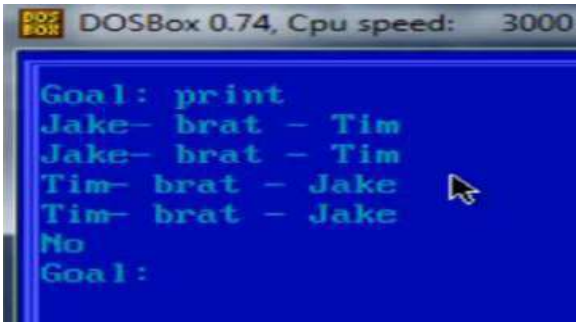


Рис. 6. Ответ к задаче «Brother»

Созданная программа в уроке 2, выполняющая решение задачи «Brother», правильно определила всех братьев с общими родителями.

ЛАБОРАТОРНАЯ РАБОТА № 1
Набор, редактирование и тестирование простейших программ в режиме Test Goal

Программа на Прологе состоит из предложений, которые могут быть фактами, правилами или запросами. Как правило, программа состоит из четырех разделов:

DOMAINS – раздел описания доменов(типов). Раздел применяется, если в программе используются нестандартные домены.

PREDICATES – раздел описания предикатов. Раздел применяется, если в программе используются нестандартные предикаты.

CLAUSES – раздел предложений. Именно в этом разделе записываются предложения: факты и правила вывода.

GOAL – раздел цели. В этом разделе записывается запрос.

Среда Visual Prolog позволяет протестировать программу без создания проекта. Для этого используется утилита Test Goal. Достаточно создать новый файл, набрать текст программы и активизировать Test Goal нажатием кнопки <G> на панели инструментов. Автономно исполняемый файл при этом не создается. Утилита Test Goal компилирует только тот код, который определен в активном окне редактора (код в других открытых окнах или модулях проектов, если они есть, игнорируются). Test Goal находит *все* возможные решения задачи и автоматически выводит значения *всех* переменных.

Пример 1. Имеется база данных, содержащая следующие факты:

родитель (илья, марина).
родитель (марина, ира).
родитель (елена, иван).
родитель (николай, ира).
родитель (ольга, алексей).
родитель (марина, саша).
родитель (сергей, иван).

Определить:

1) верно ли, что Марина является родителем Саши;

2) верно ли, что Марина является родителем Ивана;

3) кто является ребенком Николая;

4) кто родители Ивана;

5) всех родителей и их детей.

Решение:

1). Запустите среду Visual Prolog. Закройте окно проекта (если оно открыто) и откройте новый файл (**File|New**).

В появившемся окне наберите текст программы, содержащий разделы: PREDICATES (описание предиката ***родитель***), CLAUSES (перечисляются имеющиеся факты) и GOAL (запрос).

DOMAINS имя=string PREDICATES nondeterm родитель(имя, имя) CLAUSES родитель (илья, марина). родитель (марина, ира). родитель (елена, иван). родитель (николай, ира). родитель (ольга, алексей). родитель (марина, саша). родитель (сергей, иван). GOAL родитель(марина, саша) .	DOMAINS name=string PREDICATES nondeterm parent(name, name) CLAUSES parent(ilay, marina). parent(marina, ira). parent(elena, ivan). parent(nikolay, ira). parent(olga, aleksey). parent(marina, sasha). parent(sergey, ivan).

Запрос вводится отдельно в диалоговом окне (см. рис. 1).

Запустите и протестируйте программу с помощью команды **Project | Test Goal**(можно использовать кнопку на панели инструментов **<G>** или сочетание клавиш **<Ctrl>+<G>**). Результат выполнения программы будет выведен в отдельном окне

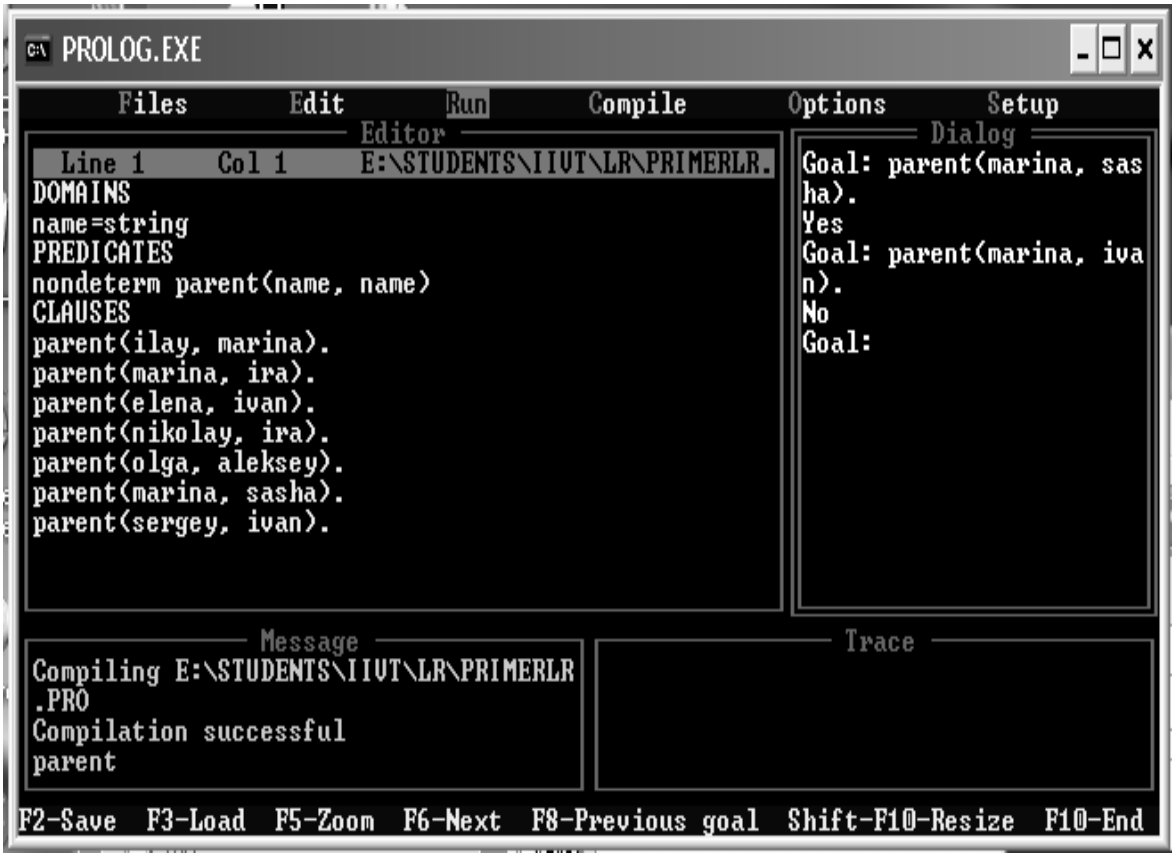


Рис. 1. Окно вывода результата

Указание: перед следующим запуском программы следует закрыть это окно.

2). Для ответа на вопрос: верно ли, что Марина является родителем Ивана, измените запрос:

GOAL

родитель(марина, иван).

После запуска программы (Project | Test Goal) будет получен ответ:

No (см. рис. 2 второй запрос)

3). Для ответа на вопрос: кто является ребенком Николая, запишите цель:

GOAL

родитель(николай, X) .

Результат (см. рис.2 третий запрос)**:**

X=ира

1 Solution

4). Для ответа на вопрос: кто родители Ивана, укажите запрос:

GOAL

родитель(X, иван), родитель(Y, иван), X<>Y.

Результат (см. рис.2 четвертый запрос)**:**

X=елена, Y=сергей

X=сергей, Y=елена

2 Solutions

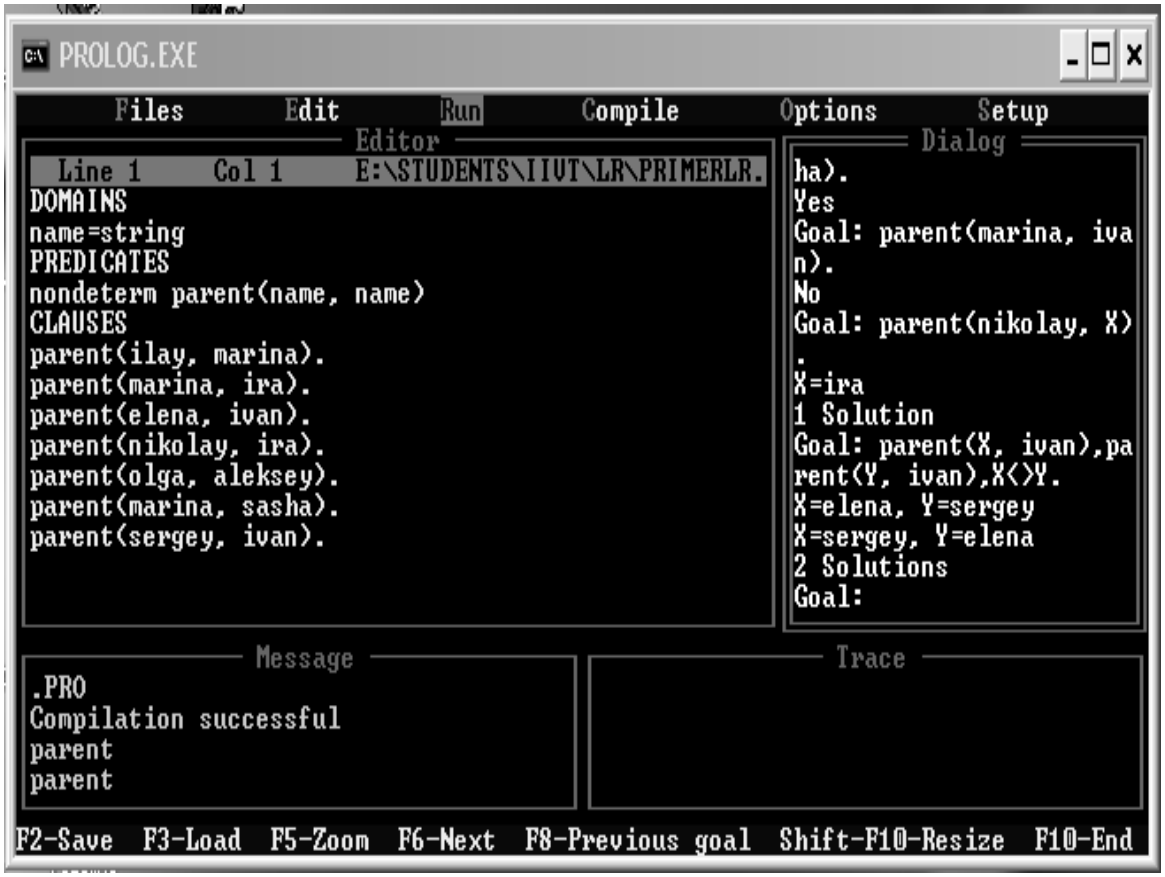


Рис. 2. Окно вывода результатов

5). Для определения всех родителей и их детей, запишите:
GOAL
родитель(X, Y).
Результат (см. рис.3 второй запрос)**:**
X=илья, Y=марина
X=марина, Y=ира
X=елена, Y=иван
X=николай, Y=ира
X=ольга, Y=алексей
X=марина, Y=саша
X=сергей, Y=иван
7 Solutions

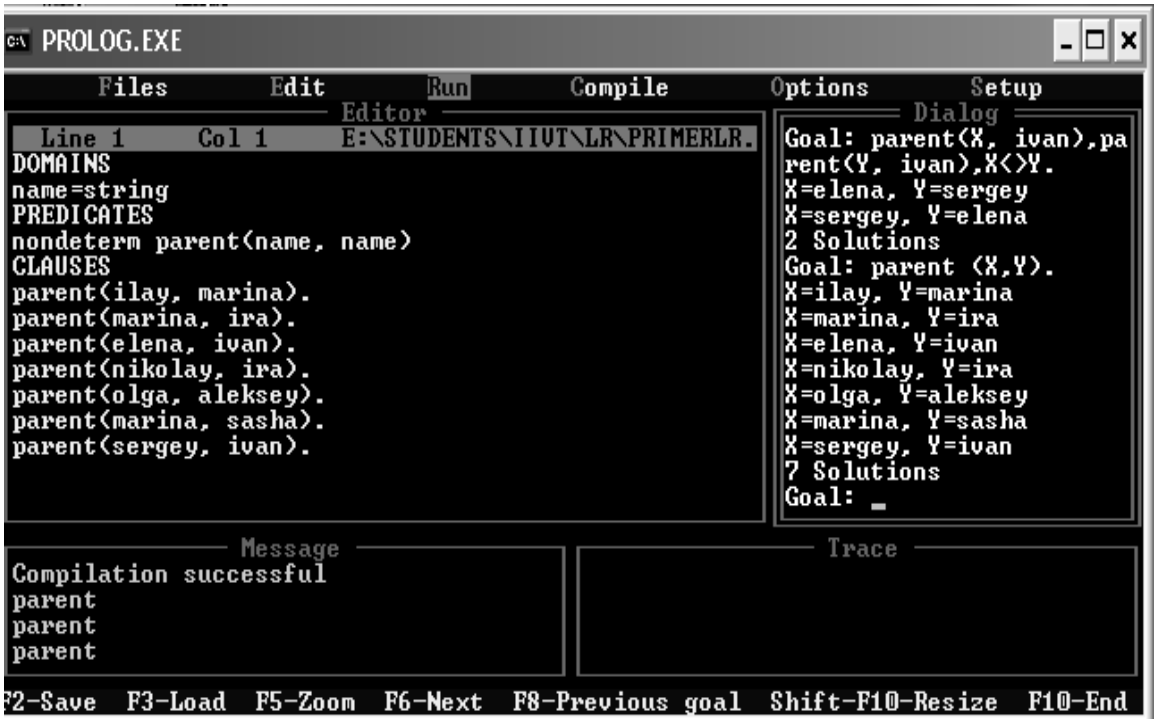


Рис. 3. Окно вывода результата

Пример 2. Имеются факты вида: *родитель(имя, имя)* и *женщина(имя)*.
а) составить правило ***мать*** и определить, кто мать Маши.
Решение:

```
DOMAINS
имя=string
PREDICATES
родитель(имя, имя)
женщина(имя)
мать(имя,имя)
CLAUSES
родитель("Марина","Ирина").
родитель("Елена", "Анна").
родитель("Ольга","Марина").
родитель("Ольга","Татьяна").
родитель("Татьяна","Катя").
родитель("Анна", "Маша").
женщина("Ольга").
женщина("Маша").
женщина("Ирина").
женщина("Елена").
женщина("Анна").
женщина("Марина").
женщина("Татьяна ").
женщина("Катя").
```

мать(X,Y):-родитель(X,Y),женщина(X).
GOAL
мать(X,"Маша").
Результат (см. рис.4)**:**
X=Анна
1 Solution

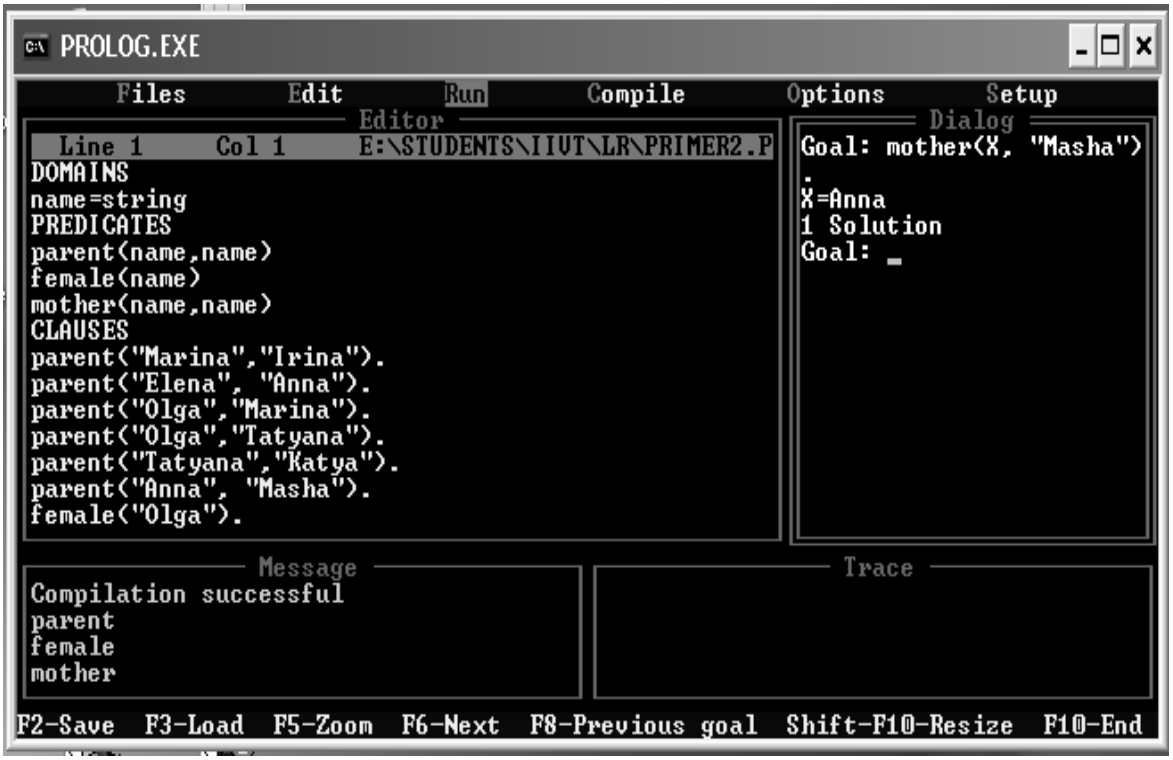


Рис. 4. Окно вывода результата

b) составить правило ***бабушка*** и определить, кто бабушка Маши.
Решение:
DOMAINS
имя=string
PREDICATES
nondeterm родитель(имя,имя)
женщина(имя)
nondeterm мать(имя,имя)
nondeterm бабушка(имя,имя)
CLAUSES
родитель("Марина","Ирина").
родитель ("Елена", "Анна").
родитель("Ольга","Марина").
родитель("Ольга","Татьяна").
родитель("Татьяна","Катя").
родитель ("Анна", "Маша").
женщина("Ольга").

женщина("Маша").
женщина("Ирина").
женщина("Елена").
женщина("Анна").
женщина("Марина").
женщина("Татьяна ").
женщина("Катя").
мать(X,Y):-родитель(X,Y),женщина(X).
бабушка(X,Z):-мать(X,Y),родитель(Y,Z).
GOAL
бабушка(X,"Маша").
Результат (см. рис. 5)**:**
X=Елена
1 Solution

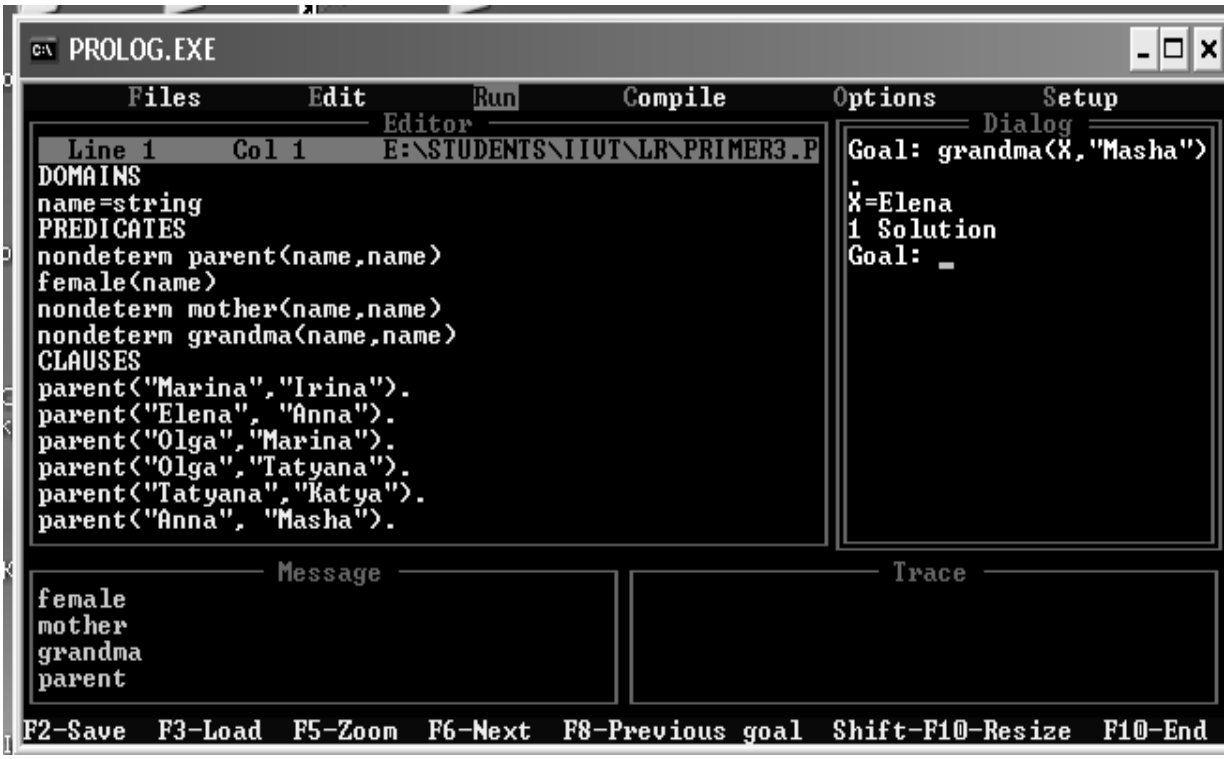


Рис. 5. Окно вывода результата

Замечание: ключевое слово *nondeterm* определяет недетерминированные предикаты, которые могут совершать откат назад и генерировать множественные решения. Таким образом, если задача предполагает возможность получения несколько решений, следует объявлять предикаты как недетерминированные.

c) составить правило ***внучка*** и определить, сколько внучек у Ольги и как их зовут.

Решение:

DOMAINS имя=string PREDICATES nondeterm родитель(имя,имя) женщина(имя) nondeterm мать(имя,имя) nondeterm бабушка(имя,имя) nondeterm внучка(имя,имя) CLAUSES родитель("Марина","Ирина"). родитель ("Елена", "Анна"). родитель("Ольга","Марина"). родитель("Ольга","Татьяна"). родитель("Татьяна","Катя"). родитель("Анна", "Маша"). женщина("Ольга"). женщина("Маша"). женщина("Ирина"). женщина("Елена"). женщина("Анна"). женщина("Марина"). женщина("Татьяна "). женщина("Катя"). мать(X,Y):- родитель(X,Y),женщина(X). бабушка(X,Z):- мать(X,Y),родитель(Y,Z). внучка(X,Y):- бабушка(Y,X),женщина(X). GOAL внучка(X, "Ольга").	DOMAINS name=string PREDICATES nondeterm parent(name,name) female(name) nondeterm mother(name,name) nondeterm grandma(name,name) nondeterm vnuchka(name,name) CLAUSES parent("Marina","Irina"). parent("Elena", "Anna"). parent("Olga","Marina"). parent("Olga","Tatyana"). parent("Tatyana","Katya"). parent("Anna", "Masha"). female("Olga"). female("Masha"). female("Irina"). female("Elena"). female("Anna"). female("Marina"). female("Tatyana"). female("Katya"). mother(X,Y):- parent(X,Y),female(X). grandma(X,Z):- mother(X,Y),parent(Y,Z). vnuchka(X,Y):- grandma(Y,X),female(X).

Запрос вводится отдельно в диалоговом окне (см. рис 6).

Результат (см. рис. 6)**:**

X=Ирина

X=Катя

2 Solutions

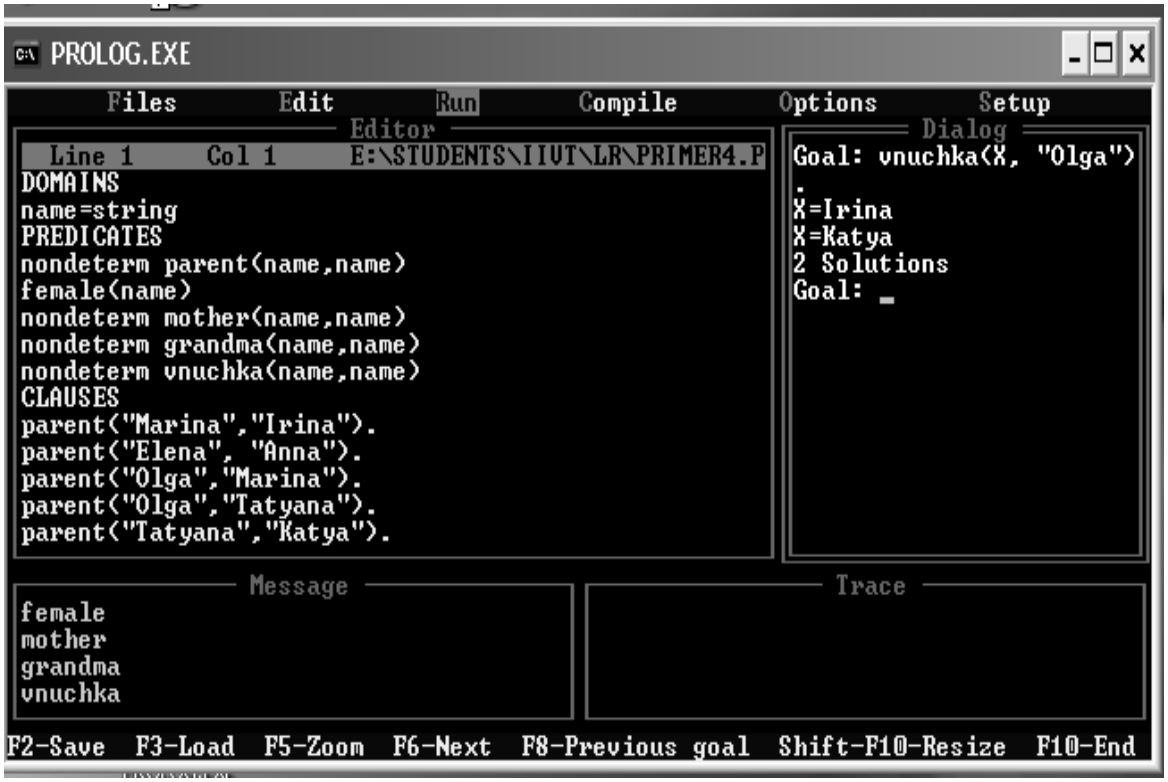


Рис. 6. Окно вывода результата

<u>Задания для самостоятельной работы</u>

1. Имеется база данных, содержащая следующие факты:
любит(“Анна,” яблоки).
любит(“Сергей”, бананы).
любит(“Андрей”, яблоки).
любит(“Света”, шоколад).
любит(“Вова”, шоколад).
любит(“Анна”, шоколад).
любит(“Света”, апельсины).
любит(“Вова”, бананы).
Составить программу, определяющую:
a) всех, кто любит бананы;
b) кто любит и шоколад, и яблоки;
c) что любит Вова;
d) что любят и Света, и Вова.
2. Имеется база данных, содержащая следующие факты:
играет (“Саша”, футбол).
играет (“Катя”, теннис).
играет (“Саша”, теннис).
играет (“Андрей”, футбол).
играет (“Олег”, футбол).
играет (“Ольга”, теннис).
играет (“Катя”, волейбол).
играет (“Олег”, волейбол).

Составить программу, определяющую:

a) каким видом спорта увлекается Андрей;

b) всех, кто играет в волейбол;

c) каким видом спорта увлекаются и Ольга, и Саша;

d) кто увлекается и футболом, и волейболом.

3. Имеется база данных, содержащая следующие факты:

любит(“Анна,” яблоки).

любит(“Сергей”, бананы).

любит(“Андрей”, яблоки).

любит(“Света”, шоколад).

любит(“Вова”, шоколад).

любит(“Анна”, шоколад).

любит(“Света”, апельсины).

любит(“Вова”, бананы).

фрукты(яблоки).

фрукты(бананы).

фрукты(апельсины).

конфеты(шоколад).

a) используя имеющиеся факты, составить новое правило ***люб_фрукты****(X)* и определить всех, кто любит фрукты;

b) используя имеющиеся факты, составить новое правило ***люб_конфеты****(X)* и определить всех, кто любит конфеты;

c) используя имеющиеся факты, составить правило ***люб_вкусное****(X*) и определить всех, кто любит и фрукты, и конфеты.

4. Имеется база данных, содержащая следующие факты:

играет (“Саша”, футбол).

играет (“Катя”, теннис).

играет (“Саша”, теннис).

играет (“Андрей”, футбол).

играет (“Олег”, футбол).

играет (“Ольга”, теннис).

играет (“Катя”, волейбол).

играет (“Олег”, волейбол).

женщина(“Катя”).

женщина(“Ольга”).

мужчина(“Саша”).

мужчина(“Андрей”).

мужчина(“Олег”).

a) используя имеющиеся факты, составить новое правило ***волей-бол_жен(X)*** и определить всех женщин, играющих в волейбол;

b) используя имеющиеся факты, составить новое правило ***фут-бол_муж(X)*** и определить всех мужчин, играющих в футбол;

с) используя имеющиеся факты, составить правило ***теннис_пара(X,Y),*** позволяющее найти смешанную теннисную пару (мужчина+женщина). Определить все такие пары.

Отчет о выполненной самостоятельной работе должен содержать:

1) тему лабораторной работы;
2) условие задачи;
3) листинг программы;
4) результаты ее тестирования.

ЛАБОРАТОРНАЯ РАБОТА № 2
Создание простейших проектов

Создание проекта позволяет протестировать пример как автономную исполняемую программу. После запуска проекта на исполнение создается exe-файл, работа которого завершается после *первого* решения, удовлетворяющего решению задачи. Запуск программы в этом режиме не обеспечивает автоматический вывод значений переменных, поэтому необходимо использовать стандартный предикат вывода **write.**

Пример. Заданы отношения-факты:

родитель(“Иван”,”Катя”).
родитель(“Анна”,” Олег ”).
родитель(“Олег”,”Дима”).
родитель(“Игорь”,”Ольга”).
родитель(“Олег”,”Виктор”).
родитель(“Игорь”,”Иван”).
мужчина(“Дима”).
мужчина(“Иван”).
мужчина(“Игорь”).
мужчина(“Олег”).
мужчина(“Виктор”).
женщина(“Катя”).
женщина(“Ольга”).
женщина(“Анна”).

Составить новое отношение-правило ***дед(X,Y)*** и определить, кто является дедушкой Кати. Создать проект и протестировать пример как автономную исполняемую программу.

Решение:

1. Запустите среду Visual Prolog и создайте новый проект (Project | New Project), активизируется окно **Application Expert** (эксперт приложения).

2. Определите имя проекта (Primer) и базовый каталог, куда будет сохранен проект (например, D:\VP\Primer)

Файл с расширением .pro содержит секции PREDICATES, GOAL, CLAUSES. Допишите необходимые определения так, чтобы получилась программа:

DOMAINS имя=string PREDICATES родитель(имя,имя) женщина(имя) мужчина(имя) дед(имя, имя) CLAUSES родитель("Иван","Катя"). родитель("Анна","Олег"). родитель("Олег","Дима"). родитель("Игорь","Ольга"). родитель("Олег","Виктор"). родитель("Игорь","Иван"). мужчина("Дима"). мужчина("Иван"). мужчина("Игорь"). мужчина("Олег"). мужчина("Виктор"). женщина("Катя"). женщина("Ольга"). женщина("Анна"). дед(X,Z):-родитель(X,Y), родитель(Y,Z), мужчина(X). GOAL дед(X,"Катя"),write(X).	DOMAINS name=string PREDICATES parent(name,name) female(name) male(name) grandpa(name,name) CLAUSES parent("Ivan","Katya"). parent("Anna","Oleg"). parent("Oleg","Dima"). parent("Igor","Olga"). parent("Oleg","Victor"). parent("Igor","Ivan"). male("Dima"). male("Ivan"). male("Igor"). male("Oleg"). male("Victor"). female("Katya"). female("Olga"). female("Anna"). grandpa(X,Z):-parent(X,Y), parent(Y,Z), male(X).

Запрос вводится отдельно в диалоговом окне (см. рис 1).

3. Откомпилируйте исходный код примера и запустите его как автономную исполняемую программу. (Project | Run, или клавиша <F9>, или кнопка <R>). Результат выполнения программы должен отобразиться в окне (см. рис.1):

Рис. 1. Окно вывода результата

Задания для самостоятельной работы

Доработайте исходный код примера следующим образом:

1) добавьте новое правило ***бабушка*** и определите, кто является бабушкой;

2) добавьте новое правило ***внук*** и определите, кто внук Анны;

3) добавьте новое правило ***брат*** и определите, кто брат Димы;

4) добавьте новое правило ***сестра*** и определите, кто сестра Ивана.

Отчет о выполненной самостоятельной работе должен содержать:

1) тему лабораторной работы;

2) условие задачи;

3) полный окончательный код примера;

4) результаты ее тестирования.

ЛАБОРАТОРНАЯ РАБОТА № 3
Поиск с возвратом

Поиск с возвратом (backtracking) – это один из основных приемов поиска решений поставленной задачи в ПРОЛОГ’е. Выполняя поиск, ПРОЛОГ может столкнуться с необходимостью выбора между альтернативными путями. Тогда он ставит маркер у места развилки (точка отката) и выбирает первую подцель. Если она не выполняется, то ПРОЛОГ возвращается в точку отката и переходит к следующей подцели.

Среда Visual Prolog позволяет использовать отладчик для пошагового выполнения программы. Отладчик работает с откомпилированным кодом. В исходном коде можно ставить точки останова и выполнять программу по ша-

гам. В режиме пошагового выполнения программы можно просматривать значения переменных и содержимое утвержденных фактов.

Пример. Имеется база данных, содержащая факты вида ***отдыхает(имя, город)***, ***украина(город***), ***россия(город),прибалтика(город).*** Составить правило, позволяющее определить, кто отдыхал в России.

Проследить поиск решения задачи с помощью отладчика Visual Prolog и построить целевое дерево поиска с возвратом.

Решение:

1. Создайте новый проект (Project | New Project) и наберите текст программы:

```
DOMAINS
имя, город=string
PREDICATES
отдыхает(имя, город)
украина(город).
россия(город).
прибалтика(город).
отдых_Россия(имя)
CLAUSES
отдыхает(sasha, antalia).
отдыхает(anna, sochi).
отдыхает(dima, urmala).
отдыхает(oleg, kiev).
украина(kiev).
россия(sochi).
прибалтика(urmala).
отдых_Россия(X):-
отдыхает(X,Y),
россия(Y).
GOAL
отдых_Россия(X),
write(X),nl.
```

```
DOMAINS
name, gorod=string
PREDICATES
otdixaet(name, gorod)
ykraina(gorod).
rossia(gorod).
pribaltika(gorod).
otdix_rossia(name)
CLAUSES
otdixaet(sasha, antalia).
otdixaet(anna, sochi).
otdixaet(dima, urmala).
otdixaet(oleg, kiev).
ykraina(kiev).
rossia(sochi).
pribaltika(urmala).
otdix_rossia(X):-
otdixaet(X,Y),
rossia(Y).
```

Запрос вводится отдельно в диалоговом окне (см.рис 1).

2. Сохраните проект (**Project | Save Project**)

3. Запустите его на исполнение (**Project | Run**, или клавиша **<F9>,** или кнопка **<R>**). Результат выполнения программы:

Anna

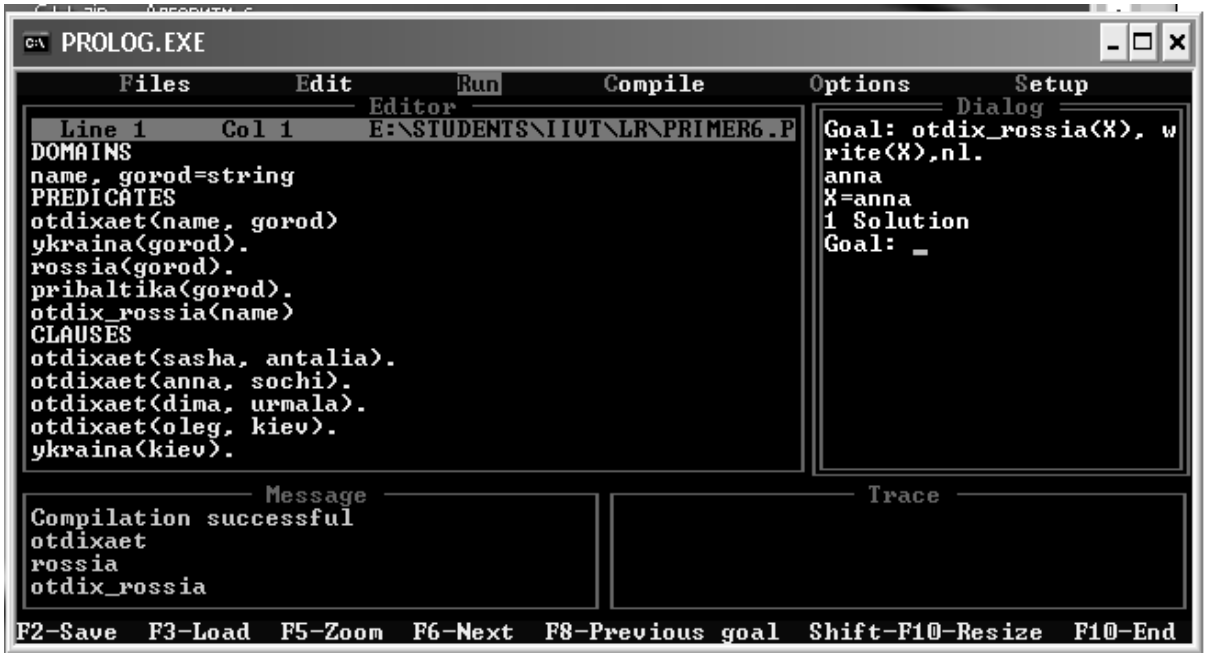


Рис. 1. Окно вывода результата

4. Проследите поиск этого решения с помощью отладчика(Debugger). Для этого:

а) запустите отладчик (**Project | Debug**);

б) в окне отладчика выберите команду **View | Local Variables** (для просмотра текущих значений переменных);

в) нажимайте клавишу **<F7>**(или **Run | Trace Into**) для пошагового выполнения программы, текущие значения переменных отображаются в окне Variables For Current Clause.

<u>Задания для самостоятельной работы</u>

1. База данных содержит следующие факты:

увлекается(“Коля”, гитара).
увлекается(“Оля”, скрипка).
увлекается(“Дима”, плаванье).
увлекается(“Таня”, теннис).
спорт(плаванье).
спорт(теннис).
муз_инстр(скрипка).
муз_инстр(гитара).

а) составить правило спортсмен и определить, кто увлекается спортом;

б) проследить за поиском решения с помощью отладчика;

в) построить целевое дерево поиска с возвратом.

<u>Отчет о выполненной самостоятельной работе должен содержать:</u>

1) тему лабораторной работы;
2) условие задачи;
3) листинг программы;
4) результаты ее тестирования;
5) целевое дерево поиска решения.

ЛАБОРАТОРНАЯ РАБОТА № 4
Управление поиском с возвратом: предикаты fail и отсечения

fail – это тождественно-ложный предикат, искусственно создающий ситуацию неуспеха. После выполнения этого предиката управление передается в точку отката и поиск продолжается. Использование предиката fail позволяет найти все решения задачи.

Чтобы ограничить пространство поиска и прервать поиск решений при выполнении какого-либо условия, используется предикат *отсечения* (обозначается **!**), Однажды пройдя через отсечение, невозможно вернуться назад, т.к. этот предикат является тождественно-истинным. Процесс может только перейти к следующей подцели, если такая имеется.

Например, p :- p1, p2, !, p3.

Если достигнуты цели p1 и p2, то возврат к ним для поиска новых решений невозможен.

Пример 1. База данных содержит факты вида: ***student(имя, курс).*** Создать проект, позволяющий сформировать список студентов 1-го курса.

Решение:

```
PREDICATES
student(symbol,integer)
spisok
CLAUSES
student(vova,3).
student(lena,1).
student(dima,1).
student(ira,2).
student(marina,1).
spisok:-student(X,1),write(X),nl,fail.
GOAL
write("Список студентов 1-курса"),nl,spisok.
```

Результат выполнения программы:

```
Список студентов 1-курса
lena
dima
marina
```

Комментарий: предлагается самостоятельно вывести программу и ее результат на экран

Пример 2. База данных содержит факты вида **father(name, name).** Создать проект, позволяющий определить кто чей отец.

Решение:

DOMAINS name=symbol PREDICATES father (name, name) everybody CLAUSES father ("Павел", "Петр"). father ("Петр", "Михаил"). father ("Петр", "Иван"). everybody:- father (X, Y), write(X," - это отец",Y,"a"),nl, fail. GOAL everybody.	DOMAINS name=symbol PREDICATES father (name, name) everybody CLAUSES father ("Pavel", "Petr"). father ("Petr", "Mixail"). father ("Petr", "Ivan"). everybody:- father (X, Y), write(X," - it is the father",Y,"a"),nl, fail.

Результат выполнения программы (см. рис. 1):

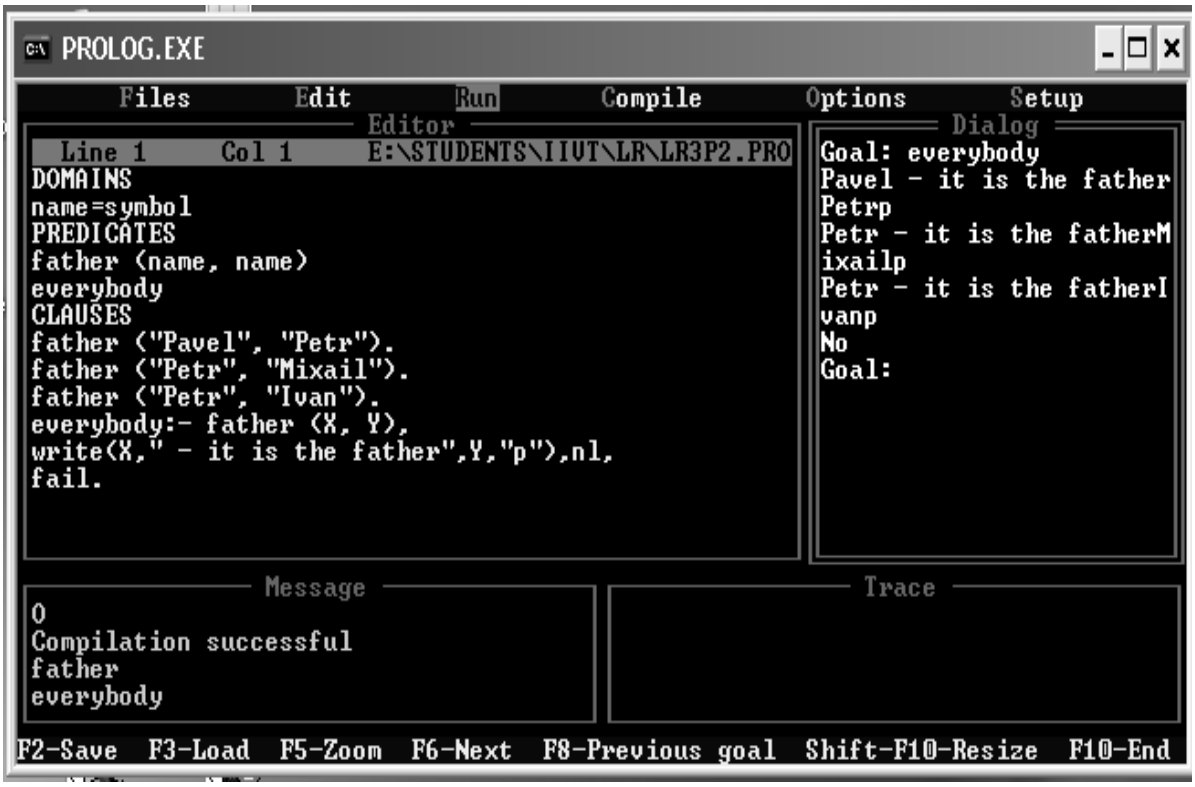


Рис. 1. Результат программы

Павел – это отец Петра
Петр – это отец Михаила
Петр – это отец Ивана

Пример 3. Создать проект, реализующий железнодорожный справочник. В справочнике содержится следующая информация о каждом поезде: номер поезда, пункт назначения и время отправления.

а) вывести всю информацию из справочника.

Решение:

DOMAINS nom=integer p, t=string PREDICATES poezd(nom,p,t) CLAUSES poezd(233,moskva,"12-30"). poezd(257,moskva,"22-40"). poezd(133,armavir,"10-20"). poezd(353,armavir,"20-40"). poezd(353,adler,"02-30"). poezd(413,adler,"11-10"). poezd(256,piter,"21-30"). GOAL write(" Расписание поездов"), nl, write("Номер Пункт прибытия Время отправления"), nl, poezd(N,P,T), write(N," ",P," ",T),nl,fail.	DOMAINS nom=integer p, t=string PREDICATES poezd(nom,p,t) CLAUSES poezd(233,moskva,"12-30"). poezd(257,moskva,"22-40"). poezd(133,armavir,"10-20"). poezd(353,armavir,"20-40"). poezd(353,adler,"02-30"). poezd(413,adler,"11-10"). poezd(256,piter,"21-30"). GOAL Write("train schedule:"),nl, write("Number Point of arrival Departure time"), nl, poezd(N,P,T), write(N," ",P," ",T),nl,fail.

Результат выполнения программы (см. рис. 2):

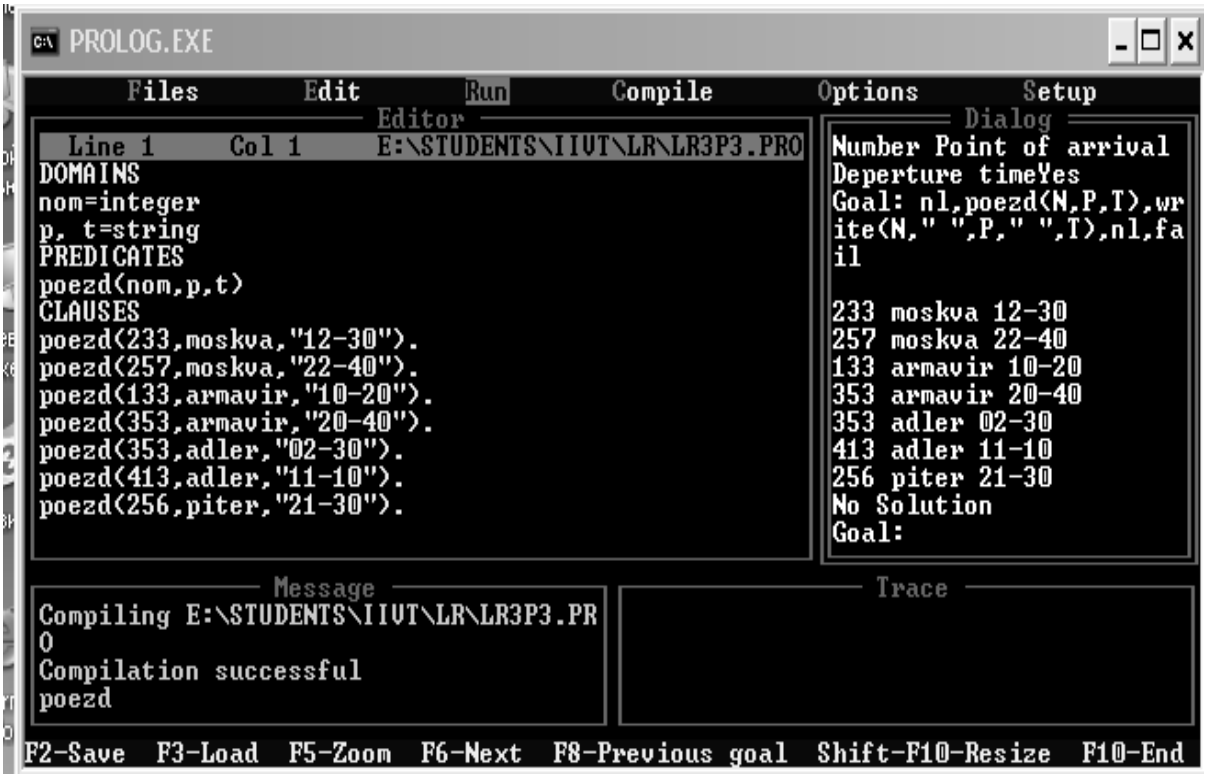


Рис. 2. Результат программы

Расписание поездов
Номер Пункт прибытия Время отправления
233 moskva 12-30
257 moskva 22-40
133 armavir 10-20
353 armavir 20-40
353 adler 02-30
413 adler 11-10
256 piter 21-30

б) организовать поиск поезда по пункту назначения

Решение:

```
GOAL
write (" Пункт назначения:"), Readln(P), nl,
write ("Номер Время отправления"), nl,
poezd(N,P,T), write(N," ",T), nl, fail.
```

Комментарий: Readln –стандартный предикат ввода строкового значения

Результат выполнения программы (см. рис. 3):

Рис. 3. Результат программы

Пункт назначения: armavir
Номер Время отправления
133 10-20
353 20-40

в) вывести информацию о поездах, отправляющихся в заданный временной промежуток

Решение:

GOAL write(" Время отправления:"),nl, write("с..."), Readln(T1), write("до..."), Readln(T2), nl, write("Номер Пункт назначения Время отправления"), nl,poezd(N,P,T),T>=T1,T<=T2,write(N," ",P," ", T), nl, fail.	GOAL write("Departure time:"),nl, write("c..."), Readln(T1), write("do..."), Readln(T2), nl, write("Number Destination Departure time"), nl,poezd(N,P,T),T>=T1,T<=T2,write(N," ",P," ", T), nl, fail.

Результат выполнения программы (см. рис. 4):

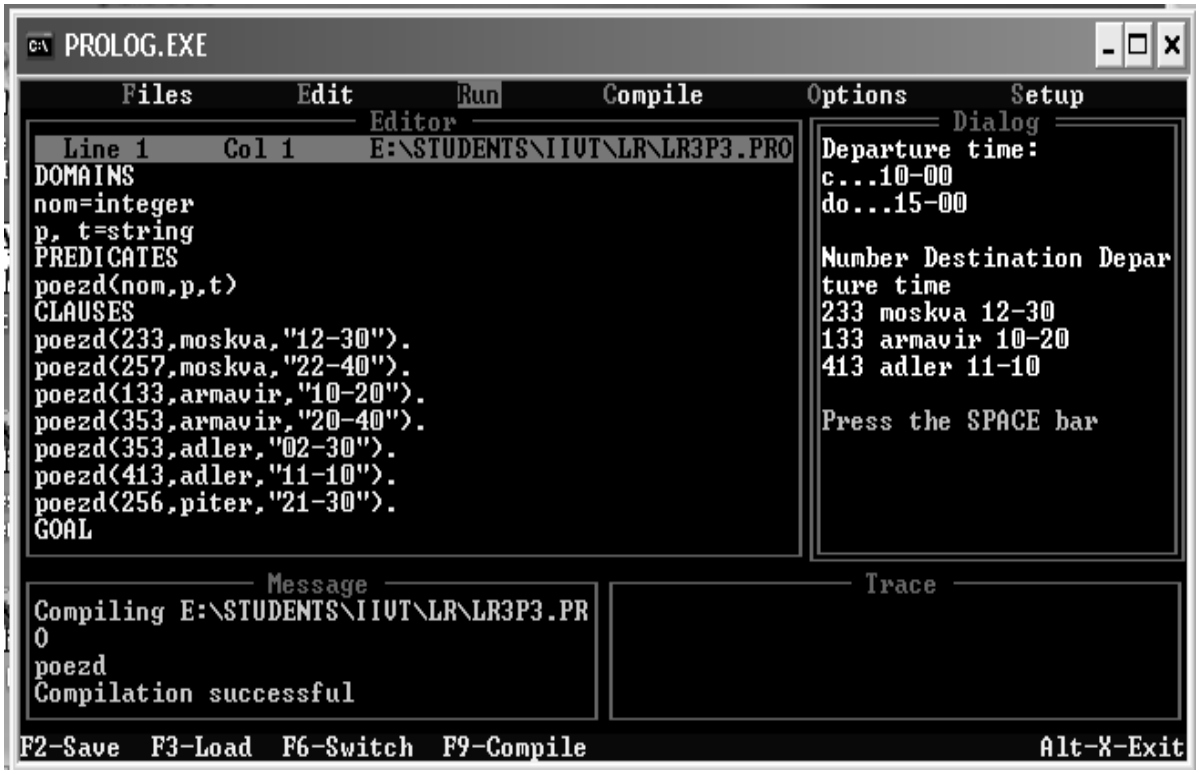


Рис. 4. Результат программы

Время отправления:
с...10-00
до...15-00
Номер Пункт назначения Время отправления
233 moskva 12-30
133 armavir 10-20
413 adler 11-10

Пример 4. Имеется база данных, содержащая данные о спортсменах: имя и вид спорта. Определить возможные пары одного из спортсменов-теннисистов с другими теннисистами.

Решение:

DOMAINS имя,вид_сп=string PREDICATES играет(имя,вид_сп) спис_спортс CLAUSES играет("Саша",теннис). играет("Аня",волейбол). играет("Олег",футбол). играет("Коля",теннис). играет("Саша",футбол). играет("Андрей",теннис). спис_спортс:- играет(X,теннис),!,играет(Y,теннис), X<>Y,write(X,"-",Y),nl,fail. GOAL write("Пары теннисистов"),nl, спис_спортс.	DOMAINS name,vid_sp=string PREDICATES igraet(name,vid_sp) spis_sports CLAUSES igraet("Sasha",tennis). igraet("Anna",volleybol). igraet("Oleg",futbol). igraet("Kolya",tennis). igraet("Sasha",futbol). igraet("Andrey",tennis). spis_sports:- igraet(X,tennis),!,igraet(Y,tennis), X<>Y,write(X,"-",Y),nl,fail. GOAL write("pary tennissistov"),nl, spis_sports.

Результат выполнения программы (см. рис. 5):

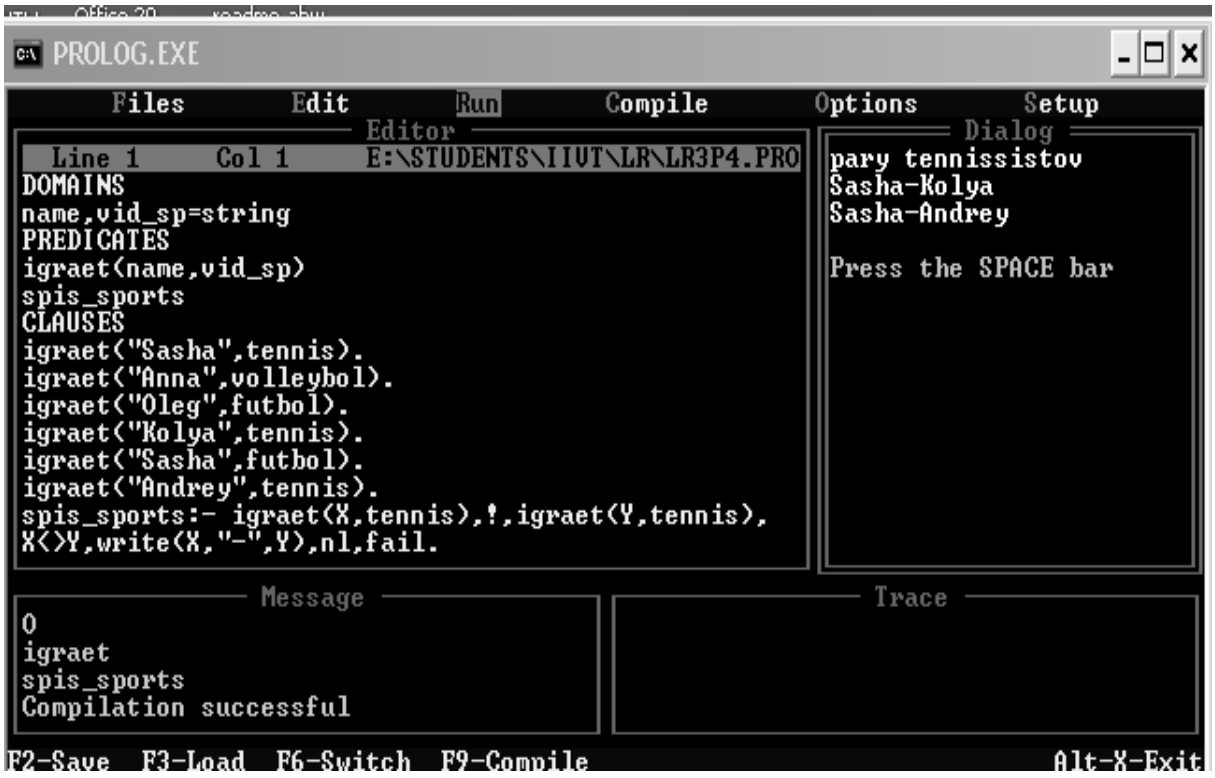


Рис. 5. Результат программы

Пары теннисистов

Саша-Коля

Саша-Андрей

Пример 5. Студенту в зависимости от набранной в процессе обучения суммы баллов **Z** присваивается квалификация:

магистр (**M**), если 80<=Z<=100

специалист (**S**), если 60<= Z< 80

бакалавр (**B**), если 40<= Z< 60

неудачник (**N**), если 0<=Z< 40

Составить программу, которая определит квалификацию в зависимости от введенного значения **Z**

Решение:

```
DOMAINS
z=integer
r=string
PREDICATES
grade(z,r)
CLAUSES
grade(Z,""):-Z<0,!,
write("Неверный ввод данных!").
grade(Z,""):-
Z>100,!,write("Неверный ввод данных!").
grade(Z,"M"):-Z>=80,!.
grade(Z,"S"):-Z>=60,!.
grade(Z,"B"):-Z>=40,!.
grade(Z,"N").
GOAL
write("Z="), readint(Z),
grade(Z,R),write(R).
```

```
DOMAINS
z=integer
r=string
PREDICATES
grade(z,r)
CLAUSES
grade(Z,""):-Z<0,!,
write("nevernuy vvod dannyh!").
grade(Z,""):-
Z>100,!,write("nevernuy vvod dannyh!").
grade(Z,"M"):-Z>=80,!.
grade(Z,"S"):-Z>=60,!.
grade(Z,"B"):-Z>=40,!.
grade(Z,"N").
GOAL
write("Z="), readint(Z),
grade(Z,R),write(R).
```

Для решения задачи составим правило ***grade***, устанавливающее связь между количеством баллов (z) и квалификацией (r). Правило состоит из нескольких частей. Первые две части обеспечивают проверку недопустимых значений Z с выводом

соответствующего сообщения. Остальные части правила определяют квалификацию в зависимости от значения Z.

Комментарий: readint – стандартный предикат ввода целочисленного значения

Результат выполнения программы (см. рис. 6):

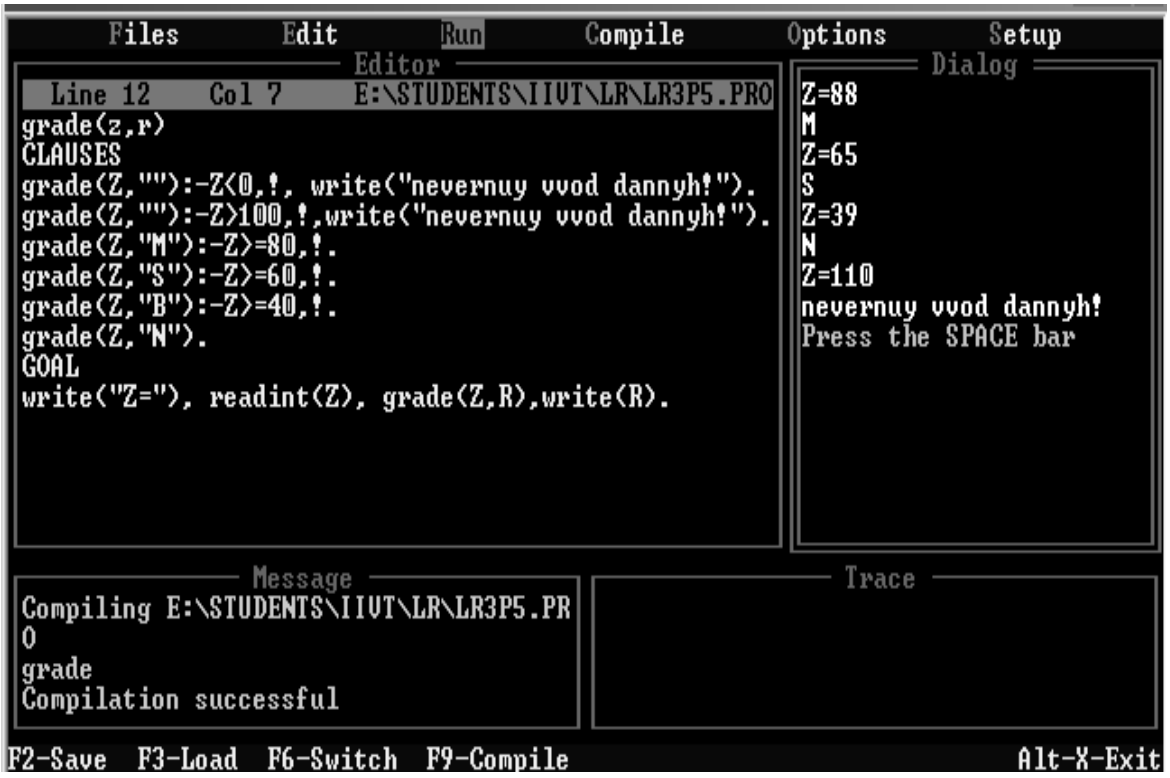


Рис. 6. Результат программы

1-й случай:
Z=88
M
2-й случай:
Z=65
S
3-й случай:
Z=39
N
4-й случай:
Z=110
Неверный ввод данных!

Задания для самостоятельной работы

1. База данных содержит факты вида: **отдыхает(имя, город), украина(город), россия(город), женщина (имя), мужчина(имя).**

а) вывести список женщин, отдыхающих в России;

б) вывести список мужчин, отдыхающих на Украине.

2. База данных содержит факты вида: **книга(автор, название, издательство, год_издания), украина(город).**

а) вывести весь список книг;

б) вывести список книг авторов Пушкина и Чехова;

в) вывести список книг, изданных в издательстве «Питер» не ранее 2000 года.

3. Составить программу, реализующую авиасправочник. В справочнике содержится следующая информация о каждом рейсе: номер рейса, пункт назначения, время вылета, дни (ежедн., чет, нечет). Вывести:

а) всю информацию из справочника;

б) информацию о самолетах, вылетающих в заданный пункт по четным дням;

в) информацию о самолетах, вылетающих ежедневно не позже указанного времени.

4. Составить программу, реализующую географический справочник. В справочнике содержится следующая информация о каждой стране: название страны, название столицы, численность населения, географическое положение (Европа или Азия). Вывести:

а) всю информацию из справочника;

б) информацию о странах, численность населения которых превышает заданное значение;

в) информацию о европейских странах, численность населения которых не превышает заданное значение.

5. Составить программу, реализующую словарь. В словаре содержится следующая информация: слово и его перевод (русские и английские слова). Реализовать вывод всего словаря, перевод с русского на английский, с английского на русский (с несколькими значениями).

6. Составить программу, реализующую телефонный справочник. В справочнике содержится следующая информация о каждом абоненте: имя и телефон. Реализовать вывод всей информации из справочника, поиск телефона по имени, поиск имени по телефону

7. База данных содержит факты вида: ***ученик(имя, класс)*** и ***увлекается(имя, хобби).*** Составить программу, которая выводит:

а) список всех учеников и их увлечения;

б) подбирает одному из учеников указанного класса, увлекающемуся кино, пару из других классов. Вывести все возможные пары.

8. База данных содержит факты вида: ***ученик(имя, класс)*** и ***играет(имя, вид_спорта).*** Составить программу, которая:

а) выводит список всех учеников заданного класса и вид спорта, которым они увлекаются;

б) подбирает одному из учеников указанного класса, играющему в бадминтон, пару из других классов. Вывести все возможные пары.

Отчет о выполненной самостоятельной работе должен содержать:

1) тему лабораторной работы;

2) условие задачи;

3) листинг программы;

4) результаты ее тестирования с различными исходными данными.

ЛАБОРАТОРНАЯ РАБОТА № 5
Арифметические вычисления

Отметим, что Пролог не предназначен для решения вычислительных задач, его возможности вычислений аналогичны соответствующим возможностям таких языков программирования как Basic, C, Pascal.

В языке Пролог имеется ряд встроенных функций для вычисления арифметических выражений, некоторые из которых перечислены в таблице 1.

Табл. 1. Математические операции и функции в Прологе

`X + Y`	Сумма `X` и `Y`
`X - Y`	Разность `X` и `Y`
`X * Y`	Произведение `X` и `Y`
`X / Y`	Деление `X` на `Y`
`X mod Y`	Остаток от деления `X` на `Y`
`X div Y`	Целочисленное деление `X` на `Y`
`abs(X)`	Абсолютная величина числа `X`
`sqrt(X)`	Квадратный корень из `X`
`random(X)`	Случайное число в диапазоне от 0 до 1
`random(Int,X)`	Случайное целое число в диапазоне от 0 до `Int`
`round(X)`	Округление X
`trunc(X)`	Целая часть X
`sin(X)`	Синус `X`
`cos(X)`	Косинус `X`
`arctan(X)`	Арктангенс X
`tan(X)`	Тангенс `X`
`ln(X)`	Натуральный логарифм `X`
`log(X)`	Логарифм X по основанию 10

Пример 1. Вычислить значение выражения Z=(2*X+Y)/(X-Y) для введенных X и Y.

Решение:

```
PREDICATES
знач_выраж(real,real)
CLAUSES
знач_выраж(X,Y):-X<>Y,
Z=(2*X+Y)/(X-Y),
write("Z=",Z);
write ("Делить на 0 нельзя!").
GOAL
```

```
PREDICATES
value_exp(real,real)
CLAUSES
value_exp(X,Y):-X<>Y,
Z=(2*X+Y)/(X-Y),
write("Z=",Z);
write ("cannot be divided by 0!").
GOAL
```

Write("X="),readreal(X), Write("Y="),readreal(Y),знач_выра ж(X,Y),nl.	Write("X="),readreal(X), Write("Y="),readreal(Y),value_ex p(X,Y),nl.

Комментарий: readreal – предикат для ввода действительных чисел

Результат выполнения программы (см. рис. 1):

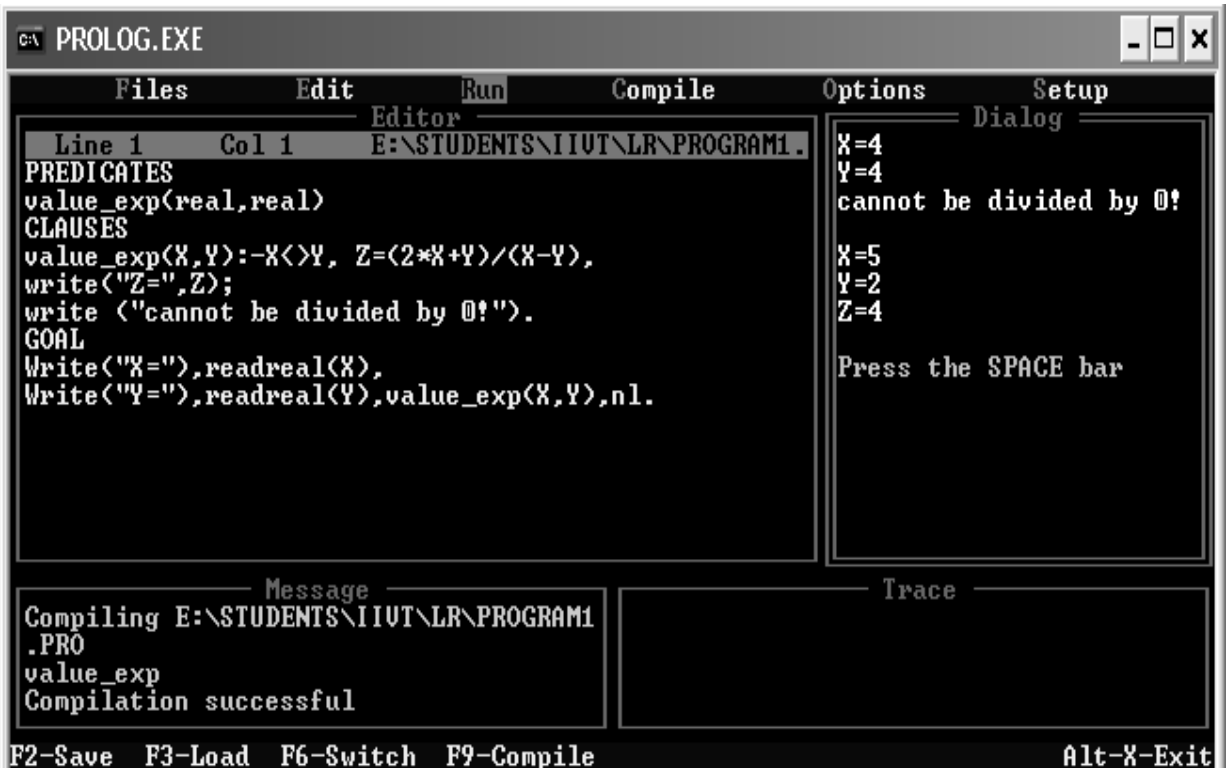


Рис. 1. Результат программы

1-й случай:
X=4
Y=4
Делить на 0 нельзя!
2-й случай:
X=5
Y=2
Z=4

Пример 2. Найти минимальное из двух введенных A и B.

Решение:

```
PREDICATES
min(integer,integer,integer)
CLAUSES
min(A,B,A):-A<=B,!.
min(A,B,B).
GOAL
Write("A="),readreal(A),Write("B="),readreal(B),
min(A,B,Min),write("min=",Min),nl.
```

Результат выполнения программы (см. рис. 9):

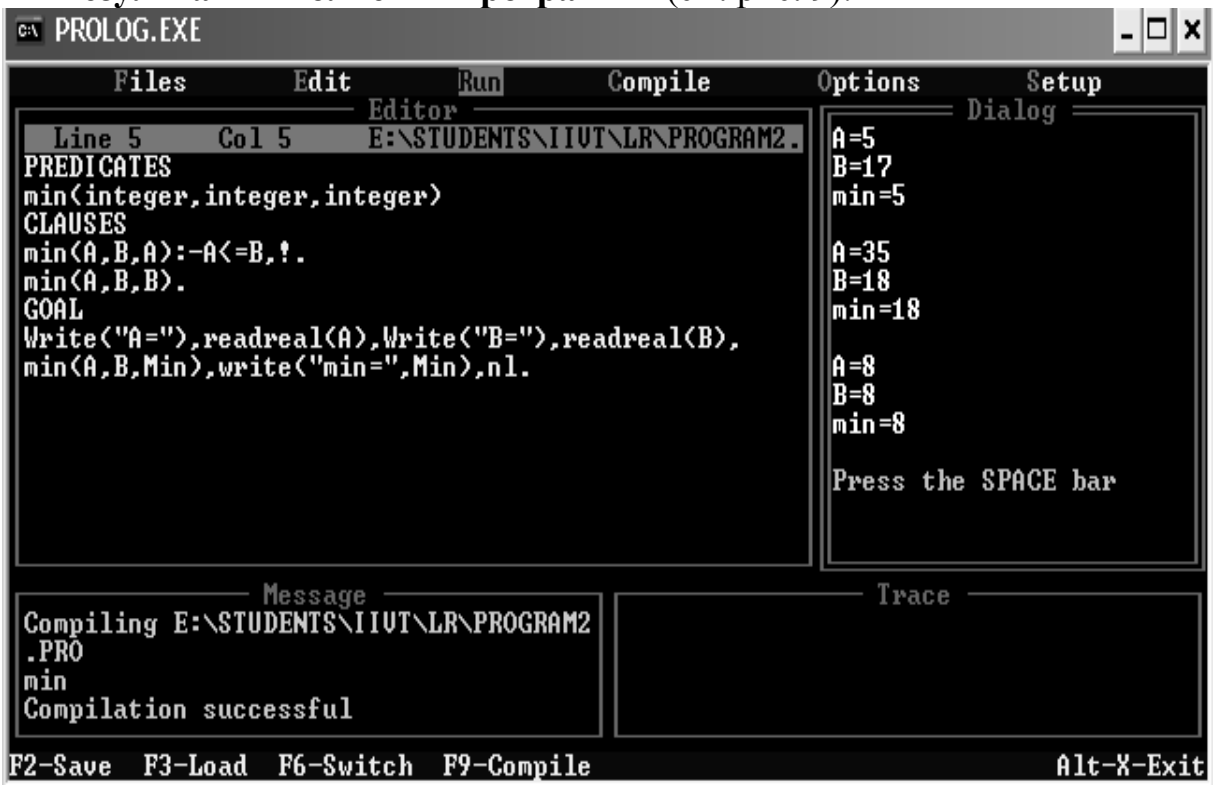


Рис. 2. Результат программы

1-й случай:
A=5
B=17
min=5
2-й случай:
A=35
B=18
min=18
3-й случай:
A=8
B=8
min=8

Пример 3. Определить, является четным или нечетным случайным образом выбранное число от 0 до 20.

Решение:

PREDICATES chet CLAUSES chet:-random(20,X),write(X),X mod 2=0, write(" - четное"),!. chet:-write(" - нечетное"). GOAL chet.	PREDICATES chet CLAUSES chet:-random(20,X),write(X),X mod 2=0, write(" - EVEN"),!. chet:-write(" - ODD"). GOAL chet.

Результат выполнения программы (см. рис. 10):

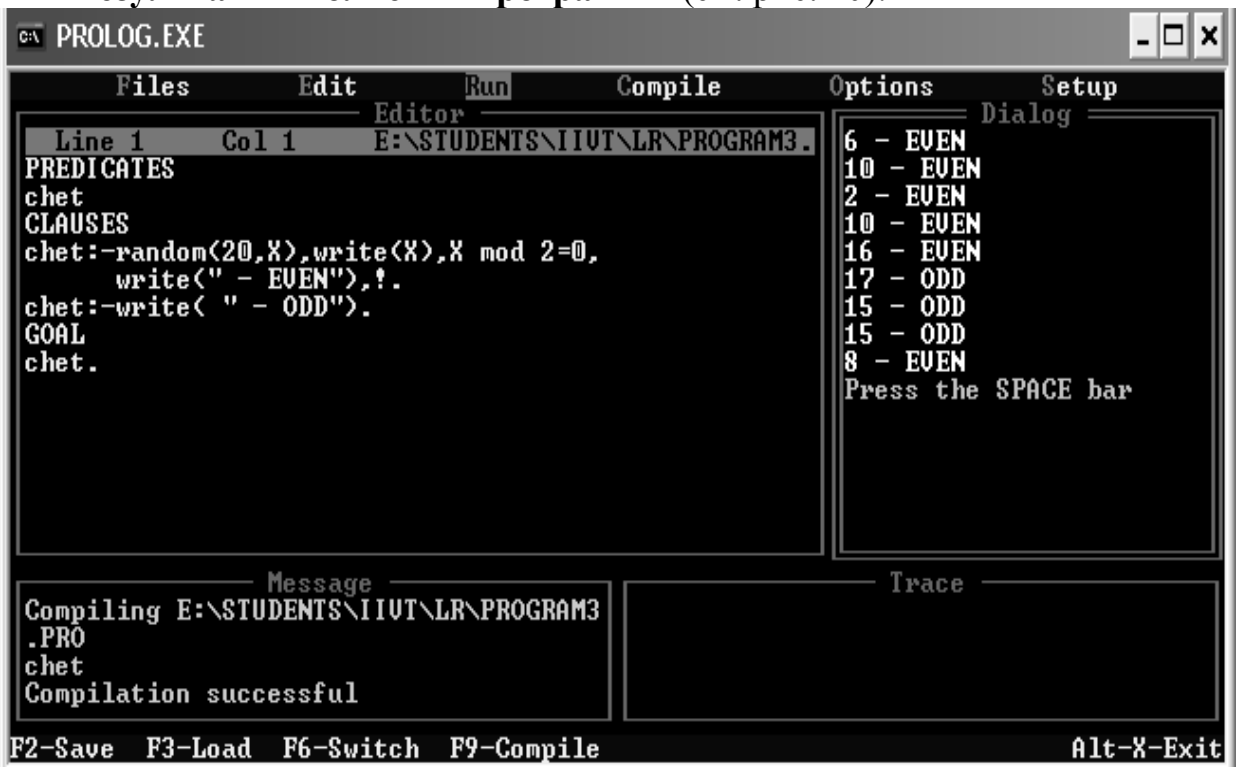


Рис. 3. Результат программы

1-й случай:
6 – четное
2-й случай:
19 – нечетное

<u>Задания для самостоятельной работы</u>

1. Составить программу для вычисления значения выражения Y=(X2+1)/(X-2) для введенного X.

2. Составить программу для вычисления значения выражения S=2(X2+Y2)/(X+Y) для введенных X и Y.

3. Составить программу для вычисления значения выражения $z=e^x \sin x +3\ln x$ для введенного X.

4. Составить программу для вычисления значения выражения $y=\ln(\lg(\sin x+e^x))$ для введенного X.

5. Составить программу для вычисления среднего арифметического двух введенных чисел.

6. Составить программу для вычисления среднего геометрического двух введенных чисел.

7. Составить программу для проверки введенного натурального числа на четность.

8. Составить программу для проверки попадает ли введенное число X в заданный промежуток [a,b].

9. Составить программу для выбора наименьшего из трех введенных чисел.

10. Составить программу для выбора наибольшего из трех введенных чисел.

Отчет о выполненной самостоятельной работе должен содержать:

1) тему лабораторной работы;
2) условие задачи;
3) листинг программы;
4) результаты ее тестирования с различными исходными данными.

ЛАБОРАТОРНАЯ РАБОТА № 6
Рекурсия

Рекурсивная процедура – это процедура, вызывающая сама себя до тех пор, пока не будет соблюдено некоторое условие, которое остановит рекурсию. Такое условие называют *граничным*. Рекурсивное правило всегда состоит, по крайней мере, из двух частей, одна из которых является *нерекурсивной*. Она и определяет граничное условие.

В рекурсивной процедуре нет проблемы запоминания результатов ее выполнения, потому что любые вычисленные значения можно передавать из одного вызова в другой как аргументы рекурсивно вызываемого предиката. Рекурсия является эффективным способом для решения задач, содержащих в себе подзадачу такого же типа.

Пример 1. База данных содержит следующие факты:

roditel(ivan,oleg).
roditel(inna,oleg).
roditel(oleg,dima).
roditel(oleg,marina).

Составить рекурсивное правило ***предок*** и определить всех предков и их потомков.

Решение:

DOMAINS name=string PREDICATES roditel(name,name) predok(name,name) CLAUSES roditel(ivan,oleg). roditel(inna,oleg). roditel(oleg,dima). roditel(oleg,marina). predok(X,Z):-roditel(X,Z). *% нерекурсивная часть правила*	DOMAINS name=string PREDICATES roditel(name,name) predok(name,name) CLAUSES roditel(ivan,oleg). roditel(inna,oleg). roditel(oleg,dima). roditel(oleg,marina). predok(X,Z):-roditel(X,Z). predok(X,Z):-roditel(X,Y),

predok(X,Z):-roditel(X,Y), *% рекурсивная часть правила* predok(Y,Z). GOAL predok(X,Y), write("Predok -",X," Ego potomok-",Y),nl,fail.	predok(Y,Z). GOAL predok(X,Y), write("Predok -",X," Ego potomok-",Y),nl,fail.

Результат выполнения программы (см. рис. 1):

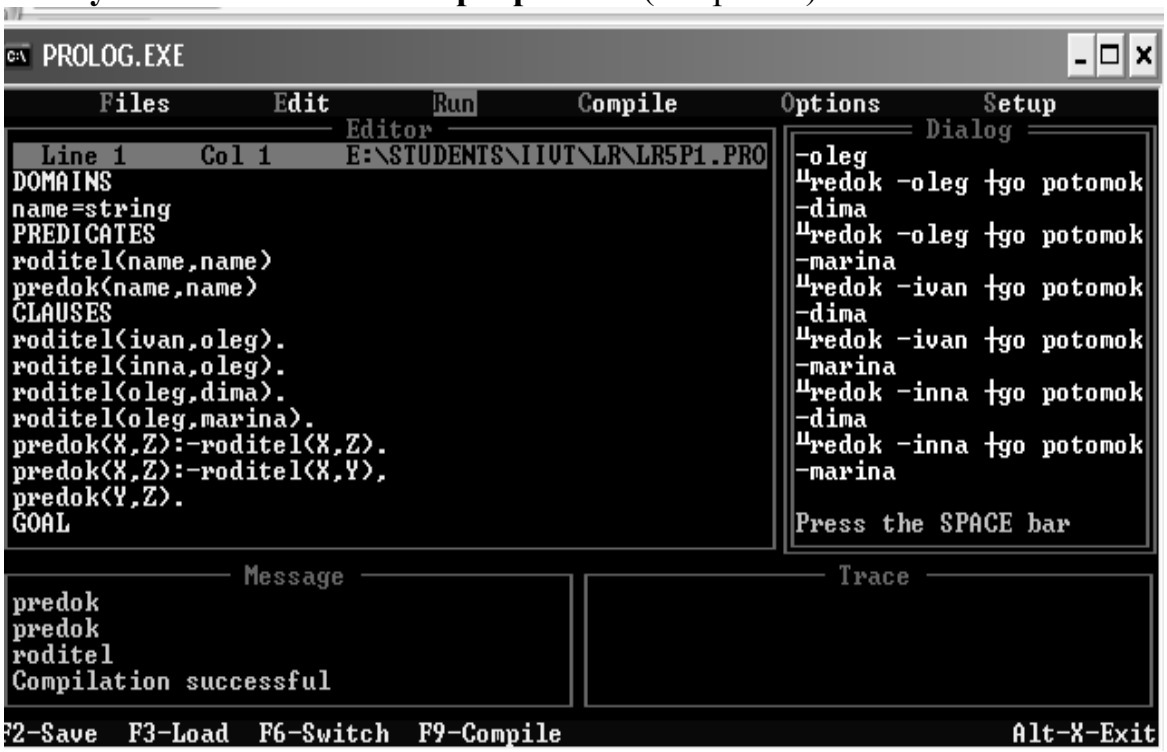


Рис. 1. Результат программы

Predok -ivan Ego potomok-oleg
Predok -inna Ego potomok-oleg
Predok -oleg Ego potomok-dima
Predok -oleg Ego potomok-marina
Predok -ivan Ego potomok-dima
Predok -ivan Ego potomok-marina
Predok -inna Ego potomok-dima
Predok -inna Ego potomok-marina

Пример 2. Вычисление факториала.

Решение:

PREDICATES fact(integer,integer) CLAUSES fact(0,1):-!. *% Факториал нуля равен единице*	PREDICATES fact(integer,integer) CLAUSES fact(0,1):-!. fact(N,F):- N1=N-1,

fact(N,F):- N1=N-1, *% уменьшаем N на единицу,* fact(N1,F1)*, % вычисляем факториал нового числа,* F=N*F1. *% а затем умножает его на N* GOAL write("N="),readint(N),fact(N,F),write("F=",F),nl.	fact(N1,F1), F=N*F1. GOAL write("N="),readint(N),fact(N,F),write("F=",F),nl.

Результат выполнения программы (см. рис. 2):

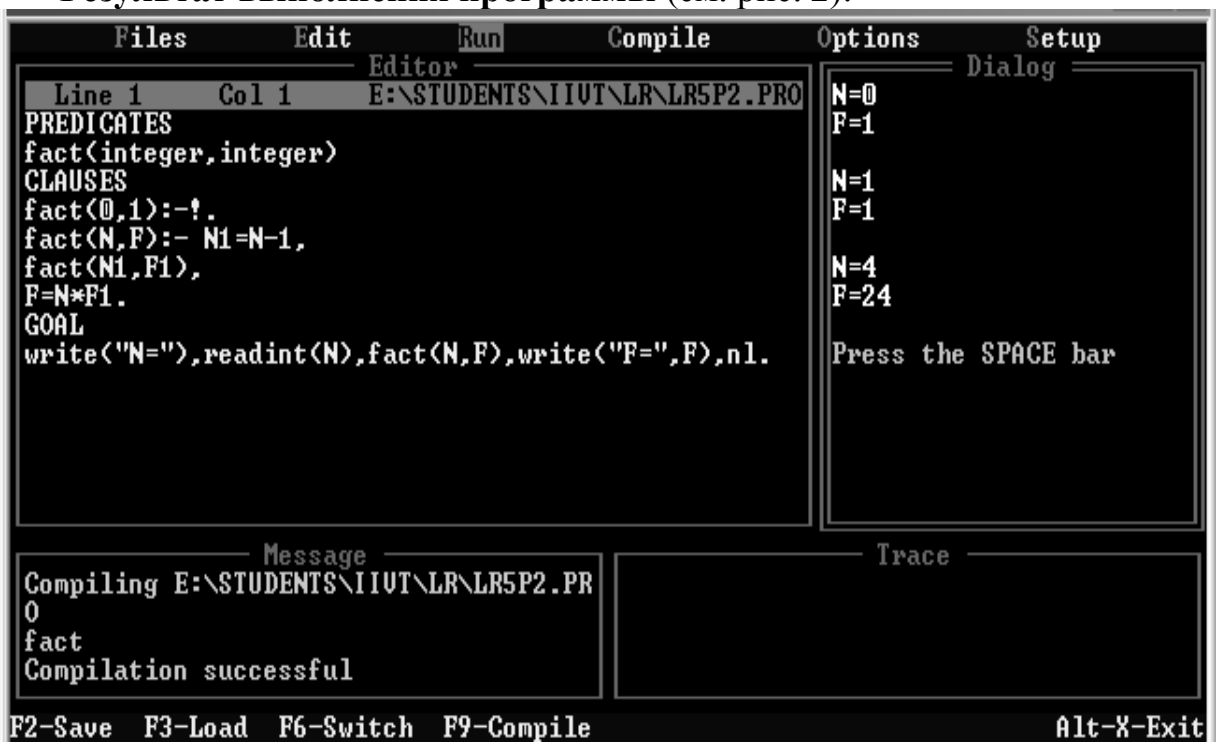


Рис. 2. Результат программы

1-й случай:	*2-й случай:*	*3-й случай:*
N=0 F=1	N=1 F=1	N=4 F=24

Пример 3. Составить программу для вычисления $Y=X^n$, X, n – целые числа

Решение:

Составим правило **stepen**, состоящее из 3-х частей.

1-я часть правила (нерекурсивная) определяет, что $X^0=1$.

2-я часть правила (рекурсивная) вычисляет X^n для положительного n.

3-я часть (рекурсивная) - вычисляет X^n для отрицательного n (добавляется необходимое условие X<>0)

PREDICATES stepen(real,real,real) CLAUSES stepen(X,0,1):-!. stepen(X,N,Y):-N>0,N1=N-1,stepen(X,N1,Y1),Y=Y1*X,!. stepen(X,N,Y):-X<>0,K=-N,stepen(X,K,Z),Y=1/Z. GOAL write("X="),readreal(X), write("N="),readreal(N), stepen(X,N,Y),write("Y=",Y),nl.	PREDICATES stepen(real,real,real) CLAUSES stepen(X,0,1):-!. stepen(X,N,Y):-N>0,N1=N-1,stepen(X,N1,Y1),Y=Y1*X,!. stepen(X,N,Y):-X<>0,K=-N,stepen(X,K,Z),Y=1/Z. GOAL write("X="),readreal(X), write("N="),readreal(N), stepen(X,N,Y),write("Y=",Y),nl.

Результат выполнения программы (см. рис. 3):

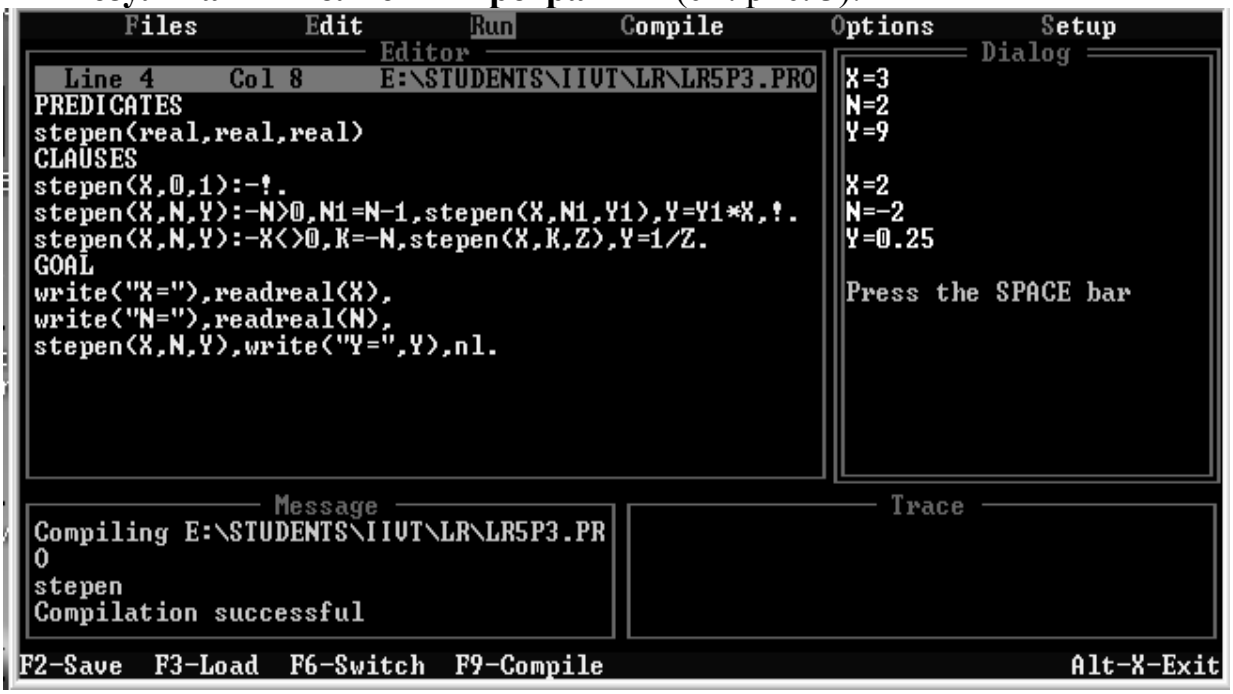


Рис. 3. Результат программы

1-й случай:

X=3

N=2

Y=9

2-й случай:

X=2

N=-2

Y=0.25

Пример 4 (Ханойские башни). Имеется три стержня: A, B и C. На стержне A надеты N дисков разного диаметра, надетые друг на друга в порядке убывания диаметров. Необходимо переместить диски со стержня A на

стержень C используя B как вспомогательный, если перекладывать можно только по одному диску и нельзя больший диск класть на меньший.

Решение:

```
PREDICATES
move(integer,char,char,char)
CLAUSES
move(1,A,B,C):-
write("Перенести диск с ",A," на ",C),nl,!.
move(N,A,B,C):-
M=N-1,move(M,A,C,B),
write("Перенести диск с ",A," на ",C),nl,
move(M,B,A,C).
GOAL
write("Ханойские башни"), nl,
write("Количество дисков:"), readint(N),nl,
move(N,'A','B','C').
```

```
PREDICATES
move(integer,char,char,char)
CLAUSES
move(1,A,B,C):-
write("perenesti c ",A," na ",C),nl,!.
move(N,A,B,C):-
M=N-1,move(M,A,C,B),
write("perenesti disk c",A," na ",C),nl,
move(M,B,A,C).
GOAL
write("hanoyskie bashni"), nl,
write("kolicestvo diskov:"), readint(N),nl,
move(N,'A','B','C').
```

Составим правило **move**, определяющее порядок переноса дисков.

1-я (нерекурсивная) часть правила определяет действие, если на стержне находится 1 диск.

2-я (рекурсивная) часть правила перемещает сначала верхние N-1 диск на стержень B, используя C как вспомогательный, затем оставшийся диск на стержень C и, наконец, диски со стержня B на C, используя A как вспомогательный.

Результат выполнения программы (см. рис. 4):

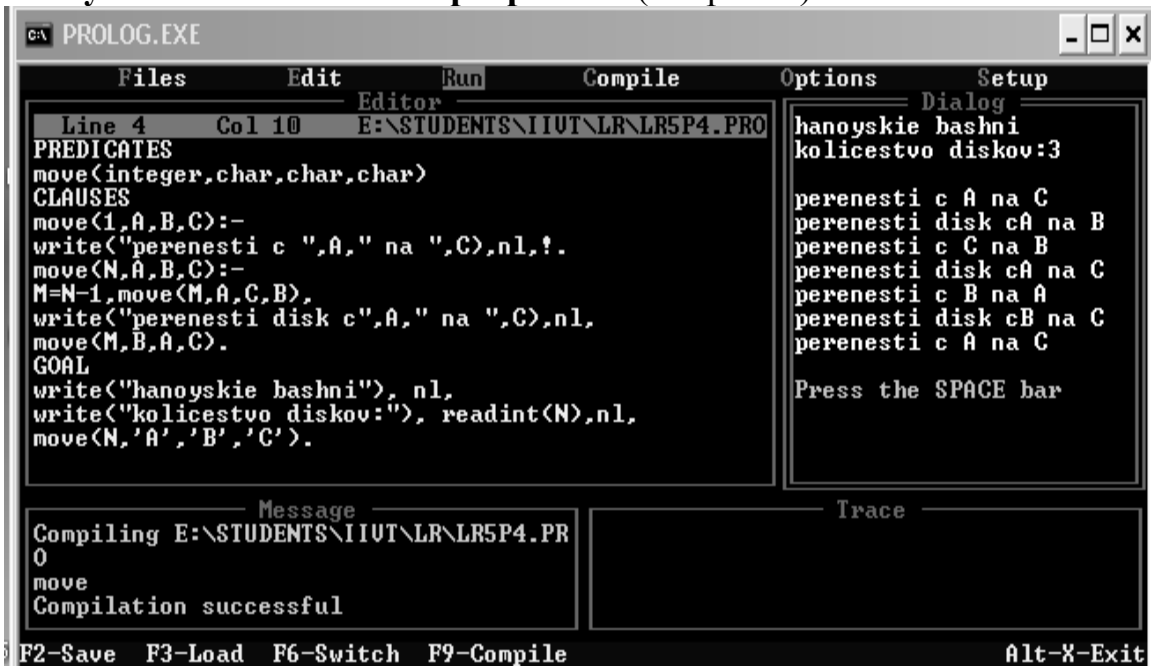


Рис. 4. Результат программы

Ханойские башни
*Количество дисков:*3
Перенести диск с A на C
Перенести диск с A на B
Перенести диск с C на B
Перенести диск с A на C
Перенести диск с B на A
Перенести диск с B на C
Перенести диск с A на C

Задания для самостоятельной работы

1. Вычислить сумму 1+2+3+…+N.
2. Подсчитать сумму ряда целых четных чисел от 2 до N.
3. Вычислить сумму ряда целых нечетных чисел от 1 до n.
4. Найти значение произведения: 2*4*6*...*26
5. Найти значение произведения: 1*3*5*...*11
6. Вычислить значение n-го члена ряда Фибоначчи: f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2).
7. Используя базу данных и правило **предок** из примера 2 составить правило для определения всех потомков-мужчин.
8. Используя базу данных и правило **предок** из примера 2 составить правило для определения всех потомков-женщин.

Отчет о выполненной самостоятельной работе должен содержать:

1) тему лабораторной работы;
2) условие задачи;
3) листинг программы;
4) результаты ее тестирования.

ЛАБОРАТОРНАЯ РАБОТА № 7
Решение логических задач в Прологе

Пролог позволяет наиболее естественным образом решать логические задачи, моделируя процесс размышления человека с помощью правил.

Многие логические задачи связаны с рассмотрением нескольких конечных множеств с одинаковым количеством элементов, между которыми устанавливается взаимно-однозначное соответствие. В Прологе эти множества можно описывать как базы данных, а зависимости между объектами устанавливать с помощью правил.

Пример 1. В автомобильных гонках три первых места заняли Алеша, Петя и Коля. Какое место занял каждый из них, если Петя занял не второе и не третье место, а Коля – не третье?

Решение:

Традиционным способом задача решается заполнением таблицы. По условию задачи Петя занял не второе и не третье место, а Коля – не третье. Это позволяет поставить символ '-' в соответствующих клетках.

Имя	I место	II место	III место
Алеша			
Петя		-	-
Коля			-

Между множеством имен участников гонки и множеством мест должно быть установлено взаимнооднозначное соответствие. Поэтому определяем занятое место сначала у Пети, затем у Коли и, наконец, у Алеши. В соответствующих клетках проставляем знак '+'. В каждой строке и каждом столбце должен быть только один такой знак.

Имя	I место	II место	III место
Алеша	-	-	+
Петя	+	-	-
Коля	-	+	-

Из последней таблицы следует, что Алеша занял третье место, Петя - первое, Коля - второе.

Программа на Прологе будет выглядеть следующим образом:

```
PREDICATES
имя(string)
место(string)
соответствие(string,string)
решение(string,string,string,string,string,string)
CLAUSES
имя(алеша).
имя(петя).
имя(коля).
место(первое).
место(второе).
место(третье).
/* Устанавливаем взаимнооднозначное соответствие
между множеством имен и множеством мест, X - имя,Y - место */
```

```
PREDICATES
name(string)
place(string)
 (string,string)
decision(string,string,string,string,string,string)
CLAUSES
name(alesha).
name (patya).
name (colya).
place(one).
place (two).
place (three).
/* Устанавливаем взаимнооднозначное соответствие между множеством имен и множеством мест, X - имя,Y - место */
```

/* *Петя занял не второе и не третье место* */ соответствие(X, Y):-имя(X), X=петя, место(Y),not(Y=второе), not(Y=третье). /* *Коля занял не третье место* */ соответствие(X, Y):- имя(X), X=коля, место(Y), not(Y=третье). соответствие(X, Y):- имя(X), X=алеша, место(Y). /* *У всех ребят разные места* */ решение(X1,Y1,X2,Y2,X3,Y3):- X1=петя,соответствие(X1,Y1), X2=коля,соответствие(X2,Y2), X3=алеша,соответствие(X3,Y3), Y1<>Y2, Y2<>Y3, Y1<>Y3. GOAL решение(X1,Y1,X2,Y2,X3,Y3), write(X1," - ",Y1),nl, write(X2," - ",Y2),nl,write(X3," - ",Y3),nl.	/* *Петя занял не второе и не третье место* */ accordance (X, Y):-name(X), X=petya, place(Y),not(Y=two), not(Y=three). /* *Коля занял не третье место* */ accordance (X, Y):- name(X), X=colya, place(Y), not(Y=three). accordance (X, Y):- name(X), X=alesha, place(Y). /* *У всех ребят разные места* */ decision(X1,Y1,X2,Y2,X3,Y3):- X1=petya, accordance(X1,Y1), X2=colya, accordance(X2,Y2), X3=alesha, accordance(X3,Y3), Y1<>Y2, Y2<>Y3, Y1<>Y3. GOAL decision (X1,Y1,X2,Y2,X3,Y3), write(X1," - ",Y1),nl, write(X2," - ",Y2),nl,write(X3," - ",Y3),nl.

Результат выполнения программы (см. рис. 1):

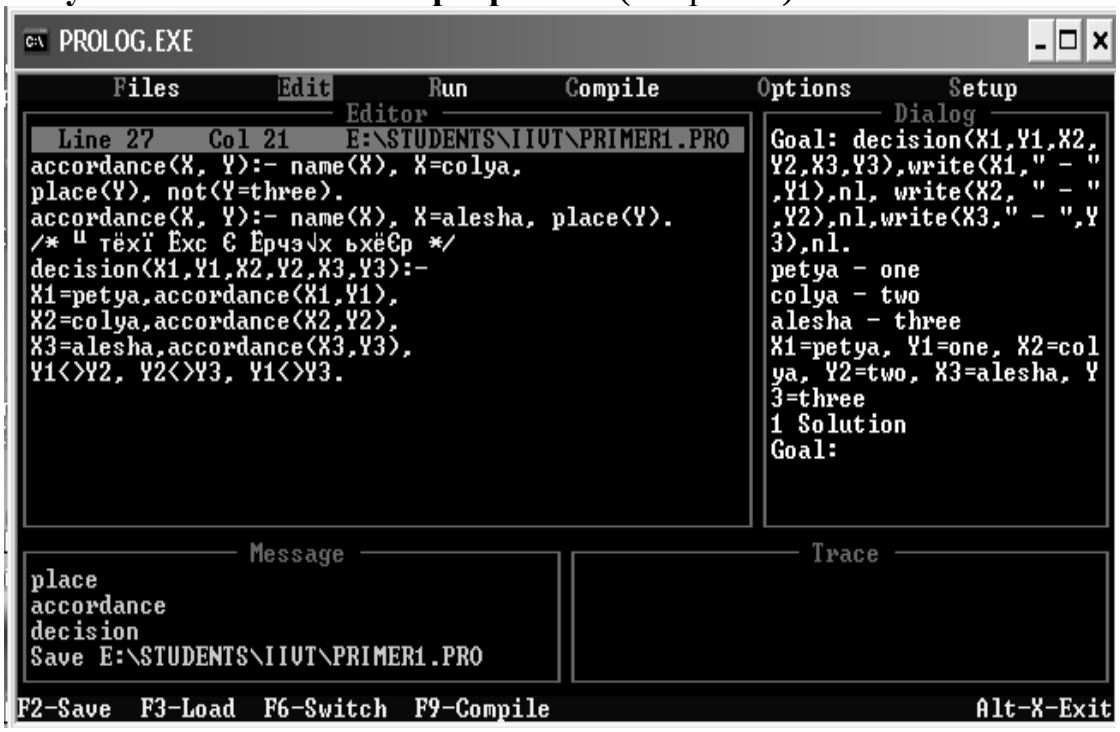


Рис. 1. Результат программы

петя – первое
коля – второе
алеша – третье

Пример 2. Наташа, Валя и Аня вышли на прогулку, причем туфли и платье каждой были или белого, или синего, или зеленого цвета. У Наташи были зеленые туфли, а Валя не любит белый цвет. Только у Ани платье и туфли были одного цвета. Определить цвет туфель и платья каждой из девочек, если у всех туфли и платья были разного цвета.

Решение:

```
PREDICATES
имя(string)
туфли(string)
платье(string)
соот(string,string,string)
решение(string,string,string,string,string,string,
string,string,string)
CLAUSES
имя(наташа).
имя(валя).
имя(аня).
туфли(белый).
туфли(синий).
туфли(зеленый).
платье(белый).
платье(синий).
платье(зеленый).
% X – имя, Y – цвет туфель, Z – цвет платья
соот(X,Y,Z):-
имя(X),туфли(Y),платье(Z),
X=наташа,Y=зеленый,Y<>Z.
соот(X,Y,Z):-
имя(X),туфли(Y),платье(Z),
X=валя,not(Y=белый),
not(Z=белый), Y<>Z.
соот(X,Y,Z):-
имя(X),туфли(Y),платье(Z),X=аня,Y=Z.
реше-
ние(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3):-
X1=наташа,соот(X1,Y1,Z1),
X2=валя, соот(X2,Y2,Z2),
X3=аня, соот(X3,Y3,Z3),
Y1<>Y2, Y2<>Y3, Y1<>Y3,
```

```
PREDICATES
name(string)
tyfli(string)
platia(string)
soot(string,string,string)
pew(string,string,string,string,string,string,
string,string,string)
CLAUSES
name(nata).
name(valia).
name(anna).
tyfli(white).
tyfli(blue).
tyfli(green).
platia(white).
platia(blue).
platia(green).
soot(X,Y,Z):-
name(X),tyfli(Y),platia(Z),
X=nata,Y=green,Y<>Z.
soot(X,Y,Z):-
name(X),tyfli(Y),platia(Z),
X=valia,not(Y=white),
not(Z=white), Y<>Z.
soot(X,Y,Z):-
name(X),tyfli(Y),platia(Z),X=anna,Y=Z.
Goal
pew(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3):-
X1=nata,soot(X1,Y1,Z1),
X2=valia, soot(X2,Y2,Z2),
X3=anna, soot(X3,Y3,Z3),
```

Z1<>Z2, Z2<>Z3, Z1<>Z3. GOAL реше- ние(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3), write(X1," туфли- ",Y1," платье-",Z1),nl, write(X2," туфли- ",Y2," платье-",Z2),nl, write(X3," туфли- ",Y3," платье-",Z3),nl.	Y1<>Y2, Y2<>Y3, Y1<>Y3, Z1<>Z2, Z2<>Z3, Z1<>Z3. pew(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3), write(X1," tyfli- ",Y1," platia-",Z1),nl, write(X2," tyfli- ",Y2," platia-",Z2),nl, write(X3," tyfli- ",Y3," platia-",Z3),nl.

Результат выполнения программы (см. рис. 2):

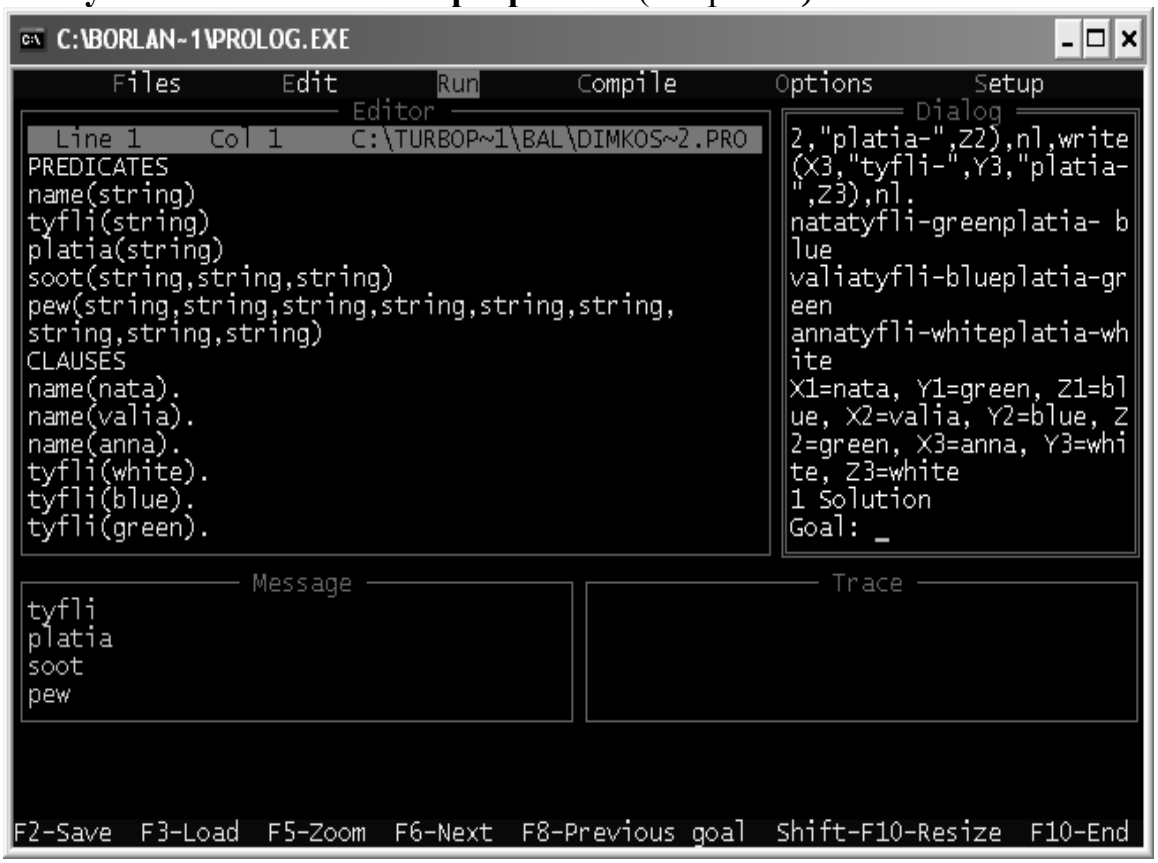


Рис. 2. Окно результата программы

наташа туфли – зеленый платье – синий
валя туфли – синий платье – зеленый
аня туфли – белый платье – белый

Пример 3. Витя, Юра и Миша сидели на скамейке. В каком порядке они сидели, если известно, что Миша сидел слева от Юры, а Витя слева от Миши.

Решение:

PREDICATES слева(string,string) ряд(string,string,string) CLAUSES */* Миша сидел слева от Юры */* слева(миша, юра). */* Витя сидел слева от Миши */* слева(витя, миша). */* Объекты X, Y и Z образуют ряд, если X слева от Y и Y слева от Z */* ряд(X, Y, Z):- слева(X,Y), слева(Y, Z). GOAL ряд(X, Y, Z), write(X,"-",Y,"-",Z),nl.	PREDICATES left(string,string) rad(string,string,string) CLAUSES left(miwa, iyra). left(vitia, miwa). rad(X, Y, Z):- left(X,Y), left(Y, Z). GOOAL rad(X, Y, Z), write(X,"-",Y,"-",Z),nl.

Результат выполнения программы (см. рис. 3):

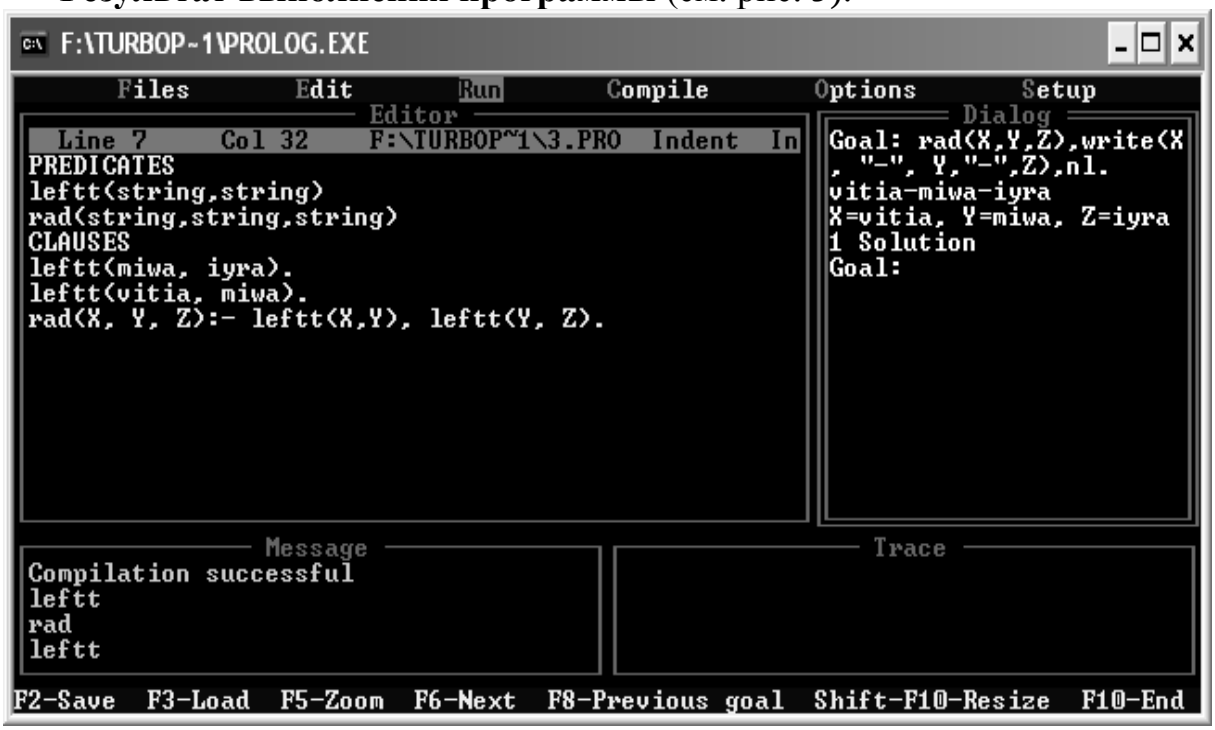


Рис. 3. Результат программы

витя-миша-юра

Пример 4. Известно, что тополь выше березы, которая выше липы. Клен ниже липы, а сосна выше тополя и ниже ели. Определить самое высокое и самое низкое дерево.

Решение:

DOMAINS name=string PREDICATES выше(name,name) ряд(name,name,name,name,name, name) CLAUSES выше(тополь,береза). выше(липа,клен). выше(ель,сосна). выше(береза,липа). выше(сосна,тополь). ряд(X1,X2,X3,X4,X5,X6):- выше(X1,X2),выше(X2,X3), выше(X3,X4),выше(X4,X5), выше(X5,X6). GOAL ряд(X,_,_,_,_,Y),write("Самое высокое - ",X),nl, write("Самое низкое - ",Y),nl.	DOMAINS name=string PREDICATES up(name,name) ryad(name,name,name,name,name, name) CLAUSES up(topol,bereza). up(lipa,klen). up(el,sosna). up(bereza,lipa). up(sosna,topol). ryad(X1,X2,X3,X4,X5,X6):- up(X1,X2),up(X2,X3), up(X3,X4),up(X4,X5), up(X5,X6). GOAL ryad(X,_,_,_,_,Y),write("Most high - ",X),nl, write("Most low - ",Y),nl.

Результат выполнения программы (см. рис. 4):

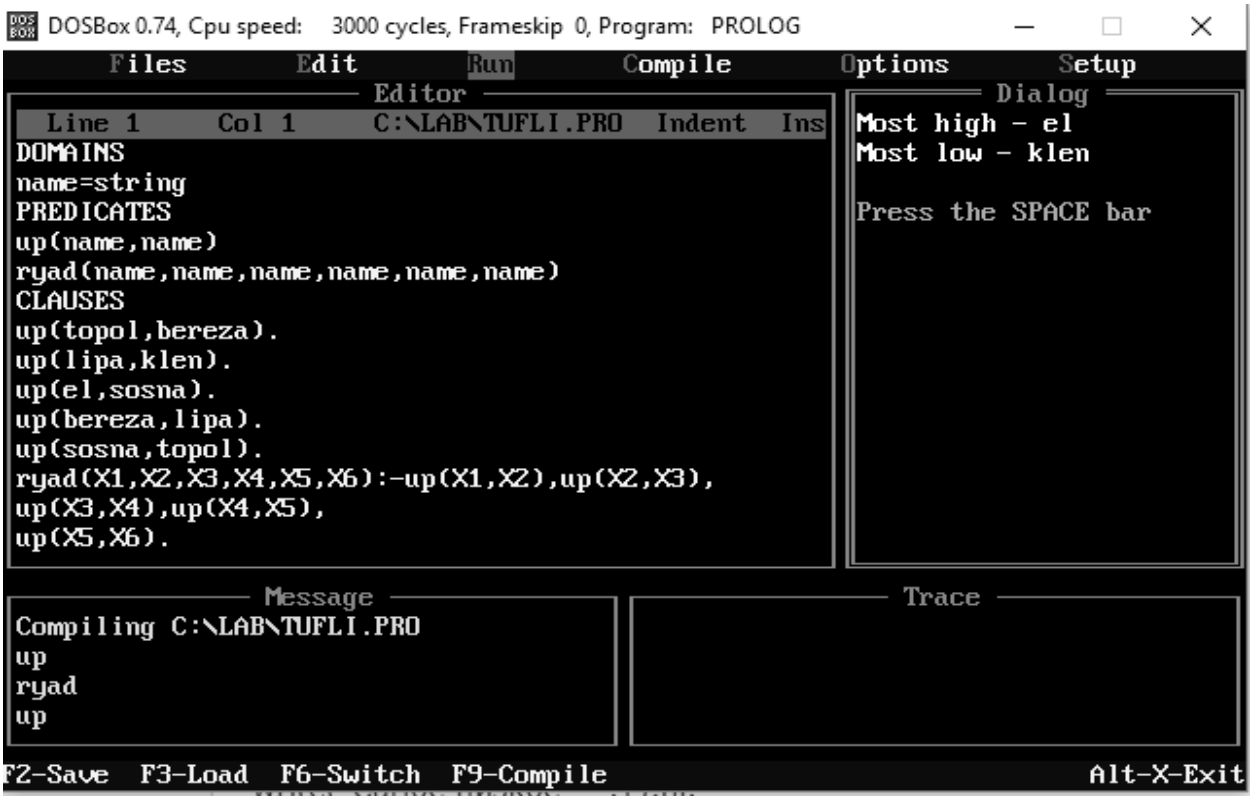


Рис. 4. Результат программы

Самое высокое – ель

Самое низкое – клен

Задания для самостоятельной работы

1. Трое ребят вышли гулять с собакой, кошкой и хомячком. Известно, что Петя не любит кошек и живет в одном подъезде с хозяйкой хомячка. Лена дружит с Таней, гуляющей с кошкой. Определить, с каким животным гулял каждый из детей.

2. Беседуют трое друзей: Белокуров, Рыжов и Чернов. Брюнет сказал Белокурову:

«Любопытно, что один из нас блондин, другой – брюнет, а третий – рыжий, но ни у кого цвет волос не соответствует фамилии». Какой цвет волос у каждого из друзей?

3. Витя, Юра, Миша и Дима сидели на скамейке. В каком порядке они сидели, если известно, что Юра сидел справа от Димы, Миша справа от Вити, а Витя справа от Юры.

4. Известно, что Волга длиннее Амударьи, а Днепр короче Амударьи. Лена длиннее Волги. Определить вторую по протяженности реку.

Отчет о выполненной самостоятельной работе должен содержать:

1) тему лабораторной работы;

2) условие задачи;

3) решение задачи традиционным способом (с помощью таблицы);

4) листинг программы;

5) результаты ее тестирования с различными исходными данными.

ЛАБОРАТОРНАЯ РАБОТА № 8
Списки

Список – это объект, который содержит конечное число других объектов. Список в ПРОЛОГе заключается в квадратные скобки и элементы списка разделяются запятыми. Список, который не содержит ни одного элемента, называется *пустым* списком.

Список является рекурсивным объектом. Он состоит из *головы* (первого элемента списка) и *хвоста* (все последующие элемента). Хвост также является списком. В ПРОЛОГе имеется операция “|”, которая позволяет делить список на голову и хвост. Пустой список нельзя разделить на голову и хвост.

Тип данных "список" объявляется в программе на Прологе следующим образом:

DOMAINS

списковый_тип = тип*

где "тип" - тип элементов списка; это может быть как стандартный тип, так и нестандартный, заданный пользователем и объявленный в разделе DOMAINS ранее.

Основными операциями на списками являются:

- формирование списка;

- объединение списков;
- поиск элемента в списке;
- вставка элемента в список и удаление из списка.

Пример 1. Сформировать список вида [7,6,5,4,3,2,1].

Решение:

```
DOMAINS
list = integer*
PREDICATES
genl(integer, list)
CLAUSES
genl(0,[]):-!.
genl(N,[N|L]):-N1=N-1, genl(N1,L).
GOAL
genl(7,L),write(L),nl.
```

Результат выполнения программы (см. рис. 1):

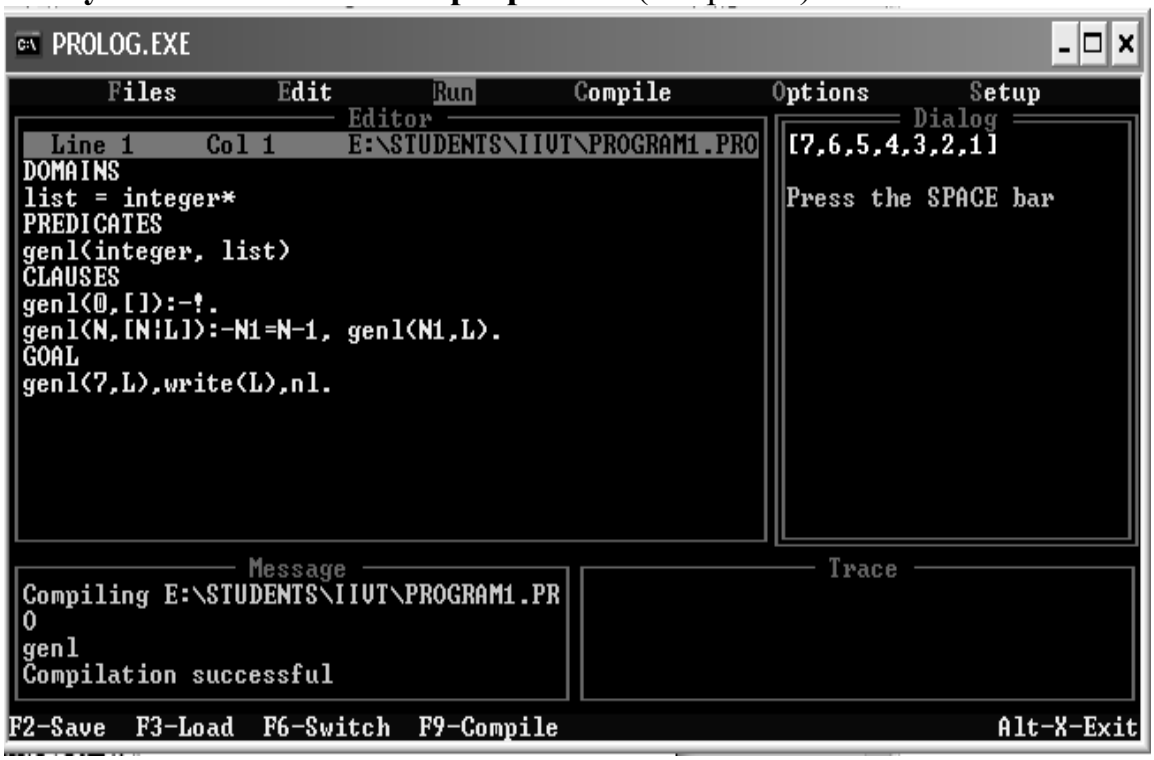


Рис. 1. Результат программы

[7,6,5,4,3,2,1]

Пример 2. Сформировать список из N элементов, начиная с 2. Каждый следующий на 4 больше предыдущего.

Решение:

```
DOMAINS
list = integer*
PREDICATES
genl( integer, integer, list )
```

```
CLAUSES
genl(N2,N2,[]):-!.
genl(N1,N2,[N1|L]):-N1<N2, N=N1+4,
genl(N,N2,L).
GOAL
write("N="),readint(N),K=4*(N+1)-2,
genl(2,K,L),write(L),nl.
```

Результат выполнения программы (см. рис. 2):

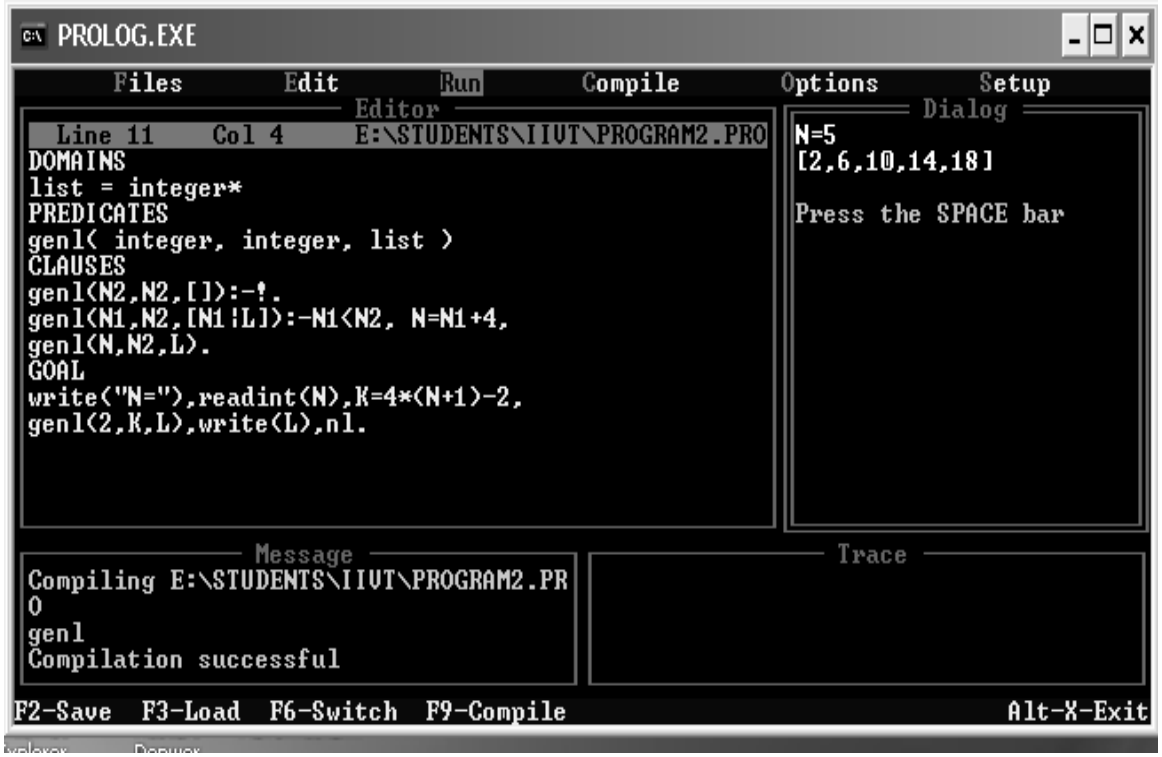


Рис. 2. Результат программы

```
N=5
[2,6,10,14,18]
```

Пример 3. Сформировать список последовательных натуральных чисел от 4 до 20 и найти количество его элементов.

Решение:

DOMAINS list = integer* PREDICATES genl1(integer, integer, list) len(integer, list) CLAUSES genl1(N2,N2,[]):-!. genl1(N1,N2,[N1|L]):-N1<N2, N=N1+1, genl1(N,N2,L). len(0,[]).	DOMAINS list = integer* PREDICATES genl1(integer, integer, list) len(integer, list) CLAUSES genl1(N2,N2,[]):-!. genl1(N1,N2,[N1|L]):-N1<N2, N=N1+1, genl1(N,N2,L). len(0,[]).

len(X,[_|L]):-len(X1,L), X=X1+1. GOAL genl1 (4,21,L),write(L),nl, len(X, L),write("Количество элементов=",X),nl.	len(X,[_|L]):-len(X1,L), X=X1+1. GOAL genl1 (4,21,L),write(L),nl, len(X, L),write("collichestvo elementov=",X),nl.

Результат выполнения программы (см. рис. 3):

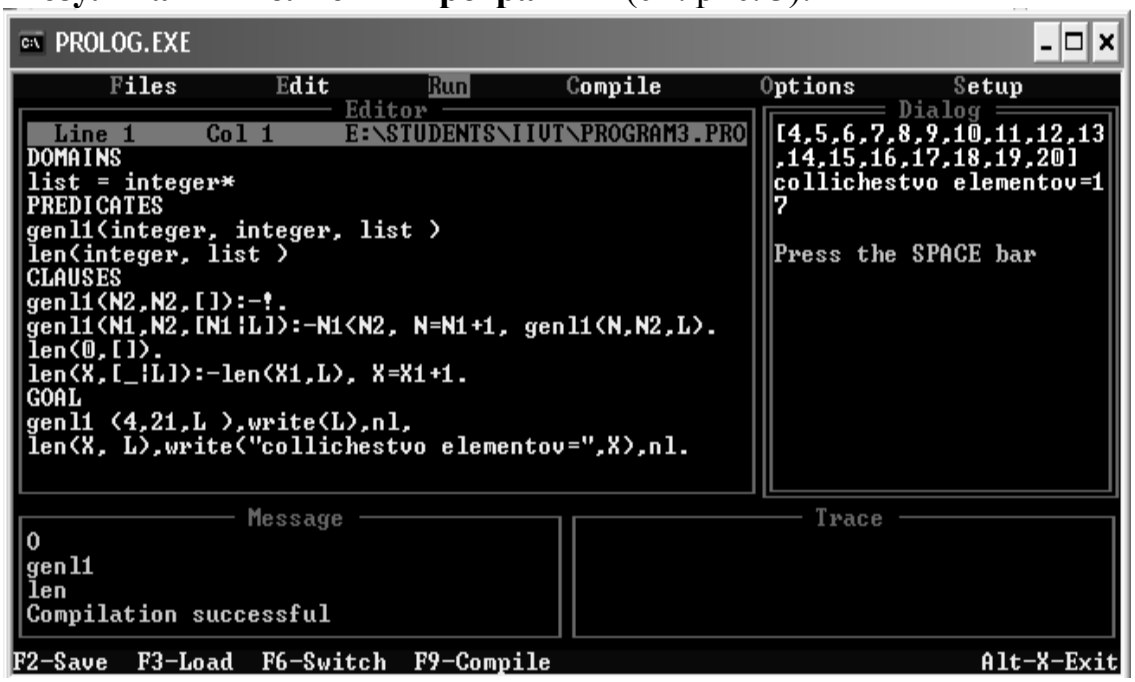


Рис. 3. Результат программы

[4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
Количество элементов=17

Пример 4. Определить, содержится ли введенное число X в заданном списке L.

Решение:

```
DOMAINS
list = integer*
PREDICATES
member(integer, list)
CLAUSES
member(X,[X|_]):-write("yes"),!.
member(X,[]):-write("no"),!.
member(X,[_|L]) :- member(X, L).
GOAL
L=[1,2,3,4], write(L),nl, write("X="),readint(X),
member(X, L),nl.
```

Результат выполнения программы (см. рис. 4):

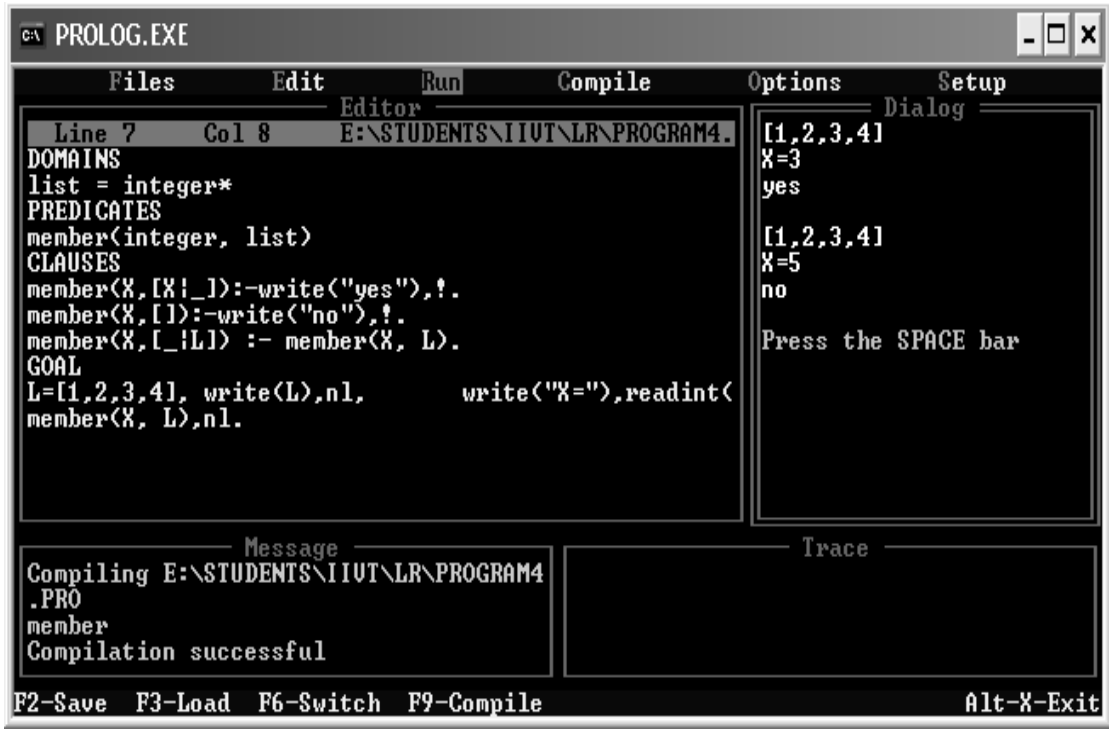


Рис. 4. Результат программы

1-й случай:

[1,2,3,4]

X=3

yes

2-й случай:

[1,2,3,4]

X=5

no

Пример 5. Сформировать списки L1=[1,2,3], L2=[10,11,12,13,14,15] и объединить их в список L3.

Решение:

```
DOMAINS
list = integer*
PREDICATES
genl1(integer,integer,list)
append(list,list,list)
CLAUSES
genl1(N2,N2,[]):-!.
genl1(N1,N2,[N1|L]):-N1<N2,N=N1+1,genl1(N, N2, L).
append([],L,L).
append([X|L1],L2,[X|L3]):-append(L1,L2,L3).
GOAL
genl1(1,4,L1),write("L1=",L1),nl,
genl1(10,16,L2),write("L2=",L2),nl,
append(L1,L2,L3),write("L3=",L3),nl.
```

Результат выполнения программы (см. рис. 5):

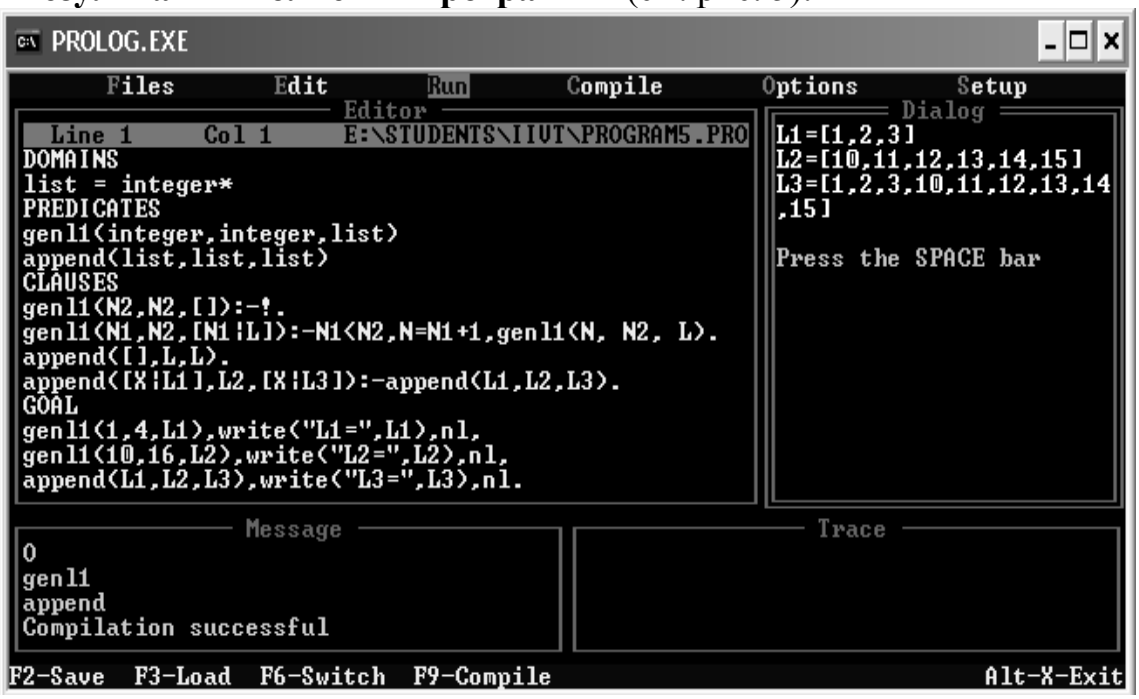


Рис. 5. Результат программы

L1=[1,2,3]

L2=[10,11,12,13,14,15]

L3=[1,2,3,10,11,12,13,14,15]

Пример 6. Удалить из списка, элементами которого являются названия дней недели, указанный элемент.

Решение:

DOMAINS list = symbol* PREDICATES del(symbol,list,list) CLAUSES del(X,[X|L],L). del(X,[Y|L],[Y|L1]):-del(X,L,L1). GOAL L=[пн, вт, ср, чт, пт, сб, вс],write("L=",L),nl, write("X="),readln(X), del(X,L,L1),write("L1=",L1),!; write("Элемент отсутствует в списке"),nl.	DOMAINS list = symbol* PREDICATES del(symbol,list,list) CLAUSES del(X,[X|L],L). del(X,[Y|L],[Y|L1]):-del(X,L,L1). GOAL L=[pn, vt, sr, ht, pt, sb, vs],write("L=",L),nl, write("X="),readln(X), del(X,L,L1),write("L1=",L1),!; write("element v spiske otsytstvyet"),nl.

Результат выполнения программы (см. рис. 6):

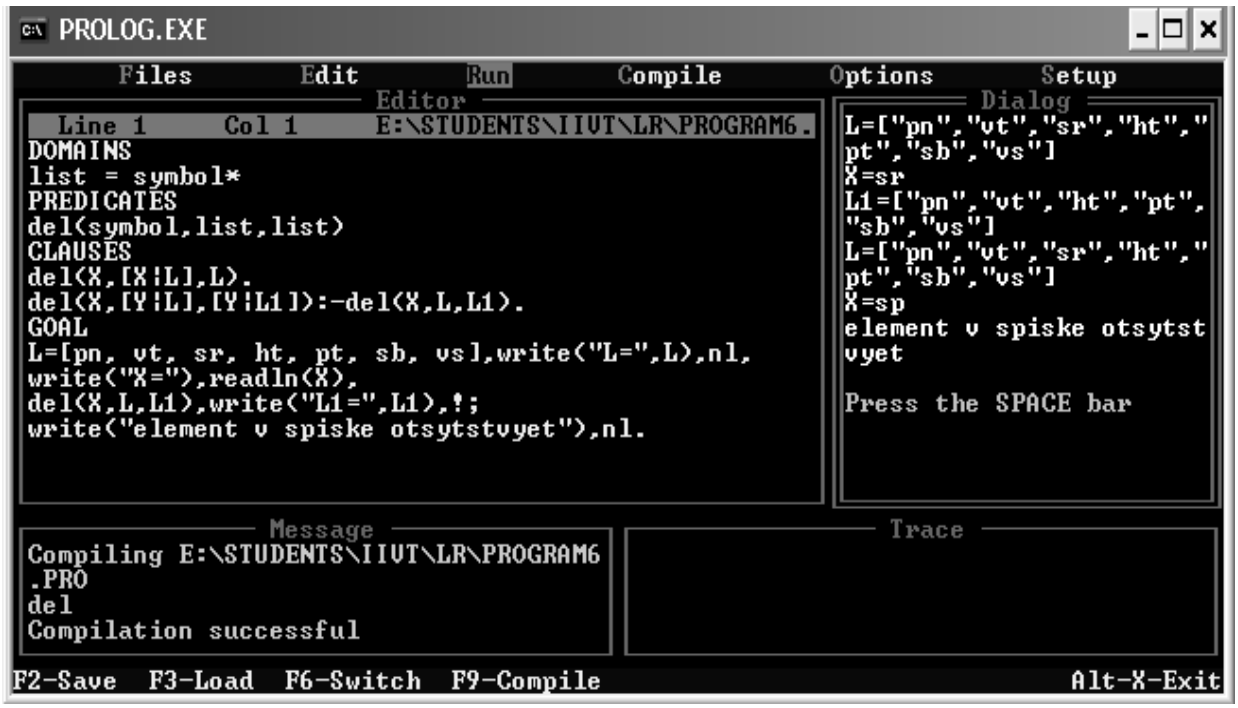


Рис. 6. Результат программы

1-й случай:
L=["пн","вт","ср","чт","пт","сб","вс"]
X=ср
L1=["пн","вт","чт","пт","сб","вс"]
2-й случай:
L=["пн","вт","ср","чт","пт","сб","вс"]
X=пр
Элемент отсутствует в списке

Пример 7. Вставить в список имен новый элемент, значение которого вводится с клавиатуры. Вывести все возможные варианты вставок.

Решение:

```
DOMAINS
list = symbol*
PREDICATES
del(symbol,list,list)
ins(symbol,list,list)
CLAUSES
del(X,[X|L],L).
del(X,[Y|L],[Y|L1]):-del(X,L,L1).
ins(X,L1,L):-del(X,L,L1).
GOAL
L=[olga, oksana, toma, dima],write("L=",L),nl,
write("X="),readln(X),
ins(X,L,L1),write("L1=",L1),nl, fail.
```

Результат выполнения программы (см. рис. 7):

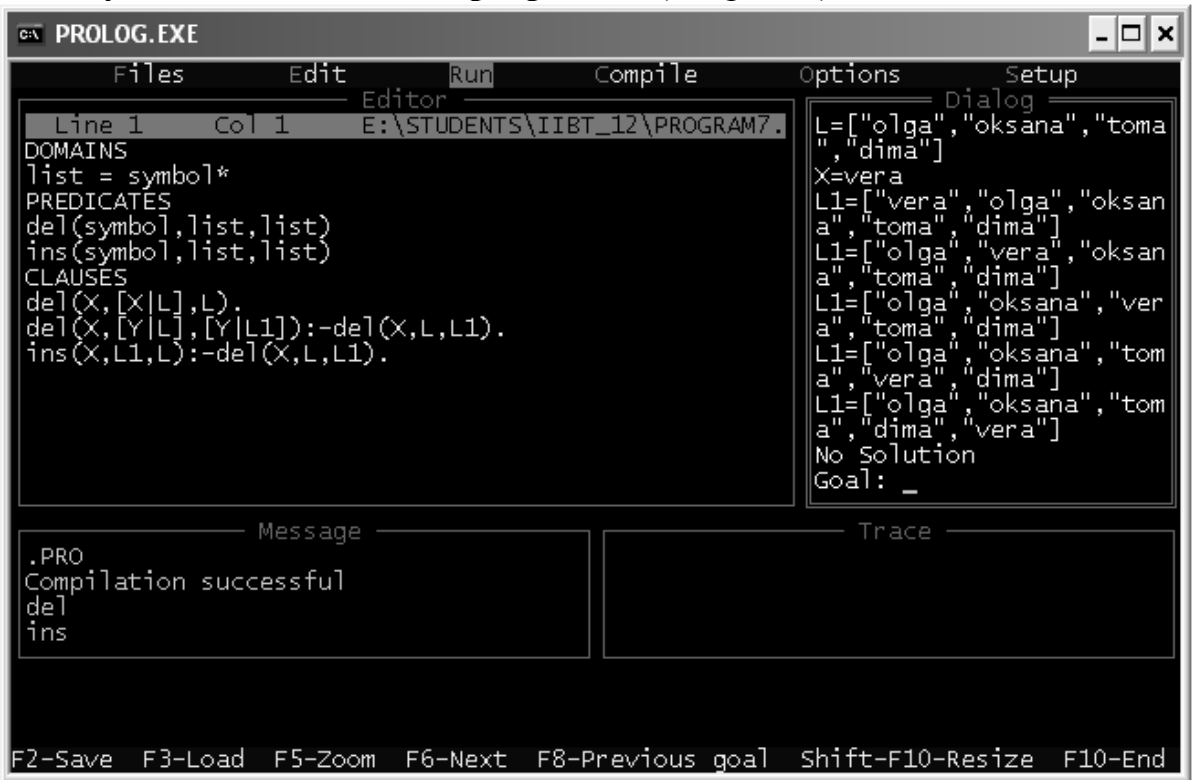


Рис. 7. Результат программы

L=["olga","oksana","toma","dima"]
X=vera
L1=["vera","olga","oksana","toma","dima"]
L1=["olga","vera","oksana","toma","dima"]
L1=["olga","oksana","vera","toma","dima"]
L1=["olga","oksana","toma","vera","dima"]
L1=["olga","oksana","toma","dima","vera"]

Пример 8. Найти сумму элементов списка целых чисел.

Решение:

```
DOMAINS
list=integer*
PREDICATES
sum_list(list, integer)
CLAUSES
sum_list([],0).
sum_list([X|L],S):-sum_list(L,S1),S=S1+X.
GOAL
L=[1,2,3,4,5],sum_list(L,S), write("S=",S).
```

Результат выполнения программы (см. рис. 8):

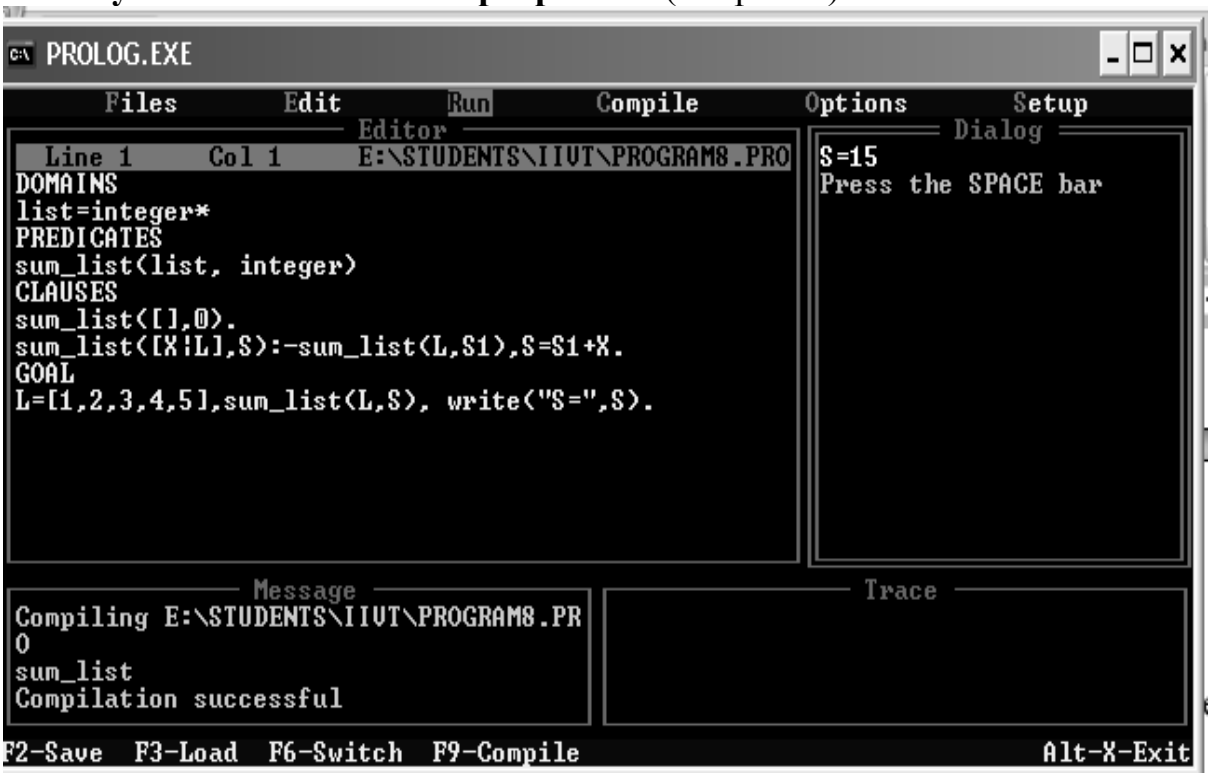


Рис. 8. Результат программы

S=15

Задания для самостоятельной работы

1. Сформировать список [2, 4, 6, 8, 10] и удалить из него введенное число.

2. Сформировать списки [1, 3, 5, 7, 9] и [2, 4, 6, 8, 10] и объединить их в один.

3. Сформировать список [3, 6, 9, 12, 15, 18] и вставить в него введенное число.

4. Сформировать список из N натуральных чисел, начиная с 10. Каждое следующее на 5 больше предыдущего.

5. Сформировать список [3, 6, 9, 12, 15] и найти сумму его элементов

6. Сформировать список [6, 5, 4, 3, 2] и найти сумму его элементов

7. Сформировать список [7, 5, 3, 1] и найти произведение его элементов

8. Сформировать список из N последовательных натуральных чисел, начиная с 10. Найти сумму его элементов

Отчет о выполненной самостоятельной работе должен содержать:

1) тему лабораторной работы;

2) условие задачи;

3) листинг программы;

4) результаты ее тестирования.

СПИСОК ЛИТЕРАТУРЫ

1. *Адаменко А.Н., Кучуков А.М.* Логическое программирование и Visual Prolog. – СПб.: БХВ-Петербург, 2003. – 992 с.
2. *Ахманов А.С.* Логическое учение Аристотеля. – М.: Едиториал УРСС, 2011. – 312 с.
3. *Вайсбурд И.А.* Развиваем логическое мышление. – М.: Эксмо, 2012. – 482 с.
4. *Ездаков А.Л.* Функциональное и логическое программирование. – М.: Бином. Лаборатория знаний, 2009. – 120 с.
5. Информатика. Задачник-практикум: в 2 т. / Под ред. И.Г. Семакина, Е.К. Хеннера: Т. 2. – М.: Бином. Лаборатория знаний, 2003. – 278 с.: ил.
6. Информатика: учеб. пособие для студ. пед. вузов / А.В. Могилев, Н.И. Пак, Е.К. Хеннер / Под ред. Е.К. Хеннера. – 3-е изд., перераб. и доп. – М.: Издательский центр “Академия”, 2004. – 848 с.
7. *Каймин В.А.* Информатика: учебник. – 2-е изд., перераб. и доп. – М.: ИНФРА-М, 2001. – 272 с.
8. *Козырева Г.Ф.* Практикум решения задач по курсу «Основы искусственного интеллекта»: учебно-методическое пособие для студентов, обучающихся по специальности «информатика». – Армавир, 2005.
9. *Леонтьев В.* Развиваем логическое мышление. – Айрис-Пресс, 2013. – 293 с.
10. *Рассел Д.* Логическое программирование. – М., 2012. – 134 с.
11. *Рублев В.С.* Языки логического программирования: учебное пособие. – М.: Интернет-Университет Информационных Технологий, 2008.
12. *Сергиевский Г.М., Волченков Н.Г.* Функциональное и логическое программирование. – М.: Академия, 2010. – 320 с.
13. *Цуканова Н.И., Дмитриева Т.А.* Логическое программирование на языке Visual Prolog. – Горячая Линия. Телеком, 2008. – 144 с.
14. *Шрайнер П.А.* Основы программирования на языке Пролог. – М.: Интернет-Университет Информационных Технологий, 2005.
15. http://www.mari-el.ru/mmlab/home/prolog/LECTION10/index.html
16. https://infopedia.su/1xa58a.html

СОДЕРЖАНИЕ

Учебное издание

Елена Викторовна Игонина

ОСНОВЫ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ И РЕАЛИЗАЦИЯ ПРОГРАММ НА ЯЗЫКЕ ПРОЛОГ

Учебное пособие

Лицензия на издательскую деятельность
ИД № 06146. Дата выдачи 26.10.01.
Формат 60 x 84 /16. Гарнитура Times. Печать трафаретная
Печ.л. 5,8 Уч.-изд.л. 5,4
Тираж 300 экз. (1-й завод 1-20 экз.). Заказ 122

Отпечатано с готового оригинал-макета на участке оперативной полиграфии
Елецкого государственного университета им. И.А. Бунина
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Елецкий государственный университет им. И.А. Бунина»
399770, г. Елец, ул. Коммунаров, 28,1