В. Н. Пичугин, Р. В. Фёдоров

Рекурсивно-логическое программирование

Лабораторный практикум.

ЧАСТЬ 1. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА ТУРБО-ПРОЛОГ.

Турбо-Пролог - это осуществленная компанией Borland International реализация языка программирования высокого уровня Пролог компиляторного типа. Ее отличает большая скорость компиляции и счета. Турбо-Пролог предназначен для выдачи ответов, которые он логически выводит при посредстве своих мощных внутренних процедур.

Главное меню Турбо_Пролога высвечивает 7 доступных пользователю опций (команд) в верхней части экрана:

1.Запуск программы на счет (Run).
2.Трансляция программы (Compile).
3.Редактирование текста программы (Edit).
4.Задание опций компилятора (Options).
5.Работа с файлами (Files).
6.Настройка системы (Setup).
7.Выход из системы (Quit).

Язык программирования Пролог (PROgramming LOGic) предполагает получение решения задачи при помощи логического вывода из ранее известных фактов. Программа на языке Пролог не является последовательностью действий – она представляет собой набор фактов и правил, обеспечивающих получение логических заключений из данных фактов. Поэтому Пролог считается *декларативным* языком программирования.

Пролог базируется на *фразах (предложениях) Хорна*, являющихся подмножеством формальной системы, называемой *логикой предикатов.*

Пролог использует упрощенную версию синтаксиса логики предикатов, он прост для понимания и очень близок к естественному языку.

Пролог имеет механизм вывода, который основан на сопоставлении образцов. С помощью подбора ответов на запросы Пролог извлекает хранящуюся информацию. Пролог пытается ответить на запрос, запрашивая информацию, о которой уже известно, что она истинна.

Одной из важнейших особенностей Пролога является то, что он ищет не только ответ на поставленный вопрос, но и все возможные альтернативные решения. Вместо обычной работы программы на процедурном языке от начала и

до конца, Пролог может возвращаться назад и просматривать все остальные пути при решении всех частей задачи.

Программист на Прологе описывает *объекты* и *отношения*, а также *правила,* при которых эти отношения являются истинными.

Например, предложение

Вася любит собак,

устанавливает отношение между объектами *Вася* и *собаки,* эти отношением является *любит*.

На языке Пролог это отношение называется *предикатом* и может быть записано в следующем виде:

любит (вася, собаки).

Правило, определяющее истинность предыдущего отношения может прдставлено предложением

Вася любит собак, если собаки не кусаются.

На языке Пролог это предложение может выглядеть следующим образом:

любит (вася, собаки):-не_кусаются(собаки).

Задав в программе несколько фактов и правил, мы может задавать Прологу вопросы, касающиеся заданных отношений. Так запрос «любит ли Вася собак?» на языке Пролог будет выглядеть так:

любит (вася, собаки).

В данном случае Пролог ответит «да» («yes»), потому что есть факт, подтверждающий это. Если мы хотим спросить «что любит Вася?», то на языке Пролог это можно сделать так:

любит (вася, What), где слово What есть имя переменной языка Пролог.

В Прологе переменные начинаются с заглавной буквы, а константы (имена собственные – с прописной).

Программа, написанная на языке Пролог состоит из пяти основных разделов: раздел описания доменов (типов объектов), раздел внутренней базы данных, раздел описания предикатов, раздел описания предложений и раздел описания цели. Ключевые слова *domains, facts(database), predicates, clauses и goal* отмечают начала соответствующих разделов. Назначение этих разделов таково:

- раздел *domains* содержит определения доменов, которые описывают различные типы объектов, используемых в программе, если используются стандартные типы, то раздел может не использоваться;
- раздел *facts(database)* содержит описание предикатов динамической (внутренней) базы данных, которые являются предикатами базы данных и могут быть изменены в процессе работы программы без перекомпиляции, если программа такой базы данных не требует, то этот раздел может быть опущен;
- раздел *predicates* служит для описания используемых программой предикатов, этот раздел является обязательным;

- в раздел *clauses* заносятся факты и правила статической базы данных, которая и является собственно программой, этот раздел является обязательным;
- в разделе *goal* на языке Пролога формулируется цель(запрос) созданной программы. Составными частями при этом могут являться некие подцели, из которых формируется единая цель программы, этот раздел является обязательным.

Пролог имеет основные шесть встроенных типов доменов:

Тип данных	Ключевое слово	Диапазон значений	Примеры использования
Символы	char	Все возможные символы	‘a’, ’b’, ‘#’, ‘B’, ‘%’
Целые числа	integer	От –32768 до 32767	-63, 84, 2349, 32763
Действительные числа	real	От +1E-307 до +1E308	360, - 8324, 1.25E23, 5.15E-9
Строки	string	Последовательность символов (не более 250)	«today», «123», «school_day»
Символические имена	symbol	1.Последовательность букв, цифр, символов подчеркивания; первый символ – строчная буква. 2. Последовательность любых символов, заключенная в кавычки.	flower, school_day «string and symbol»
Файлы	file	Допустимое в DOS имя файла	mail.txt, LAB.PRO

Если в программе необходимо использовать новые домены данных, то они должны быть описаны в разделе *domains*.

Пример 1:

```
domains
number=integer
name, person=symbol.
```

Предикаты описываются в разделе *predicates*. Предикат представляет собой строку символов, первым из которых является строчная буква. Предикаты могут не иметь аргументов, например «go» или «repeat». Если предикаты имеют аргументы, то они определяются при описании предикатов в разделе predicates:

Пример 2:

```
predicates
mother (symbol, symbol)
```

father (symbol, symbol).

Факты и правила определяются в разделе *clauses,* а вопрос к программе задается в разделе *goal.* Для запуска программ можно использовать утилиту среды визуальной разработки *Visual Prolog – TestGoal*. Подробнее о запуске программ написано в разделе 6.

Основные понятия языка Турбо-Пролог.

Турбо-Пролог - это декларативный язык, программы на котором содержат объявления логических взаимосвязей, необходимых для решения задачи.В Турбо_Прологе рассматриваются отношения между утверждениями и объектами, характерные для логики предикатов.

В программах на Прологе существует три типа предложений (clauses): факт, правило вывода, цель.Каждое предложение должно заканчиваться точкой. Факт - утверждение, истинность которого безусловна.Например,

```
likes(mary,apples).   /* Мэри любит яблоки */
```

или

```
male(bob)            /* Боб - мужчина */
parent(bob,ann).     /* Боб - родитель Энн */
```

Правило - утверждение, зависящее от условий.Например,

```
child(ann,bob) :- parent(bob,ann). /* Энн - дитя Боба,
                                      если Боб - родитель Энн */
```

или

```
father(X,Y) :-parent(X,Y),male(X  )./* Для всех X и Y
                                       X является отцом Y,если
                                       X является родителем Y и
                                       X - мужчина */
```

Цель - вопрос пользователя к системе о том, какие утверждения являются истинными.

Для указанных выше примеров на вопрос

```
child(ann,bob)  /* является ли Энн ребенком Боба ?*/
```

будет выдан ответ

```
true  /* истина */,
```

а на вопрос

```
father(X,ann)  /* кто является отцом Энн ? */
```

будет выдан ответ

```
X = Bob       /* отцом Энн является Боб */.
```

На все поставленные вопросы Пролог пытается ответить с помощью фактов и правил вывода. Он решает задачу, просматривая программу сверху вниз и слева направо. Сначала анализируется цель и ведется поиск такого факта или правила вывода, с помощью которого она может быть достигнута. При нахождении такого факта после соответствующей подстановки переменных Пролог переходит к анализу следующей цели при условии, что предыдущая

достигнута (доказана). Если некоторая цель последняя, доказательство заканчивается. При отсутствии нужных фактов, но наличии правила вывода, которое могло быть применено, цель заменяется условием этого правила с соответствующей подстановкой переменных. Теперь условием выполнения цели становится доказательство условия (правой части) правила вывода. Процесс нахождения соответствия между целью и фактом или правилом называется у н и ф и к а ц и е й. В ходе унификации Пролог ищет все альтернативные решения.

Программа Турбо-Пролога включает определенные разделы, не все из которых являются обязательными:

```
domains
   /*(домены) - раздел объявлений*/;
database
   /* описания предикатов динамической базы данных */
predicates
  /* описания предикатов */
goal
  /* целевое утверждение */
clauses
  /* утверждения - факты и правила */
```

В программе по крайней мере должны быть разделы predicates и clauses.

Раздел domains напоминает объявление данных в традиционных (императивных) языках, например таких, как Паскаль и Си. Существуют следующие типы доменов:

char (символьный),
integer (целый),
real (вещественный),
string (строковый),
symbol (для цепочки из букв, цифр и символов подчеркивания с первой строчной буквой либо цепочки знаков в двойных кавычках),
file (файловый).

По отношению к именам объектов (идентификаторам) в Прологе используются следующие правила :

1) имя может включать латинские буквы, цифры и символ подчеркивания, причем первым символом не должна быть цифра;
2) имена символических констант должны начинаться со строчной буквы;
3) в имени можно использовать одновременно и строчные и прописные буквы.

Пример работающей программ.

Задание:

Родственные отношения.

Кроме родственных отношений parent (родитель) и ancestor (предок) программа должна содержать хотя бы одно из следующих отношений:
brother (брат);
sister (сестра);
grand-father (дедушка);
grand-mother (бабушка);
uncle (дядя);

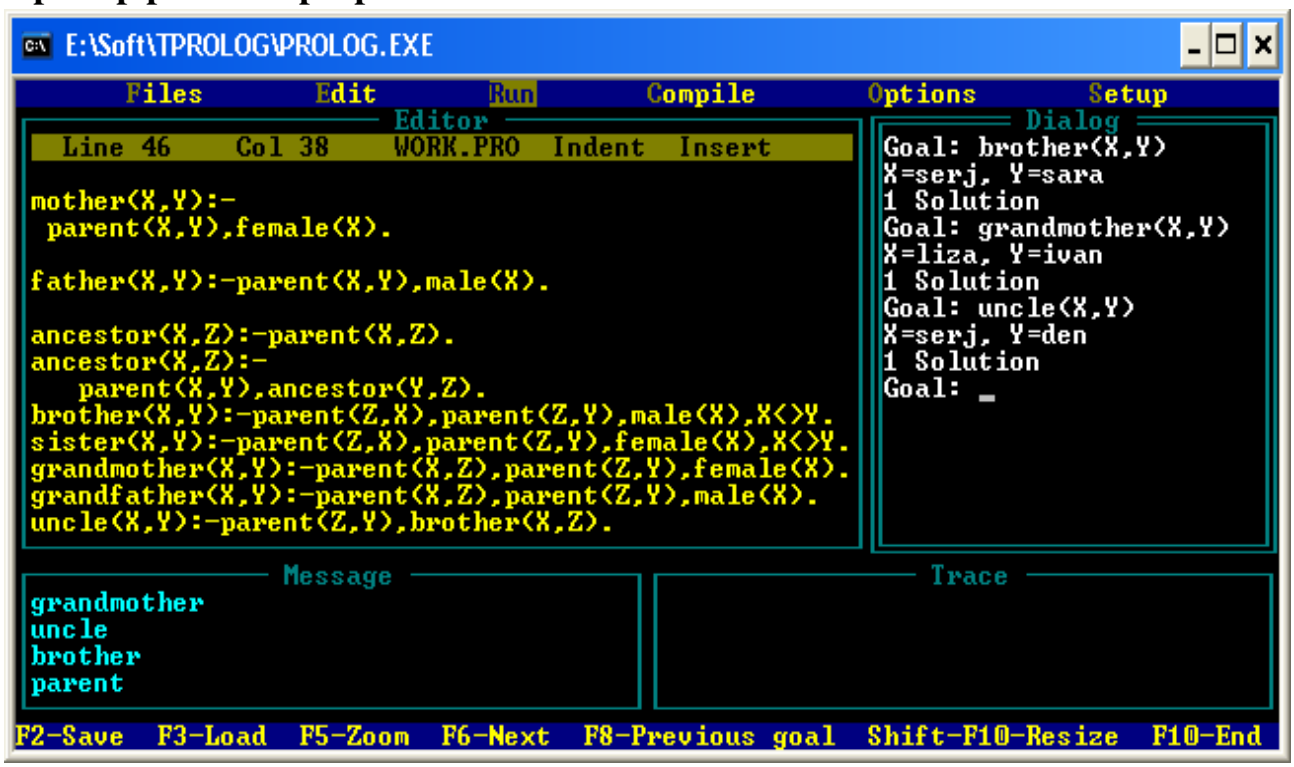
Текст программы:

```
domains
        s=symbol
predicates
parent(s,s)
female(s)
male(s)
mother(s,s)
father(s,s)
ancestor(s,s)
child(s,s)
brother(s,s)
sister(s,s)
grandmother(s,s)
grandfather(s,s)
uncle(s,s)
clauses
parent(tom,bob).
parent(liza,tony).
parent(king,serj).
parent(bob,pat).
parent(king,sara).
parent(jim,tony).
parent(tony,ivan).
parent(sara,den).
female(liza). female(sara).
male(tom). male(dan).
male(bob). male(jim). male(tony). male(king).
male(serj). male(pat). male(ivan).
child(Y,X):-parent(X,Y).
mother(X,Y):-
 parent(X,Y),female(X).
 father(X,Y):-parent(X,Y),male(X).
```

```
ancestor(X,Z):-parent(X,Z).
ancestor(X,Z):-
  parent(X,Y),ancestor(Y,Z).
brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X<>Y.
sister(X,Y):-parent(Z,X),parent(Z,Y),female(X),X<>Y.
grandmother(X,Y):-parent(X,Z),parent(Z,Y),female(X).
grandfather(X,Y):-parent(X,Z),parent(Z,Y),male(X).
uncle(X,Y):-parent(Z,Y),brother(X,Z).
```

Описание брата. X будет братом Y если у них есть общий родитель, X – мужчина и X не равен Y, т. е. X не читается братом самому себе.
Описание бабушки. X будет бабушкой Y если она является родителем некой Z, которая в свою очередь является родителем Y, при этом X – женщина.
Описание дяди. X будет дядей для Y, если некая Z будет родителем для Y и X является братом для Z.

Пример работы программы:



Лабораторная работа N 1

ВЫПОЛНЕНИЕ ПРОГРАММ НА ПРОЛОГЕ.

Цель работы - ознакомление со структурой программ и синтаксом языка Пролог; построение простейшей интеллектуальной вопросно-ответной системы.

Синтаксис языка Турбо-Пролог и структура программы

Программа на Турбо-Прологе имеет следующую обобщенную структуру:

```
domains
  /* ...
        объявление доменов
        ... */
predicates
  /* ...
        объявление предикатов
        ... */
goal
  /* ...
        подцель_1, подцель_2, и т.д.
        ... */
clauses
  /* ...
        предложения (факты и правила)
        ...*/
```

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

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

```
predicates
  predicateName (argument_type1, argument_type2,...,argument_typeN)
```

Здесь ***argument_type1, ..., argument_typeN*** - либо стандартные домены, либо домены, объявленные в секции ***domains***. Объявление домена аргумента и описание типа аргумента - суть одно и то же.

В секции ***domains*** объявляются любые нестандартные домены, используемые для аргументов предикатов. Домены в Прологе являются аналогами типов в других языках. Основными стандартными доменами Турбо-Пролога являются: ***char***, ***integer***, ***real***, ***string*** и ***symbol***. Основная форма объявления доменов имеет следующий вид:

```
domains
  argument_type1, ..., argument_typeN - <стандартный домен>
  argument_1, ..., argument_N         - <составной домен 1>;
                                        <...>;
                                        <составной домен N>;
```

В секции ***goal*** задается внутренняя цель программы; это позволяет программе запускаться независимо от среды разработки. Если внутренняя цель включена в программу, то Турбо-Пролог выполняет поиск только первого решения, и связываемые с переменными значения не выводятся на экран. Если внутренняя цель не используется, то в процессе работы в диалоговом окне будет вводиться внешняя цель. При использовании внешней цели Турбо-Пролог ищет все решения и выводит на экран все значения, связываемые с переменными.

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

Арность (размерность) предиката - это число принимаемых им аргументов; два предиката с одним именем могут иметь различную арность. Предикаты с различными версиями арности должны собираться вместе, причем и в секции ***predicates*** и в секции, ***clauses***; однако предикаты с различной арностью рассматриваются как абсолютно разные.

Правила имеют форму:

ЗАГОЛОВОК: - <Подцель1>, <Подцель2>, ..., <ПодцельN>

Для разрешения правила Пролог должен разрешить все его подцели, создав при этом соответствующее множество связанных переменных. Если же одна из подцелей ложна, Пролог возвратится назад и просмотрит альтернативные решения предыдущих подцелей, а затем вновь пойдет вперед, но с другими значениями переменных. Этот процесс называется "поиск с возвратом".

Рассмотрим программу CH05EX03.PRO, которая содержит сведения об именах и возрастах нескольких игроков в теннисном клубе.

```
                                /* Program CH05EX03.PRO */
domains
  child = symbol
  age = Integer
predicates
```

```
  player(child, age)
clauses
  player(peter,9).
  player(paul,10).
  player(chris,9).
  player(susan,9}.
```

Поиск всех возможных пар девятилетних игроков может быть осуществлен путем задания следующей составной цели:

player(Person1,9), player(Person2,9), Person1 <> Person2.

(На естественном языке это прозвучало бы так: найти Лицо1 в возрасте 9 лет и Лицо2 в возрасте 9 лет, отличное от Лица1.)

1. Турбо-Пролог попытается найти решение для первой подцели player(Person1,9) и перейдет к следующей подцели только после того, как первая подцель будет достигнута. Первая подцель согласуется сопоставлением Person1 с peter. Теперь Турбо-Пролог может попытаться согласовать следующую подцель:

player(Person2,9)

Опять же Person2 сопоставляется с peter. Теперь Пролог переходит к третьей и последней подцели:

Person1 <> Person2

2. Так как и Person1, и Person2 связаны с peter, эта подцель не выполняется. Вследствие этого Турбо-Пролог выполняет поиск с возвратом к предыдущей подцели и ищет другое решение для второй подцели

player(Person2,9)

Эта подцель выполняется при сопоставлении Person2 с chris.

3. Теперь третья подцель

Person1 <> Person2

может быть выполнена, так как peter и chris отличны. Таким образом, целевое утверждение полностью согласовано путем образования турнирной пары из Криса и Питера.

4. Однако, поскольку Турбо-Пролог должен найти все возможные решения целевого утверждения, он находит точку поиска с возвратом предыдущей цели. Так как

player(Person2,9)

также может быть согласовано, если принять Person2 за susan, то Турбо-Пролог еще раз проверяет третью подцель. Достигается успех (так как peter отличен от susan), и другое решение для всего целевого утверждения найдено.

5. В дальнейшем поиске решений Турбо-Пролог вновь возвращается к точке поиска с возвратом второй подцели, но все возможности для этой подцели уже

исчерпаны. Вследствие этого поиск с возвратом теперь выполняется от первой подцели. Она вновь может быть согласована сопоставлением Person1 с chris. Вторая подцель теперь имеет успех в результате сопоставления Person2 с peter, так что третья подцель согласована, и все целевое утверждение опять выполнено.

6. В поисках еще одного решения целевого утверждения Турбо-Пролог возвращается к точке поиска с возвратом второй подцели в правиле. Здесь Person2 ставится в соответствие chris и при этом условии проверяется третья подцель. Она не выполняется, так как Person1 и Person2 эквивалентны, и тогда выполняется поиск с возвратом от второй подцели в поисках другого решения. Person2 сопоставляется с susan, и теперь третья подцель выполняется (Крис и Сюзан).

7. И вновь Пролог возвращается ко второй подцели, но на этот раз безуспешно. Когда вторая подцель не выполняется, процесс возвращается к первой подцели, на этот раз находя соответствие Person1 с susan. Пытаясь выполнить вторую подцель, Пролог сопоставляет Person2 с peter, и впоследствии третья подцель выполнится при этих условиях. Итак, найдено пятое решение.

8. Снова осуществляется возврат ко второй подцели, где Person2 сопоставляется с chris. Найдено шестое решение задачи, и получено полное множество пар.

9. Последнее исследуемое решение связывает с susan как Person1, так и Person2. Так как это приводит к невыполнению последней подцели, Турбо-Пролог должен вернуться ко второй подцели, но там не осталось никаких новых вариантов. Тогда Турбо-Пролог возвращается для поиска к первой подцели, но все возможности для Person1 уже исчерпаны. Для данного целевого утверждения не может быть найдено других решений, и работа программы завершается.

Таким образом, после ввода следующей составной цели для программы CH05EX03:

player(Person1,9), player(Person2,9), Person1 <> Person2.

Турбо-Пролог ответит следующим:

```
Person1=peter, Person2=chris
Person1=peter, Person2=susan
Person1=chris, Person2=peter
Person1=chris, Person2=susan
Person1=susan, Person2=peter
Person1=susan, Person2=chris
6 Solutions Goal :_
```

Задание к лабораторной работе

1. Провести тестирование программы CH05EX03.PRO (см.прил.1).

2. Дописать текст программы LAB01.PRO (объявить домены и предикаты и задать факты - предикаты male, female, mother, father) таким образом, чтобы удовлетворялись все запросы вида brother(X,Y); sister(X,Y); uncle(X.Y); grandfather(X,Y).

3. Дописать определение отношения grandmother(X,Y) и дополнить базу фактов, таким образом, чтобы удовлетворялся запрос grandmother(X,Y).

Порядок выполнения задания

1. Загрузить Турбо-Пролог.
2. Загрузить текст программы CH05EX03.PRO.
3. Ввести цели player(X,9); player(X,10); player(peter,X); player(X1,9),player(X2,9).X1 <> X2 и убедиться в правильности работы программы.
4. Загрузить текст LAB01.PRO (см. прил.1).
5. Внести требуемые изменения.

Содержание отчета

Отчет должен содержать полученный текст программы и результаты запросов.

Рекомендуемая литература

1. Доорс Дж., Рейблейн А.Р., Вадера С. Пролог - язык программирования будущего /Пер. с англ. - М.: Финансы и статистика, 1990.
2. Ин Ц., Соломон Д. Использование Турбо-Пролога /Пер. с англ. - М.: Мир, 1993.

Лабораторная работа N 2

ПРОГРАММИРОВАНИЕ ИТЕРАЦИЙ

Цель работы - ознакомление с предопределенными предикатами ввода/вывода, неудачного завершения и отсечения; изучение метода программирования итераций; реализация на Прологе итерационного процесса.

ПОВТОР И РЕКУРСИЯ

В Прологе отсутствует возможность непосредственного задания итерационного процесса вычислений, т.е. не реализованы синтаксические конструкции типа ***FOR***, ***WHILE*** или ***REPEAT***. Существует непрямой способ выражения повтора. Пролог позволяет только два вида повторения - возврат, в котором осуществляется поиск многих решений в одном запросе, и рекурсию, в которой процедура вызывает сама себя. Фактически, Турбо-Пролог распознает специальный случай рекурсии - хвостовую рекурсию - и компилирует ее в итерацию на машинном языке.

Итерация реализуется следующим образом. Когда выполняется процедура поиска с возвратом, происходит поиск другого решения целевого утверждения. Это осуществляется путем возврата к последней основной, имеющей альтернативное решение, подцели, реализации альтернативы и очередного возврата. Поэтому поиск с возвратом можно использовать для выполнения повторяющихся процессов. Например, определив предикат с двумя предложениями.

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

```
/* Программа CH07EX02.PRO*/
predicates
    repeat
    typewriter
clauses
    repeat.
    repeat :- repeat.

    typewriter :-
        repeat,
        readchar(C),/*Читать символ, его значение присвоить C*/
        write(C),
        char_int(C,13). /* C равно 13 в коде ASCII? */
```

Программа CH07EX02.PRO показывает, как работает repeat. Правило ***typewriter*** :- ... описывает процесс приема символов с клавиатуры и отображения их на экране, пока пользователь не нажмет клавишу Enter (код 13 в ASCII).

Предикат ***typewriter*** определяет следующую последовательность действий.

1. Выполняется ***repeat*** (который ничего не делает).
2. Переменной **C** присваивается значение символа.
3. Символ **C** отображается.
4. Проверяется, соответствует ли **C** коду 13.
5. Если соответствует, то завершение. Если нет - возврат и поиск альтернативы. Так как ни ***write***, ни ***readchar*** не являются альтернативами, постоянно происходит возврат к ***repeat***, который всегда имеет альтернативные решения.
6. Считывается следующий символ, отображается и проверяется на соответствие, коду 13 в ASCII.

Следует заметить, что **C** теряет свое значение после возврата и выполнения предиката **readchar(C)**. Этот предикат и сбрасывает значение переменной **C**. Такой тип освобождения переменной существенен, когда поиск с возвратом применяется для определения альтернативных решений целевого утверждения, но не эффективен при использовании поиска с возвратом в других целях. Смысл в том, что хотя поиск с возвратом и может повторять операции сколько угодно раз, он не способен "запомнить" что-либо из одного повторения для другого. Все переменные теряют свои значения, когда обработка возвращается и проходит те шаги, на которых эти значения устанавливались. Это наиболее простой способ сохранения значений списка, общих результатов или любых других значений при выполнении циклов повторения.

Условие выхода из цикла может определяться любым предикатом, одно из двух альтернативных описаний которого должно содержать предопределенный предикат ***fail*** ("неудачное завершение"). Его использование означает: "Решение поставленного целевого утверждения не было достигнуто, поэтому - вернуться назад и искать альтернативное решение". Предикат ***fail*** всегда вызывает режим "неудачное завершение", но можно вызвать поиск с возвратом и обратившись, например, к такому целевому утверждению, как 4 <> 2+2.

Кроме предикатов ***readchar*** и ***char_int***, называемых предопределенными или стандартными предикатами (они могут пониматься как стандартные процедуры или функции в алгоритмических языках программирования), Турбо-Пролог содержит еще 90 таких предикатов, из которых для программирования ввода/вывода могут использоваться следующие:

- ***readint*** - читает целое число с текущего устройства ввода и связывает его с заданной переменной;

- ***readln*** - читает, строку с текущего устройства ввода и связывает его с заданной переменной (в качестве конца строки используется возврат каретки CR);

- ***readreal*** - читает действительное число с текущего устройства ввода и связывает его с заданной переменной;

- ***nl*** - осуществляет перевод строки, такой, как при посылке на текущее устройство вывода символа возврата каретки (CR).

Для предотвращения бектрекинга (возврата к перебору очередных альтернатив после первой найденной) используется предикат отсечения ***cut***, который записывается как восклицательный знак - **!**.

Таким образом, если предикат ***fail*** инициирует бектрекинг, то предикат ***cut*** его завершает.

Задание к лабораторной работе.

1. Провести тестирование программы LAB02.PRO (см. прил.2).
2. Изменить программу LAB02.PRO так, чтобы она воспринимала целые числа, выводила их на экран и завершалась при вводе числа 0.
3. Модифицировать программу LAB02.PRO так, чтобы она воспринимала два десятичных числа и выводила их на экран.
4. Модифицировать программу LAB02.PRO таким образом, чтобы она вычисляла сумму двух введенных десятичных чисел и выдавала эту сумму на экран. Программа должна завершаться, если одно из двух вводимых чисел равно 0.

Порядок выполнения задания.

1. Загрузить Турбо-Пролог.
2. Загрузить программу LABQ2.PRO и убедиться в правильности ее работы.
3. Внести требуемые изменения.

Содержание отчета.

Отчет должен содержать полученные тексты программ и результаты их работы.

Рекомендуемая литература.

1. Братко И. Программирование на языке Пролог для искусственного интеллекта /Пер. с англ. - М.: Мир, 1990.
2. Янсон А. Турбо-Пролог в сжатом изложении /Пер. с нем. - М.: Мир, 1991.
3. Малпас Дж. Реляционный язык Пролог и его применение /Пер. с англ. - М.: Мир, 1990.

Лабораторная работа N 3.

ПРОГРАММИРОВАНИЕ МНОГООКОННОГО ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Цель работы - ознакомление с простейшими предопределенными предикатами, реализующими многооконный интерфейс; реализация на Прологе простейшей программы вычислений с использованием многооконного интерфейса.

УПРАВЛЕНИЕ ЭКРАНОМ В ПРОГРАММАХ НА ПРОЛОГЕ

Турбо-Пролог имеет набор предикатов управления экраном, содержащий поддержку различных программ работы с экраном. Ниже рассматриваются только простейшие предикаты, которых, однако, достаточно для большинства программ.

Окно - часть площади экрана, возможно, ограниченная рамкой. Окно создается с помощью предиката makewindow, в котором задаются координаты левого верхнего угла окна и количество строк и столбцов, которые займет окно. Когда окно создано, оно становится активным (текущим), и вся выводимая информация автоматически направляется в него. Изменить направление ввода /вывода можно при помощи предиката shiftwindow. Все предикаты управления экраном, описанные ниже, работают в активном окне. Например, при обращении к предикату cursor для задания положения курсора в 4-й строке и 15-м столбце он будет помещен в 4-й строке и 15-м столбце текущего окна. Каждое окно имеет определенную позицию курсора, которая запоминается при переходе к другому окну. При удалении окна с помощью предиката removewindow содержимое экрана "за окном" автоматически восстанавливается.

ПРЕДОПРЕДЕЛЕННЫЕ ПРЕДИКАТЫ

Предикат ***makewindow*** создает новое окно на экране; имеет формат:

makewindow(WNo,ScrAttr,FrAttr,Heading,Row,Col,Height,Width)

Описания аргументов этого предиката приведены в табл. 3.1, атрибуты цветов образуются из значений, приведенных в табл. 3.2.

Вычисление величин атрибутов производится путем сложения цвета букв и цвета фона. Для задания мигающего изображения добавляется 128. Например, для букв желтого цвета на красном фоне значение атрибута будет 64+14, т.е. 78.

При определении окно заполняется цветом фона, и курсор помещается в его правый верхний угол.

Makewindow может быть использовано и со свободными параметрами, при этом возвращаются значения, соответствующие текущему окну. Когда создано несколько перекрывающихся окон, активное окно видимо полностью (находится "сверху").

Таблица 3.1

Аргумент	Тип	Описание
Wno	Целый	Используется предикатами работы с окнами как ссылка на большое окно. Его можно использовать несколько раз при определении окон. При этом обращение с помощью этой ссылки происходит только к последнему из окон, созданных этой ссылкой.
ScrAttr	Целый	Атрибут цвет окна.
FrAttr	Целый	Атрибут цвета рамки окна. Если он равен 0, то окно без рамки.
Heading	Строка	Текст заголовка окна.
Row,Col	Целое	Определяет левый верхний угол окна.
Height	Целый	Высота окна, включая рамку.
Width	Целый	Ширина окна, включая рамку.

Таблица 3.2

Цвет букв	Величина	Цвет фона	Величина
Черный	0	Черный	0
Серый	8	Голубой	16
Голубой	1	Зеленый	32
Светло-голубой	9	Синий	48
Зеленый	2	Красный	64
Светло-зеленый	10	Алый	80
Синий	3	Коричневый	96
Светло-синий	11	Белый	112
Красный	4		
Светло-красный	12		
Алый	5		
Светло-алый	13		
Коричневый	6		
Желтый	14		
Белый	7		

Например, Makewindow(l, 7, 136, "My first window", 1, 20, 4, 28).определяет окно номер 1, черно-белое (***SorAttr*** равно 7). Окно будет иметь рамку (***FrAttr*** равно 135) и заголовок "My first window". Само окно будет, иметь 4 строки в высоту и 23 позиции в ширину, и его верхний правый угол будет расположен в 20-й позиции 1-й строки экрана. Заметьте, что строки и позиции нумеруются числами 0,1,2,... и т.д., и рамка занимает 2 строки и 2 столбца.

Предикат ***shiftwindow*** осуществляет активизацию другого окна и имеет формат:

shiftwindow(WindowNo)

Окно с номером ***WindowNo*** будет активизировано на экране, причем курсор будет помещен в ту позицию, в которой он был в момент предыдущего обращения к окну. Если ***WindowNo*** свободно, то ***shiftwindow*** свяжет ***WindowNo*** с номером текущего окна. Если ***WindowNo*** связано с номером еще не определенного окна, возникнет ошибка при исполнении программы.

Предикат ***clearwindow*** очищает текущее окно:

clearwindow - /* (no arguments) */

При этом окно заполняется цветом фона, и курсор помещается в верхнюю левую позицию окна.

Предикат ***removewindow*** удаляет активное в этот момент окно:

removewindow /* (no arguments) */

При этом активным становится окно, активизированное перед удалением. Если окна не определены, то возникает ошибка при исполнении.

Предикат cursor выполняет две задачи: помещает курсор в указанную позицию окна или (если аргументы свободны) связывает аргументы с параметрами текущей позиции курсора:

cursor(Row, Col) /* (i,i), (o,o) */

Если ***Row*** и ***Col*** связаны с целыми неотрицательными числами, то после обработки предиката курсор помещается в позицию, определенную координатами ***(Row, Col)***, причем левый верхний угол окна соответствует (0,0). Если ***Row*** и ***Col*** свободны, то они после обработки предиката принимают значения, соответствующие текущему положению курсора. Если ***Row*** и ***Col*** указывают на позицию за пределами окна (или отрицательны), возникает ошибка.

Например, программа LAB03.PRO (см. прил.3) использует окна, превращая компьютер в простую машину для сложения, которая складывает два вводимых числа и сообщает результат. Оба операнда и результат отображаются в окнах.

Отметим переопределение окна 2 в программе. Новое окно определяется с ссылкой на тот же номер; всегда будет использоваться окно, определенное последним.

Задание к лабораторной работе

1. Провести тестирование программы LAB03.PRO.
2. Изменить программу LAB03.PRO так, чтобы она в специально выделенных окнах выводила сумму, разность и произведение двух введенных в соответствующие окна вещественных чисел. Предикат для считывания вещественных чисел с клавиатуры - ***readreal(Number)***.

Порядок выполнения задания.

1. Загрузить Турбо-Пролог.
2. Загрузить программу LAB03.PRO и убедиться в правильности ее работы. Для запуска программы задайте цель start.
3. Внести требуемые изменения.

Содержание отчета.

Отчет должен содержать полученный текст программы и результаты ее работы.

Рекомендуемая литература.

1. Ин Ц., Соломон Д. Использование Турбо-Пролога /Пер. с англ. - М.: Мир, 1993.
2. Янсон А. Турбо-Пролог в сжатом изложении /Пер. с нем. - М.: Мир, 1991.

Лабораторная работа N 4

ПРОГРАММИРОВАНИЕ ОБРАБОТКИ СПИСКОВ

Цель работы - ознакомление с синтаксисом определения и методами простейшей обработки списков; построение простейших программ обработки списков.

СПИСКИ И РЕКУРСИЯ

Обработка списков, т.е. объектов, которые содержат конечное число элементов, - мощное средство в Прологе.

Списком в Прологе называется объект, который содержит конечное число других объектов. Каждая составляющая списка называется элементом. При оформлении списочной структуры данных элементы списка отделяются запятыми и заключаются в квадратные скобки. Например,

[dog, cat, canary]

["valerie ann", "Jennifer caitlin", "benjamin thomas"]

Элементы списка могут быть любыми, включая другие списки. Однако все элементы списка должны принадлежать одному типу, и декларация ***(domains***) списка имеет вид:

```
domains
elementlist = elements*
elements  = ....
```

где ***elements*** имеют единый тип (например: integer, real или symbol) или являются набором отличных друг от друга элементов, отмеченных разными функторами. Например:

```
elementslist =  elements*
elements  =  i(integer); r(real); s(symbol)
/* функторы здесь i, r и s */
```

Список является рекурсивным составным объектом, Он состоит из двух частей - головы, которой является первый элемент, и хвоста, которым является список, включающий все последующие элементы. Хвост списка - всегда список, голова списка - всегда элемент. Например:

головой [a, b, c] является a,

хвостом [a, b, c] является [b, c]

головой [c] является c

хвостом [c] является [],

где [] есть обозначение пустого списка. Пустой список нельзя разделить на голову и хвост. В Прологе голова от хвоста отделяется вертикальной чертой (|). Например:

[a, b, c] эквивалентно [a| [b, c]]

[a| [b, c]] эквивалентно [a| [b| [c]]]

В одном и том же списке можно использовать оба вида разделителей при условии, что вертикальная черта есть последний разделитель. Так, [a, Ь, c, d] эквивалентно [a, b |[c, d].

Таким образом, например, в результате сопоставления списка [X, Y, Z] и списка ["эгберт", "ест", "мороженое"] переменным будут присвоены следующие значения: X-"эгберт", У-"ест", Z-"мороженое"; результат сопоставления списка [7] и списка [X|Y] есть X-7, Y-[]; списка [1, 2, 3, 4] и списка [X, Y | Z] есть X-1, Y-2, Z-[3,4]; списки [1, 2] и [3 | x] не смогут быть сопоставлены.

Так как список имеет рекурсивную составную структуру данных, алгоритм его обработки также рекурсивен. Для задания такого алгоритма обычно используются два предложения, в первом из которых указывается, что делать с обычным списком (списком, который можно разделить на голову и хвост), во втором, - что делать с пустым списком.

Например, печать элементов списка программируется следующим образом:

```
/*печать элементов списка*/
domains
list = integer* /*или любой другой тип*/
predicates
write_a_list(list)
clauses
write_a_list([]) /*если список пустой - ничего не делать*/
write_a_list([H|T]) :-
/*присвоить H-голова, T-хвост, затем ... */
write(H), nl,
write_a_list(T).
goal
write_a_list([l, 2, 3]).
```

При первом просмотре целевое утверждение ***write_a_llst([l,2,3])*** удовлетворяет второму предложению, при этом H и T получат значения 1 и [2, 3] соответственно. Компьютер напечатает 1 и вызовет рекурсивно ***write_a_list***:

```
wrlte_a_llst([2, 3]). /*это write_a_list(T).*/
```

Этот рекурсивный вызов удовлетворяет второму предложению. На этот раз H-2 и T-[3], так что компьютер печатает 2 и снова рекурсивно вызывает write_a_list с целевым утверждением

```
write_a_list([3])
```

Целевое утверждение подходит под второе предложение с H-3 и T-[]. Компьютер печатает 3 и вызывает

```
write_a_list([]).
```

Этому целевому утверждению подходит первое предложение. Второе предложение не подходит, так как [] нельзя разделить, на голову и хвост. Так что, если бы не было первого предложения, целевое утверждение было бы невыполнимым.

На этом работа программы заканчивается, поскольку цель достигнута.

Подсчет элементов списка может быть выполнен, например, программой вида

```
/* Вычисление длины списка */
domains
list = integer* /* или любой другой тип */
predicates
length_of(list, integer)
clauses
length_of([], 0).
length_of([_|T], L) :-
length_of(T, TailLength),
L = TailLength + 1.
```

Вычисление длины списка с применением хвостовой рекурсии может быть выполнено, например, следующей программой:

```
domains
list = Integer* /* или любой другой тип */
predicates
length_of(list, integer, integer)
clauses
/*************************************************************
 * Если список пуст, присвоить значение второго аргумента *
 * (результат) третьему (счетчик).                          *
 *************************************************************/
length_of([], Result, Result).
/*************************************************************
 * Иначе добавить 1 к счетчику и повторить.          *
 *************************************************************/
length_of([_|T], Result, Counter) :-
NewCounter - Counter +1,
Length_of(T, Result, NewCounter).
goal
length_of([l, 2, 3], L, 0), /* начать со счетчика-0*/ write(L), n1.
```

Данная версия предиката ***length_of*** более сложная и менее логичная, чем предыдущая. Она продемонстрирована лишь для доказательства того, что хвостовые рекурсивные алгоритмы для целевых утверждений (которые,

возможно, требуют различных типов рекурсии) можно найти различными средствами.

Задание к лабораторной работе.

1. Провести тестирование программы LAB04.PRO (прил.4).
2. Изменить программу LAB04.PRO так, чтобы она обеспечивала ввод и вывод списка символов в прямом и обратном порядке. Чтение символа с клавиатуры до символа Esc может быть определено, например, предикатом readsim:

```
readsim(S) :- readchar(S), S. <> 27, write(S,"  ").
readsim(27) :- fall.
```

3. Изменить программу LAB04.PRO так, чтобы она обеспечивала ввод и вывод списка слов в прямом и обратном порядке и вычисление числа слов в этом списке.

Порядок выполнения задания.

1. Загрузить Турбо-Пролог.
2. Загрузить программу LAB04.PRO и убедиться в правильности ее работы.
3. Внести требуемые изменения.

Содержание отчета.

Отчет должен содержать подученные тексты программ и результаты их работы.

Рекомендуемая литература.

1. Братко И. Программирование на языке Пролог для искусственного интеллекта /Пер. с англ. - М.; Мир, 1990.
2. Стобо Д.Ж. Язык программирования Пролог /Пер. с англ. - М.: Радио и связь, 1993.
3. Доорс Дж., Рейблейн А.Р., Вадера С. Пролог - язык программирования будущего /Пер. с англ. - М.: Финансы и статистика. 1990.

ЧАСТЬ 2. ОСНОВЫ РАБОТЫ В VISUAL PROLOG.

Создание TestGoal –проекта для выполнения программ.

Для использования утилиты TestGoal для выполнения программ требуется определить некоторые опции компилятора Visual Prolog. Для этого необходимо выполнить следующие действия:

1. Запустите среду визуальной разработки Visual Prolog.
2. Создайте новый проект: выберите команду Project | New Project, активизируется диалоговое окно Application Expert.
3. Определите базовый каталог и имя проекта, рекомендуется имя в поле Base Directory задать по следующему образцу: C:\student\641\ivanov\TestGoal, а имя в поле Project Name следует определить как TestGoal. Далее следует установить флажок Multiprogrammer Mode и щелкнуть мышью внутри полей Name of.VPR File и Name of.PRJ File. Определите цель проекта: на вкладке Target рекомендуется выбрать следующие параметры: Windows32, Easywin, exe, Prolog.
4. Установите требуемые опции компилятора для созданного TestGoal-проекта, для этого следует активизировать диалоговое окно Compiler Options при помощи команды Options | Project | Compiler Options. Далее откройте вкладку Warnings и выполните следующие действия:

- установите переключатель Nondeterm. Это нужно для того, чтобы компилятор принимал по умолчанию, что все определенные пользователем предикаты – недетерминированные (могут породить более одного решения);
- снимите флажки Non Quoted Symbols, Strong Type Conversion Check, Check Type of Predicates.
- нажмите кнопку OK, чтобы сохранить установки опций компилятора.

Запуск и тестирование программы.

Для проверки правильности настройки системы, создадим простую тестовую программу. Для создания нового окна редактирования можно использовать команду меню File | New. Редактор среды визуальной разработки – стандартный текстовый редактор.

Введите в окне редактирования следующий текст:

Goal write (“Hello world!”),nl.

Это раздел цели программы и для того, чтобы она могла быть выполнена, следует активизировать команду Project | Test Goal или нажать комбинацию клавиш <Ctrl> + <G>.

Результат выполнения программы будет расположен вверху в отдельном окне, которое необходимо закрыть перед тем, как тестировать другую Goal.

Утилита Test Goal интерпретирует Goal как специальную программу, которая компилируется, генерируется в исполняемый файл и Test Goal запускает его на выполнение. Эта утилита внутренне расширяет заданный код Goal, чтобы сгенерированная программа находила все возможные решения и показывала значения всех используемых переменных.

Замечание: утилита Test Goal компилирует только тот код, который определен в активном окне редактора.

Лабораторная работа N 5

ВЫПОЛНЕНИЕ ПРОГРАММ НА VISUAL PROLOG..

1. Задание: Написать программу, демонстрирующую разбиение сложных слов на составляющие слоги.

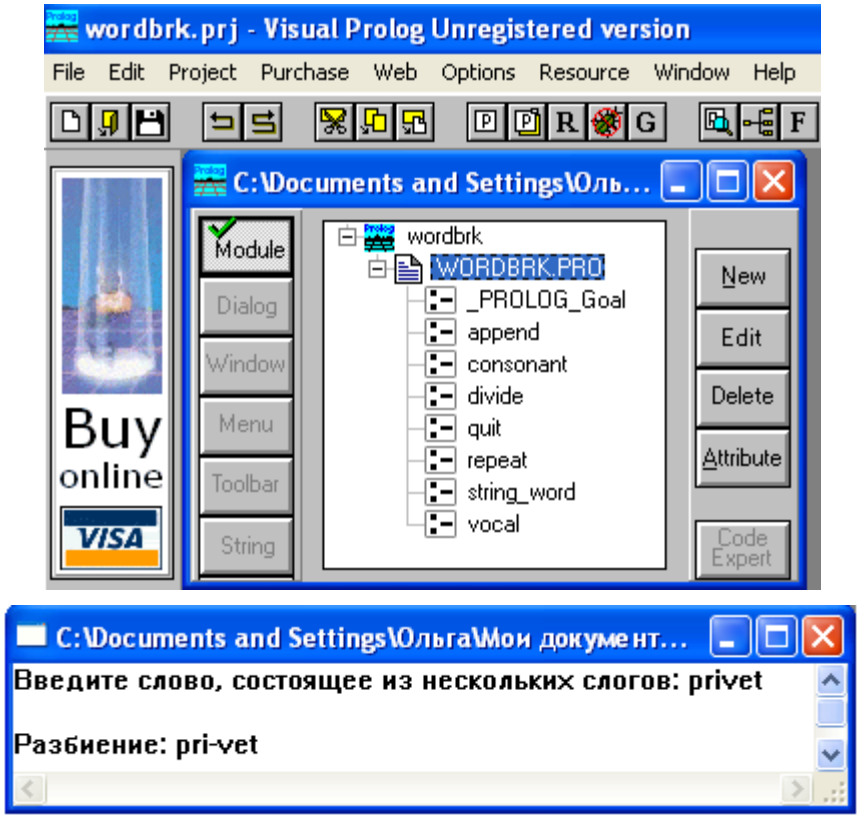
Текст программы:

```
domains
  letter = char
  word_ = letter*
predicates
  nondeterm divide(word_,word_,word_,word_)
  vocal(letter)
  consonant(letter)
  nondeterm string_word(string,word_)
  append(word_,word_,word_)
  nondeterm repeat
  quit(string)
clauses
  divide(Start,[T1,T2,T3|Rest],D1,[T2,T3|Rest]):-
     vocal(T1),
     consonant(T2),
     vocal(T3),
       append(Start,[T1],D1).
  divide(Start,[T1,T2,T3,T4|Rest],D1,[T3,T4|Rest]):-
     vocal(T1),
     consonant(T2),
     consonant(T3),
     vocal(T4),
       append(Start,[T1,T2],D1).
  divide(Start,[T1|Rest],D1,D2):-
     append(Start,[T1],S),
     divide(S,Rest,D1,D2).
```

```
  vocal('a').
  vocal('e').
  vocal('i').
  vocal('o').
  vocal('u').
  vocal('y').
  consonant(B):-
     not(vocal(B)),
     B <= 'z',
     'a' <= B.
  string_word("",[]):-
     !.
  string_word(Str,[H|T]):-
     bound(Str),
     frontchar(Str,H,S),
     string_word(S,T).
  string_word(Str,[H|T]):-
     free(Str),
     bound(H),
     string_word(S,T),
     frontchar(Str,H,S).

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

  repeat.
  repeat:-repeat.

  quit(""):-
     exit,
     !.
  quit(_).
goal
  repeat,
  write("Введите слово, состоящее из нескольких слогов: "),
  readln(S),nl,
  quit(S),
  string_word(S,Word),
  divide([],Word,Part1,Part2),
```

string_word(Syllable1,Part1),
string_word(Syllable2,Part2),
write("Разбиение: ",Syllable1,"-",Syllable2),nl,
fail.

Пример работы программы:



2. Задание: как переправить на другой берег Волка, Козу, Капусту и Человека. В этой программе использованы следующие обозначения Волк(Wolf), Коза(Goat), Человек(Farmer), Капуста(Garbage).

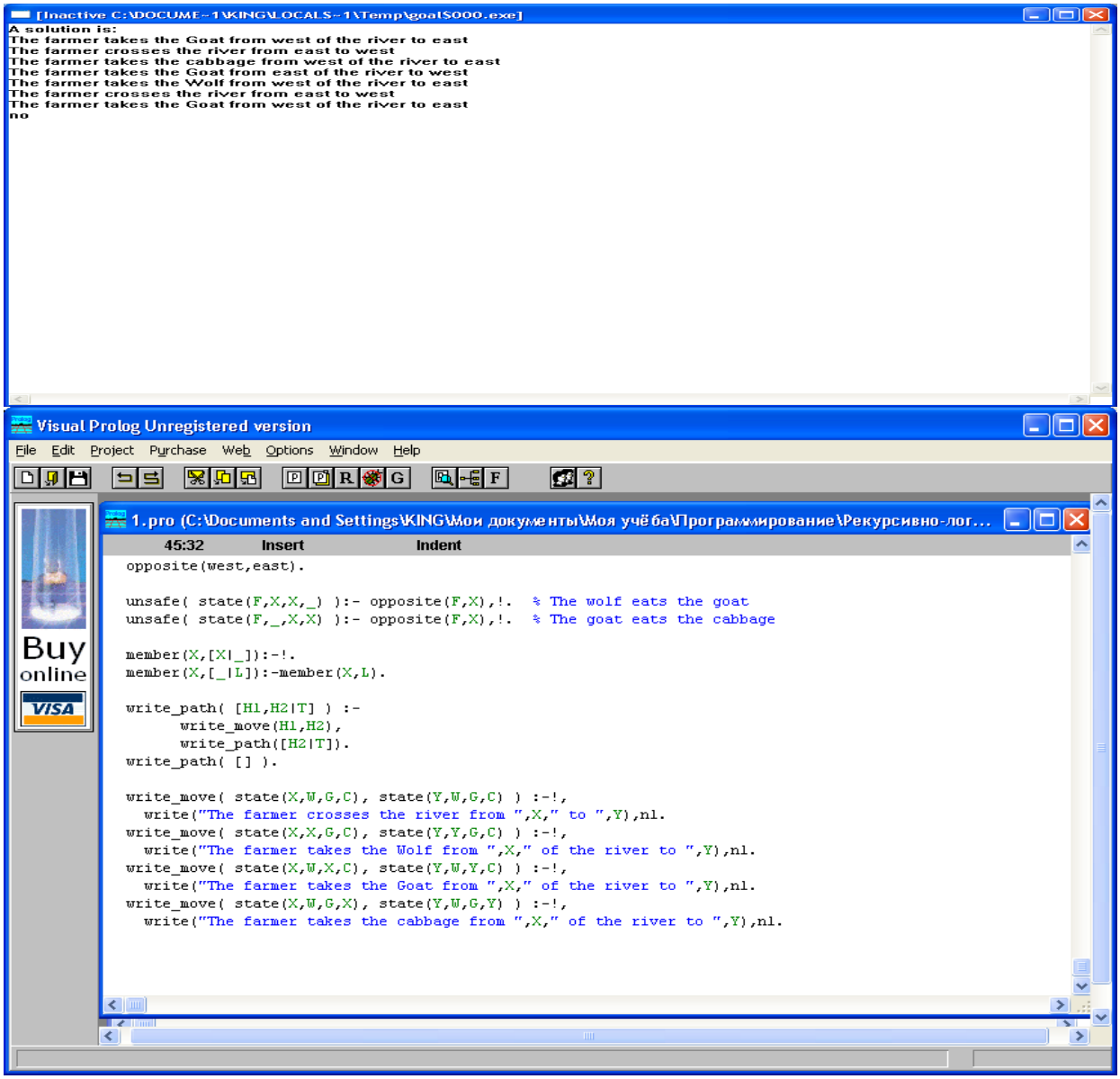
Текст программы:

```
DOMAINS
  LOC   = east ; west
  STATE = state(LOC farmer,LOC wolf,LOC goat,LOC cabbage)
  PATH  = STATE*
  PREDICATES
  go(STATE,STATE)              % Start of the algorithm
  path(STATE,STATE,PATH,PATH)  % Finds a path from one state to another
  nondeterm move(STATE,STATE)  % Transfer a system from one side to another
  opposite(LOC,LOC)            % Gives a location on the opposite side
  nondeterm unsafe(STATE)      % Gives the unsafe states
  nondeterm member(STATE,PATH) % Checks if the state is already visited
```

```
  write_path(PATH)
  write_move(STATE,STATE)
GOAL
        go(state(east,east,east,east),state(west,west,west,west)),
        write("solved").
CLAUSES
  go(StartState,GoalState):-
        path(StartState,GoalState,[StartState],Path),
        write("A solution is:\n"),
        write_path(Path).
  path(StartState,GoalState,VisitedPath,Path):-
        move(StartState,NextState),                  % Find a move
        not( unsafe(NextState) ),                    % Check that it is not unsage
        not( member(NextState,VisitedPath) ),        % Check that we have not had this
situation before
        path( NextState,GoalState,[NextState|VisitedPath],Path),!.
  path(GoalState,GoalState,Path,Path).               % The final state is reached
  move(state(X,X,G,C),state(Y,Y,G,C)):-opposite(X,Y). % Move FARMER + WOLF
  move(state(X,W,X,C),state(Y,W,Y,C)):-opposite(X,Y). % Move FARMER + GOAT
  move(state(X,W,G,X),state(Y,W,G,Y)):-opposite(X,Y). % Move FARMER +
CABBAGE
  move(state(X,W,G,C),state(Y,W,G,C)):-opposite(X,Y). % Move ONLY FARMER
  opposite(east,west).
  opposite(west,east).

  unsafe( state(F,X,X,_) ):- opposite(F,X),!.  % The wolf eats the goat
  unsafe( state(F,_,X,X) ):- opposite(F,X),!.  % The goat eats the cabbage

  member(X,[X|_]):-!.
  member(X,[_|L]):-member(X,L).

  write_path( [H1,H2|T] ) :-
        write_move(H1,H2),
        write_path([H2|T]).
  write_path( [] ).

  write_move( state(X,W,G,C), state(Y,W,G,C) ) :-!,
    write("The farmer crosses the river from ",X," to ",Y),nl.
  write_move( state(X,X,G,C), state(Y,Y,G,C) ) :-!,
    write("The farmer takes the Wolf from ",X," of the river to ",Y),nl.
  write_move( state(X,W,X,C), state(Y,W,Y,C) ) :-!,
```

```
  write("The farmer takes the Goat from ",X," of the river to ",Y),nl.
write_move( state(X,W,G,X), state(Y,W,G,Y) ) :-!,
  write("The farmer takes the cabbage from ",X," of the river to ",Y),nl.
```

Пример работы программы:



Из решения ясно что:

1) Фермер берёт козу с запада на восток
2) Возвращается на запад
3) Берёт капусту с запада на восток
4) Забирает козу с востока на запад
5) Берёт волка с запада на восток
6) Возвращается на запад
7) Забирает козу с запада на восток

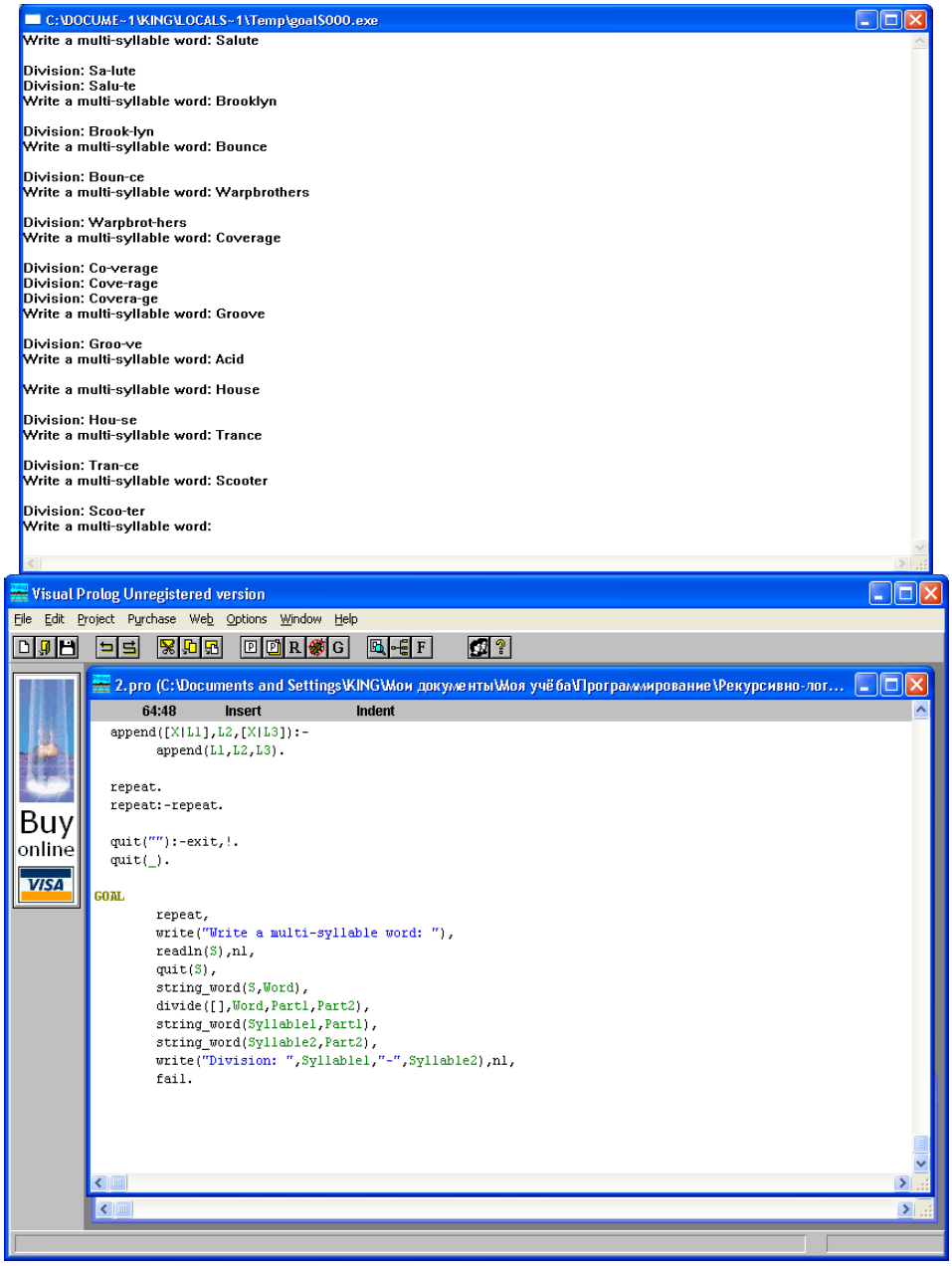
3. Задание: программы делает перенос слов. Вводится многослоговое слово, и выводится возможный вариант переноса.

Текст программы:

```
DOMAINS
        letter = char
        word_ = letter*
PREDICATES
        nondeterm divide(word_,word_,word_,word_)
        vocal(letter)
        consonant(letter)
        nondeterm string_word(string,word_)
        append(word_,word_,word_)
        nondeterm repeat
        quit(string)
CLAUSES
  divide(Start,[T1,T2,T3|Rest],D1,[T2,T3|Rest]):-
        vocal(T1),
        consonant(T2),
        vocal(T3),
        append(Start,[T1],D1).
  divide(Start,[T1,T2,T3,T4|Rest],D1,[T3,T4|Rest]):-
        vocal(T1),
        consonant(T2),
        consonant(T3),
        vocal(T4),
        append(Start,[T1,T2],D1).
  divide(Start,[T1|Rest],D1,D2):-
        append(Start,[T1],S),
        divide(S,Rest,D1,D2).
  vocal('a').
  vocal('e').
  vocal('i').
  vocal('o').
  vocal('u').
  vocal('y').
```

```
consonant(B):-
    not(vocal(B)),
    B <= 'z',
    'a' <= B.
string_word("",[]):-!.
string_word(Str,[H|T]):-
    bound(Str),
    frontchar(Str,H,S),
    string_word(S,T).
string_word(Str,[H|T]):-
    free(Str),
    bound(H),
    string_word(S,T),
    frontchar(Str,H,S).
append([],L,L):-!.
append([X|L1],L2,[X|L3]):-
    append(L1,L2,L3).
repeat.
repeat:-repeat.
quit(""):-exit,!.
quit(_).
GOAL
    repeat,
    write("Write a multi-syllable word: "),
    readln(S),nl,
    quit(S),
    string_word(S,Word),
    divide([],Word,Part1,Part2),
    string_word(Syllable1,Part1),
    string_word(Syllable2,Part2),
    write("Division: ",Syllable1,"-",Syllable2),nl,
    fail.
```

Пример работы программы:



4. Задание: Метро содержит две линии и одну станцию пересадки. Как проехать в метро? Ввести начальную и конечную станции, получить маршрут.

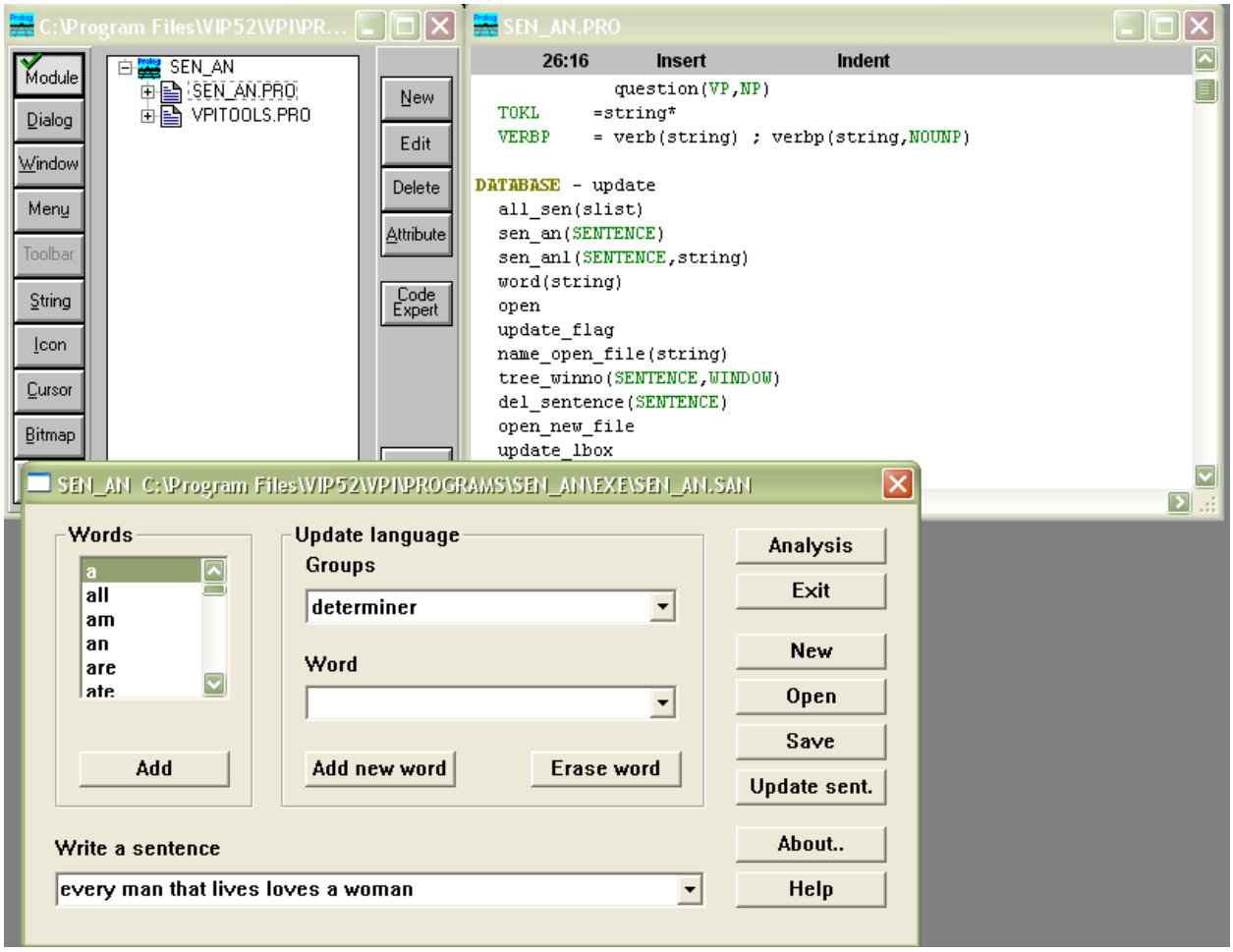
Текст программы:

DOMAINS
element = symbol
*list = element**
PREDICATES
append (list,list,list)
mline (integer,list)
member (element,list)

```
 prelist (list,element,list)
 postlist (list,element,list)
 deletel (list,list,list)
 betweenl (element,element,list,list)
 numb (element,list,integer)
 change_line (integer,integer,element)
 q1 (element,element)
 q2
CLAUSES
% Проверка принадлежности элемента к списку
 member (H,[H|_]).
 member (H,[_|T]):-member (H,T).
% К первому списку добавляется второй список, результат в третьем списке
 append ([],L,L).
 append ([H|L1],L2,[H|L3]):-append (L1,L2,L3).
% Список элементов L1 до элемента A в списке L
 prelist (L1,A,L):-append (L1,[A|_],L).
% Удаление списка L1 из списка L, результат - список L2
 deletel (L,L1,L2):-append (L1,L2,L).
 deletel (L,L1,L2):-append (L2,L1,L).
% Список L1 часть списка L после элемента A
 postlist (L1,A,L):-prelist (L2,A,L),deletel (L,L2,L3),deletel (L3,[A],L1).
% Номер элемента X в списке
 numb (X,[X|_],1)
 numb (X,[_|Y],N):-numb (X,Y,N1), N=N1+1.
% Список ABL - часть списка L между элементами A и B, включая A и B
 betweenl (A,B,L,ABL):-numb (A,L,N1),numb(B,L,N2),N1<=N2,
        prelist (L1,A,L),postlist (L2,B,L),deletel (L,L1,L3),
        deletel (L3,L2,ABL).
 betweenl (A,B,L,BL):-prelist (L1,B,L),postlist (L2,A,L),
 deletel (L,L1,L3),deletel (L3,L2,BL).
% Список станций линии 1,a,b,c,p,d,e,f - имена станций
 mline (1,[a,b,c,p,d,e,f]).
% Список станций линии 2, имена станций:g,h,c,p,k,l,m,n
 mline (2,[g,h,c,p,k,l,m,n]).
% Станцией пересадки с линии 1 на линию 2 является станция p
 change_line (1,2,p).
% Нахождение маршрута между станциями S1 и S2
 q1 (s1,S2):-mline (1,X),n1, write("Станции первой линии:",X),n1,
      member (S1,X), member (S2,X),betweenl (S1,S2,X,V),
      write ("Ваш маршрут 1-й линии:",V).
```

```
q1 (s1,S2):-mline (2,X),n1, write("Станции второй линии:",X),n1,
    member (S1,X), member (S2,X),betweenl (S1,S2,X,V),
    write ("Ваш маршрут 2-й линии:",V).
q1 (s1,S2):-mline (1,X), mline (2,Y),
    change_line (1,2,S3),n1, write ("Пересадка на станции",S3),n1,
    member (S1,X), member (S2,Y),
    betweenl (S1,S3,X,V1),
    write ("ехать по 1-й линии:",V1),n1,
    betweenl (S3,S2,Y,V2),
    write ("ехать по 2-й линии:", V2).
q1 (s1,S2):-mline (1,X), mline (2,Y),
    member (S1,Y), member (S2,X),
    change_line (1,2,S3),n1, write ("Пересадка на станции",S3),n1,
    betweenl (S1,S3,Y,V1),
    write ("ехать по 2-й линии:",V1),n1,
    betweenl (S3,S2,X,V2),
    write ("ехать по 1-й линии:", V2).
% Создание окна, ввод исходных данных, вызов основной процедуры
q2:-makewindow (1,62,241,"Метро",0,0,25,80),clearwindow,
write ("Введите, откуда ехать?"), readln(S1),
write ("Куда?"), readln (S2),
q1 (S1,S2).
GOAL
q2
```

Пример работы программы:



Рекомендуемая литература.

1. Адамченко А., Кугуков А. Логическое программирование и Visual Prolog в подлиннике. – С-Петербург, БХВ-Петербург, 2003.
2. Сайт специальности прикладная информатика в экономике УрГЭУ.htm.

ПРИЛОЖЕНИЯ

1.Текст программы работы N 1.

```
domains
  .......
predicates
  .......
clauses
  .......
```

```
parent(X,Y):-mother(X,Y). /*Y-родитель X, если Y-мать
X */ parent(X,Y):-father(X,Y). /*Y-родитель X, если Y-
отец X */

brother(X,Y):-          /* Y - брат X, если   */
 male(Y),          /* Y - мужчина, и     */
 parent(X,P),      /* P - родитель X, и  */
 parent(Y,P),      /* P - родитель Y, и  */
 X <> Y.           /* X и Y - разные люди*/

sister(X, Y) :-    /* Y - сестра X, если */
 female(Y),        /* Y - женщина, и     */
 parent(X,P),      /* P - родитель X, и  */
 parent(Y,P),      /* P - родитель Y, и  */
 X <> Y.           /* X и Y - разные люди*/

uncle(X,U) :-      /* U - дядя X, если ...*/
 mother(X,P)
 brother(P,U).

uncle(X,U) :-
 father(X,P),
 brother(P,U).

grandfather(X,G) :-    /* G - дедушка X, ... */
 father(P,G),
 mother(X,P).
grandfather(X,G) :-
 father(X,P),
 father(P,G).
```

2. Текст программы работы N 2.

```
domains
 name = symbol
predicates
 write_message
 repeat
 do_echo
 check(name)
goal
 write_message,
 do_echo.
clauses
 repeat.
 repeat :- repeat.

 write_message :-
   nl, write("Введите, пожалуйста, имена"),nl,
write ("Я повторю их"),nl,
write ("Чтобы остановить меня, введите stop"), nl,nl.


 do_echo :-
   repeat,
   readln(Name),
   write (Name),nl,
   check(Name),!.

 check(stop) :-
   nl, write(" - OK, bye").
 check (_) :- fail.
```

3. Текст программы работы N 3.

```
predicates
 start
 run(integer)
 do_sums
 set_up_windows
 clear_windows
                        /* Goal: start */
clauses
 start :- set_up_windows, do_sums.

 set_up_windows :-
   makewindow(1, 7, 7, "", 0, 0, 25, 80),
   makewindow(1, 7, 7, "Left operand", 2, 5, 5, 25),
   makewindow(2, 7, 7, "", 2, 35, 5, 10),
   nl, write(" PLUS "),
   makewindow(2, 7, 7, "Right operand", 2, 50, 5, 25),
   makewindow(3, 7, 7, "Gives", 10, 27, 5, 25).
   makewindow(4, 7, 7, "", 17, 22, 5, 35).

 do_sums :- run(_), clear_windows, do_sums

 run(Z) :-
   shiftwindow(1),
   cursor(2, 1), readint(X),
   shiftwindow(2),
   cursor (2, 10), readint(Y),
   shiftwindow(3), Z=X+Y, cursor(2, 10), write(Z),
   shiftwindow(4),
   write("Please press the space bar"),
   readchar(_).

 clear_windows :-
   shiftwindow(1), clearwindow,
   shiftwindow(2), clearwindow,
   shiftwindow(3), clearwindow,
   shlftwindow(4), clearwindow.
```

4. Текст программы работы N 4.

```
/*trace*/
domains
 list = integer*
predicates
 run
 rdlist(list)
 tail_list(list)
 head_list(list)
goal
 run.
clauses
 run :-  write("Введите список - "),nl,
        rdlist(L),
        write("Обратный порядок - "),nl
        tail_list(L),nl,
        write("Прямой порядок - "),nl,
        head_list(L).

 rdlist([S|L]) :- readint(S),
                  rdlist(L).
 rdlist([]).

 tail_list([S|T]) :-  tail_list(T),
                      write(S,", ").
 tail_list([]).

 head_list([H|T])  :- write(H,","),
                      head_,list(T),
head_list([]).
```