КЫРГЫЗСКИЙ ГОУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И. Раззакова

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра **«ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СИСТЕМ»**

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПРОЛОГ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Для студентов специализации
552801.04 «Программное обеспечение вычислительной техники и автоматизированных систем»

Бишкек - 2005

РАССМОТРЕНО
На заседании кафедры
«Программное обеспечение
компьютерных систем»
Прот.№12 от 13.07.2005г.

ОДОБРЕНО
Методическим советом ФИТ

Прот. №.

УДК 519.682

Составитель – **МАКИЕВА З.Д.**

Программирование на языке Пролог: методические указания к выполнению лабораторных работ /Кырг. гос. техн. ун-т, Бишкек, 2005, ? с.

Представлены теоретические сведения логического языка Пролог с примерами разработанных программ, также приведены задания на составление программ для самостоятельной работы. Логический язык Пролог предлагается для изучения по дисциплине “Функциональное и логическое программирование” (раздел - логическое программирование).

Предназначено для студентов специальности «Программное обеспечение вычислительной техники и автоматизированных систем» всех форм обучения.

Библиогр. ? названий.

Рецензент:

Содержание

Введение

По мере преодоления технических проблем развития компьютеров накапливались трудности в области создания программного обеспечения. Начались поиски языков программирования. И хотя степень абстракции от машинных языков (Ассемблеров) к языкам более высокого уровня возрастала, все эти языки несли на себе печать фон-неймановской архитектуры и необходимость детально описывать процедуры получения решения. Их основной недостаток состоял в необходимости заранее знать, какие могут быть заданы вопросы, и запрограммировать процедуры, которые будут давать ответы на них.

Ситуация изменилась с появлением логического программирования, основанного на идеях и методах математической логики. Логическая программа строится как набор утверждений (фактов и правил) об объектах, функциях и отношениях предметной области. (Описание задач является статистическим и никого вычислительного процесса не задаёт.)

С самого раннего детства мы начинаем рассуждать, причем, основное средство рассуждений — высказывания и, основанные на них логические выводы: «Если будешь слушаться, то мама купит новую игрушку», «Если будет повышенная температура, то не пойдешь в школу». А если температура не повышенная, можно ли не идти в школу? Можно, если ты не готов к контрольной работе, которая как раз сегодня. И стоит пойти в школу, если ты к ней готов и температура повышенная, но не очень. Логика бывает разной, бывает мужской, женской. И, если верить поговорке то, что «для русского — здорово, то для немца — смерть». То есть все зависит от системы аксиом и правил вывода. Если говорить математическим языком, то у них (женщин и мужчин, немцев и русских) различные дедуктивные системы.

Дедуктивной системой называется способ задания множества путем указания исходных элементов (аксиом исчисления) и правил вывода, каждое из которых описывает, как строить новые элементы из исходных.

Используются различные термины для обозначения понятия дедуктивной системы: исчисление, формальная (аксиоматическая) теория(система).

В основе логического программирования лежит описание задачи совокупностью утверждений на некотором формальном логическом языке и получение решения с помощью вывода в некоторой формальной (дедуктивной) системе.

В общем смысле, **Логическое программирование** представляет собой множество таких методов решения проблем, в которых используются приемы логического вывода для управления знаниями, представленными в декларативной форме.

Самыми известными системами такого рода являются реализации языка Prolog (Programming in Logic). Пролог явился итогом многолетней исследовательской работы. Первая официальная версия Пролога была разработана в 1972 году Аланом Кольмероэ в Марсельском университете во

Франции. Пролог известен как декларативный язык, так как программа на этом языке программирования не является последовательностью действий, а представляет собой набор фактов с правилами, обеспечивающими получение заключений на основе этих фактов.

Пролог был принят в качестве базового языка в японской программе создания ЭВМ 5 поколения, ориентированной на исследование методов логического программирования и искусственного интеллекта, а также разработку нового поколения компьютеров.

Пролог имеет ряд преимуществ по сравнению с процедурными языками, например:

– для определенных задач программа на Прологе требует одну десятую часть строк кода по сравнению с программой на процедурном языке.

– благодаря декларативному, а не процедурному подходу, такие ошибки как зацикливания устраняются с самого начала.

Лабораторная работа №1.

Пролог базируется на предложениях Хорна, являющихся подмножеством формальной системы, называемой логикой предикатов.

Логика предикатов была разработана для наиболее простого преобразования принципов логического мышления в записываемую форму.

В логике предикатов исключаются все несущественные слова. Затем на первое место ставятся отношения, а после него сгруппированные объекты.

Предложения на естественном языке	Синтаксис логики предикатов
Цветок прекрасный.	beautiful (flower)
Маша любит розы.	likes (masha, rose)
Роза красная.	red (rose)
Маша любит розы, если они красные	likes (masha, rose) if red (rose)

Можно показывать отношение одного предмета к другому, например, 3>2. Предикат же показывает, верно ли отношение. Результатом является ложь или истина(0 или 1). Результатом предиката Больше(3,2) будет 1, а Больше(2,3) будет 0. Слово «Больше» можно заменить каким-либо символом, например: R(3,2).

Предикат — это функция, имеющая результат 0 или 1.

Программы на языке Prolog обычно состоят из 4-х основных программных разделов:

- раздел domains (доменов)
- раздел predicates (предикатов)
- раздел clauses (предложений)
- раздел goal (целей)

Раздел clauses

Раздел clauses является сердцевиной программы. В этом разделе помещаются все факты и правила, составляющие программу.

Факт представляет собой либо свойство объекта, либо отношения между объектами. Факт самодостаточен. Факт не требует доказательства и

может быть использован как основа для логического вывода. Факт заканчивается символом “.”

Например:

Ann likes cats.⇒ likes (ann , cats).

Cat is white. ⇒ white (cat).

Rose is red. ⇒ red (rose).

Факты – это отношения или свойства, о которых известно, что они принимают значение «истина».

Правила позволяют вывести один факт из другого. Правило принимает значение «истина», если заданный набор условий является истинным.

Например: Анна любит все, что любит Катя. Катя любит всё красное.

likes (ann, Something): - likes (kate, Something).

likes (kate, Something): - red (Something).

Все правила имеют две части. Символ «:-» имеет смысл «если» и служит для разделения двух частей правила: заголовка и тела.

Заголовок – это факт, который будет истинным, если будут истинными условия находящиеся в теле.

Тело – это условия, которые должны быть истинными, чтобы заголовок был истинным.

Заметим, что наименования фиксированных постоянных объектов, то есть известных величин пишутся с маленькой буквы ann, kate. Если хочется написать с большой буквы, заключаем в кавычки.

Наименования переменных объектов начинаются с большой буквы или символа подчеркивания. Если для выполнения заголовка правила требуется выполнение нескольких условий применяются логические связки конъюнкция ∩, которая заменяется запятой (или словом and) и дизъюнкция ∪, которая заменяется “;” или or.

Например:

eats (Who, What): - food (What), likes (Who, What). (Мы едим что-то, если что-то – еда *и* она нам нравится).

flowers (What): - rose (What); violet (What).(Что-то является цветком, если это что-то – роза *или* фиалка).

Раздел предикатов.

Если в разделе clauses существует созданный вами предикат, то этот предикат должен быть объявлен в разделе предикатов, иначе Prolog не поймет о чем идет речь. Если этот стандартный, встроенный предикат Пролога, то объявлять его не нужно.

Объявление предиката начинается с имени предиката, затем в скобках следует ноль и более доменов (типов) аргументов предиката.

predicatename (*тип аргумента* argument _ type_1 *имя аргумента* name1, argument _ type_2 name2, …..)

Типы аргументов разделяются запятой. Можно указывать имена аргументов – это улучшает читаемость программы, но не сказывается на

скорости исполнения, так как компилятор их игнорирует. (В конце “.” не ставится в отличие от clauses.)

Доменами (типами) могут быть либо стандартные домены, либо собственные объявленные в разделе домены.

Имя предиката начинается с маленькой буквы (начинать имя с большой буквы запрещается во многих версиях Пролога) и может иметь длину до 250 символов.

Разрешается использовать заглавные, строчные буквы, цифры и символы подчеркивания. Нельзя использовать пробел, *, / , — и некоторые др. символы.

Арность (размерность предиката) – это количество аргументов, которые он принимает. Можно иметь 2 предиката с одним и тем же именем, но отличающихся арностью.

В разделах predicates и clauses предикаты с одним и тем же именем должны группироваться вместе. Предикаты с одним именем, но различной арности является разными, но тоже должны группироваться.

Например:

```
farther (symbol)     %  этот человек - отец.
farther (symbol, symbol)    % этот человек является отцом другого.
```

Переменные

Как уже упоминалось выше, имена переменных начинаются с заглавной буквы или символа подчеркивания, после которой может стоять любое количество букв (заглавных и строчных), цифр или символов подчеркивания.

Вместо неопределённого X лучше использовать осмысленный выбор имен. Важное отличие Пролога от алгоритмических языков – он не имеет оператора присваивания.

Переменные в Прологе инициализируются при сопоставлении с константами в фактах или правилах. До инициализации переменная свободна, после присвоения ей значения она становится связанной. Переменная остается связанной только то время, которое необходимо для получения решения по запросу, затем Пролог освобождает ее и ищет другое решение. Нельзя сохранить информацию, присвоив значение переменной. Переменные используются как часть процесса поиска решения, а не как хранилище информации.

Пример:

```
predicates
        likes (symbol, symbol)
clauses
        likes (ellen, reading).
        likes (mark, football).
        likes (ellen, dancing).
        likes (eric, reading).
        likes (eric, swimming).
goal
        likes (Person, reading),
```

likes (Person, swimming).

Раздел goal (цели)

Задав какие-то факты в Прологе, мы можем задавать запросы, касающиеся отношений между ними, т.е. задав запрос, фактически мы ставим ему цель для выполнения.

В разделе goal задается внутренняя цель программы. Это позволяет, программе выполнятся независимо от среды визуальной разработки (VDE).

Цели могут быть простыми и сложными.

Цель, состоящая из двух и более целей, называется сложной целью, а каждая часть сложной цели называется подцелью.

Составные цели можно использовать для поиска решения, в котором каждая подцель истинна (конъюнкция), разделяя подцели запятой или истинна одна из подцелей, разделяя подцели точкой с запятой.

Пример:

```
domains
  name, color = symbol
        price = integer
predicates
  car (name, color, price)
  truck (name, color, price)
clauses
   car (ford, red, 12000).
   car (toyota, white, 30000).
   car (nissan, black, 25000).
   truck (ford, blue, 8000).
   truck (nissan, black, 15000).
   truck (toyota, orange, 25000).
 goal
   car (Name, Color, 20000).
```

или составная цель

```
   car (Name, Color, Price), Price < 20000.
```

или использовать конъюнкцию и дизъюнкцию

```
   car (Name, Color, Price), Price < 20000;
   truck (Name, Color, Price), Price < 10000.
```

выполнится I подцель, если нет, то будет выполняться II.

Ответ

```
   Name=ford, Color=red, Price=12000
   Name=ford, Color=blue, Price=8000
2 Solutions
```

Иногда удобно составные цели записывать в разделе предикатов.

В разделе предикатов q1
q2

в разделе clauses

```
 q1: - car (Name, Color, Price), Price < 20 000.
 q2: - truck (Name, Color, Price), Price < 10 000.
```

в разделе goal q1, nl, q2.

Далее в отдельной главе мы изучим стандартные предикаты ввода и вывода, а пока предлагаю использовать несколько стандартных предикатов:

nl – новая строка,

write (“q1=”, “Name =”, Name) – вывод текста (в кавычках любой текст, без кавычек значение указанной переменной).

readchar (_) – ввод любого символа.

Комментарии

Хорошим стилем программирования является включение комментариев.

Многострочные комментарии заключаем между символами / *…*/, а перед однострочным комментарием ставим символ %.

В Visual Prolog можно не пользоваться комментарием после каждого субдомена в объявлениях доменов и предикатов,

likes (symbol Personname, integer Personage).

Ознакомившись с основами логического языка Пролог, мы можем написать свои первые программы.

Задания для самостоятельной работы:

1. Ввести 10 фактов о том, кто что любит.

Например: Саша любит читать, Маша любит плавать и т.п.

Задать запросы, чтобы получить ответы «да», «нет», списки тех, кто любит читать, кто любит читать и плавать.

2. Ввести перечень предметов с указанием цвета предмета. Один из элементов списка - цветок.

Алине нравится все зеленое. Сабине – все красное. Саша любит зеленый и красный цвет.

Определить:

Что нравится Алине?
Что нравится Сабине?
Что нравится Саше?
Кому нравятся цветы?

3.Известно, что:

Маше нравятся цветы.
Аня любит сына.
Сын Ани любит Машу.
Оля любит всех, кого любит Аня.
Нам нравится все, что нравится человеку, которого мы любим.

Определить:

a) Нравятся ли сыну Ани цветы?
b) Любит ли Аня Машу?
c) Что любит Оля?
d) Кому нравятся цветы?
e) Нравятся ли Ане цветы?
f) Кто любит Машу и цветы?

4. Имеются данные о работниках фирмы: фамилия, пол, возраст, семейное положение. Программа должна вывести список холостых мужчин старше 35 лет.

5. Имеются данные о детях сотрудников: фамилия, имя, возраст. Составить список детей, получающих новогодние подарки, если подарки выдают детям в возрасте до 12 лет.

Лабораторная работа №2

Раздел domains (доменов)

В Visual Prolog есть встроенные стандартные домены short, ushort, long, ulong, integer, unsigned, byte, word, dword – целые типы данных, char – 1 символ, real – вещественный тип, string, symbol - строки.

Их можно использовать при декларации типов аргументов предикатов без описания в разделе domains. Также можно описать собственные домены. Раздел domains служит двум полезным целям.

Во-первых, можно задать доменам осмысленные имена, даже если эти домены аналогичны уже имеющимся стандартным.

Пример:

Рубашка красного цвета стоимостью 200 сомов. Можно написать предикат thing (symbol, symbol, integer) и соответственно факт thing (shirt, red, 200).

Более осмысленная запись:

```
domains
        name, color  =  symbol
        price = integer
 predicates
        thing (name, color, price).
```

Объявление новых доменов с использованием стандартных имеет вид
<имя> = <имя стандарт. домена>

```
domains
  i = integer
  r = real
  name, color = symbol
```

Объявления типа r = real ускоряют набор программы.

Введение же “авторских” наименований доменов типа name, color = symbol позволяет внести больше семантики и обеспечивает контроль типов значений переменных, поскольку смешивать в ходе выполнения программы переменные формально различных доменов нельзя, т.е. в примере

thing (name, color, price)

Нельзя записать цель thing (color, _,_).

надо thing (_, color, _).

Можно использовать перечислимый тип
Name = ann; olga; flowers
Например:

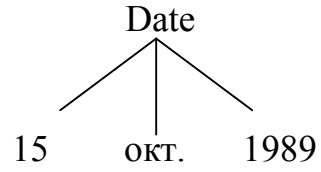
```
domains
Name = ann; olga; flowers
        predicates
                like(Name, Name)
        clauses
                like(ann, flowers).
                like(olga, flowers).
                like(eric, flowers).  – ошибка!
```

Во-вторых, объявление специальных доменов используется для описания структур данных, отсутствующих в стандартных доменах.

Простой объект данных – это переменная или константа. Переменные Пролога всегда локальные. Это означает, что, если два различных клоза содержат переменные с одним и тем же именем *X*, их следует понимать как различные переменные. Это не исключает, что они могут быть связаны одна с другой в процессе унификации.

Константы включают символы, числа и атомы. Атомы представляют символьные строки типа symbol или типа string. Константное значение есть ее имя.

Составные объекты данных позволяют рассмотреть некоторые части информации как единое целое таким образом, чтобы затем можно было легко разделить их.
Возьмем дату 15 октября 1989, она состоит из 3-х частей.
Представим как древовидную структуру.



Можно объявить:

```
domains
date_ bd = date ( string , unsigned , unsigned)
```

а затем записать

```
D = date (october, 15, 1989).
```

Функтор (не функция) с 3 аргументами - это новый объект данных, который можно обрабатывать наряду с символами и числами.
Не путать функтор с функцией. ***Функтор не предполагает ни каких вычислений или преобразований. Это только осмысленное имя некоторого составного объекта, поддерживающего совместную запись аргументов***.

Составные объекты данных позволяют представлять и обрабатывать сложные структуры информационных объектов. Составной объект состоит из имени, обычно называемого *функтором* и одного или более аргументов. Имеется возможность объявлять домен с несколькими альтернативными функторами.

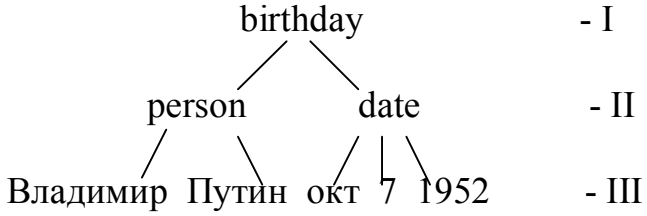
```
domain =    alternative1(D, D, ...);
           alternative2(D, D, ...);
       …
```

Здесь ***alternative1*** и ***alternative2*** есть различные специфицирующие объект функторы. Нотация (*D, D,* ...) представляет список имен доменов, которые или декларируются далее (в глубину) или определяются через стандартные домены (такие как ***symbol***, ***integer***, ***real*** и т.д.) .

Аргументы составного объекта могут в свою очередь сами быть составными объектами.
Составной объект может рассматриваться и трактоваться как единый объект; кроме того, функтор позволяет фиксировать различия между объектами. Visual Prolog позволяет конструировать объекты с многоуровневым описанием. Допускаются смешанные декларации доменов. Вы можете использовать предикаты:

- принимающие аргументы более чем одного возможного типа, используя *функтор декларации*;
- принимающие переменное число аргументов специфицированного типа;
- принимающие переменное число аргументов, некоторые из которых могут быть более чем одного типа

Например, рассмотрим чей-то день рождения, как информацию со следующей структурой:



birthday(person(“Владимир”, “Путин”), date(“октябрь”, 7, 1952))

Здесь функторами являются birthday, person, date.

Составной объект может быть унифицирован с простой переменной или с составным объектом, т.е. можно передавать целый набор значений, как единый объект: X = date (окт., 7, 1952). Также date (окт., 7, 1952) сопоставляется с (M, D, Y) и присваивает переменным M = “окт.” , D = 7, Y = 1952.

Объявления составных доменов

```
domains
birthday = birthday (name, date)
name = name (first, last)
date = date (m, d, y)
first, last, m = string
d, y = integer
```

Например:

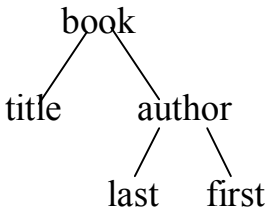
```
domains
        thing = book (title, author); car (marka); dog(name)
         % если перечисляем несколько доменов, то пишем через ";"
        title, author, marka, name = symbol
predicates
nondeterm owns(name,thing)
clauses
        owns( bill, book("Titan", "Teodor Drizer")).
        owns (bill, dog (dick)).
        owns (bill, car (bmw)).
goal
        owns (bill, Thing).
```

Ответ

```
Thing=book("Titan","Teodor Drizer")
Thing=dog("dick")
Thing=car("bmw")
3 Solutions
```

Нарисуем при многоуровневых объектах дерево.

Например:



Декларация домена объявляет только 1 уровень, а не целое дерево.

```
domains
book = book (title, author (first, last)) – не правильно.
```

должно быть:

```
domains
        book = book (title, author)  %   I уровень
        author = author (first, last)  %  II уровень
        title, first, last = symbol     %  III уровень
```

Автоматическое преобразование типов.

Необязательно, чтобы при сопоставлении двух переменных они принадлежали одному и тому же домену. Переменные могут быть связаны с константами из различных доменов.

Visual Prolog автоматически выполняет преобразование типов из одного домена в другой, но только в следующих случаях:

- преобразование между строками (string) и идентификаторами (symbol)
- между целыми, действительными числами и символами (char). При преобразовании символа в числовое значение этим значением является величина символа в коде ASCII.

Такое преобразование типов означает, что вы можете

- вызвать предикат с аргументом типа string, задавая ему аргументы типа symbol и наоборот;
- передавать предикату типа char параметры типа integer;
- передавать предикату типа real параметры типа integer;
- использовать в предложениях и сравнениях символы без необходимости получения их кодов в ASCII.

Анонимные переменные.

Анонимные переменные используются, если значение переменной несущественно, т.е. если вам нужна только определенная часть запроса, то анонимная переменная нужна для игнорирования ненужных значений. Анонимная переменная обозначаются символом подчеркивания "_". Например, если речь идет только об авторе, а наименование книг, издательств, год издания не играют роли, то создаем предикат book (Author, _ , _ , _).

Пример:
```
predicates
    father (symbol, symbol)
clauses
    father (bill, eric).
    father (eric, mark).
    father (serge, ellen).
goal
    father (Father, _ ).
```
Решение будет
```
Father = bill
Father = eric
Father = serge
3 Solutions
```

Задания для самостоятельной работы

1. Создать небольшой телефонный справочник, имеются данные: фамилия, инициалы, адрес, номер телефона. Задать запросы по фамилии, по номеру, по адресу.

2. Составить список отцов и матерей. Сделать запросы:
   а) всех родителей; б) родителей конкретного ребенка.
3. Тони, Майкл и Джон – члены альпинклуба.

Каждый член альпинклуба или горнолыжник, или скалолаз, или и то и другое.

Никто из скалолазов не любит дождь.
Все горнолыжники любят снег.
Майкл любит все, что не любит Тони, и не любит все, что любит Тони.
Тони любит снег и дождь.
Джон любит снег.
Определить:

a) Есть ли член альпинклуба, который является скалолазом и не является горнолыжником? Если есть, то кто он?
b) Кто член альпинклуба, который является скалолазом и горнолыжником?
c) Тони скалолаз или горнолыжник?

Задачу решить, используя перечислимый тип данных.

Лабораторная работа № 3

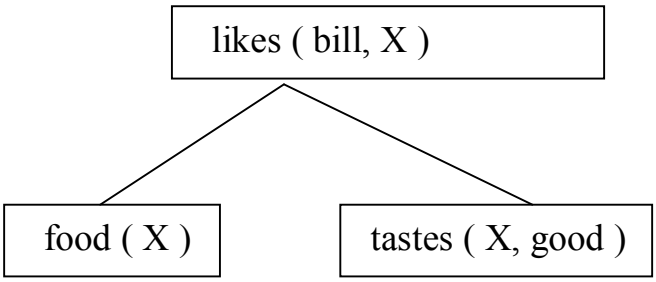
Поиск с возвратом (backtracking)

Часто при решении реальной задачи мы придерживаемся определенного пути для её логического завершения. Если полученный результат не даёт исходного ответа, мы должны выбрать другой путь. Например, поиск выхода из лабиринта. Один из верных способов найти конец лабиринта – это поворачивать налево на каждой развилке лабиринта до тех пор, пока не попадётё в тупик. Тогда следует вернуться к последней развилке и попробовать свернуть вправо, после чего опять поворачивать налево на каждой развилке. Путём методичного перебора всех возможных путей находящихся в выходе.

Visual Prolog при поиске решения задачи использует именно такой метод проб и возвращений назад. Этот метод называется поиск с возвратом. Если, начиная поиск решения задачи (или целевого утверждения), Visual Prolog должен выбрать между альтернативными путями, то он ставит маркер у места ветвления (называемого точкой отката) и выбирает первую подцель, которую и станет проверять. Если данная подцель не выполнится (что эквивалентно достижению тупика в лабиринте) Visual Prolog вернётся к точке отката и попробует проверить другую подцель.

Пример:

```
Predicates
   likes ( symbol, symbol )
   tastes ( symbol, symbol )
   food ( symbol )
Clauses
   likes ( bill, X ): - food ( X ), tastes ( X, good ).
   tastes ( pizza, good ).
   tastes ( pepper, bad ).
   food ( pepper ).
   food ( pizza ).
 goal
   likes ( bill, What ).
```



Процесс поиска решений называется сопоставлением, а совпадением цели с фактом или головой правила при сопоставлении называется унификацией.

Когда Пролог пытается произвести согласование целевого утверждения, он начинает поиск с вершины программы. Он обнаруживает соответствие с первым предложением и переменная What унифицируется с переменной X. Сопоставление с заголовком правила заставляет Visual Prolog попытаться удовлетворить это правило, ⇒, обращается к первой подцели food (X).

Если выполняется новое обращение, поиск соответствия для этого обращения вновь начинается с вершины программы. Пытаясь согласовать первую подцель, Visual Prolog, начиная с вершины программы, производит сопоставление с каждым фактом или заголовком правила. Соответствие у первого факта food (pepper), ⇒, X связана с pepper. Поскольку существует более чем один возможный ответ на обращение food(X) Visual Prolog ставит точку возврата возле факта food (pepper).

Когда установление соответствия обращения завершается успешно, говорят, что обращение возвращается, и может быть испытана очередная подцель. Поскольку X связана с pepper, следующее обращение имеет вид tastes (pepper, good).

Опять поиск с вершины программы обращения завершается неудачно.

Теперь Visual Prolog запускает механизм возврата, т.е. идёт к точке отката. При откате в поиске с возвратом все переменные освобождаются. X – свободная переменная.

Обращение food (X) обнаруживает соответствие с фактом food (pizza), т.е. X= pizza.

Пролог переходит к следующей подцели в правиле.

Проверяем tastes(pizza, good) опять начиная с вершины программы.

На этот раз решение найдено.

Основные правила поиска с возвратом.

1. Подцели должны быть согласованы по порядку, сверху вниз.
2. Предикатные предложения проверяются в порядке, в каком появляются в программе сверху вниз.
3. Когда подцель соответствует заголовку правила, далее должно быть согласовано тело этого правила, которое теперь образует новое множество подцелей для согласования.
4. Целевое утверждение считается согласованным, если соответствующий факт найден для каждой ветки целевого дерева.

Управление поиском решений.

Встроенный механизм поиска с возвратом в Прологе может привести к поиску ненужных решений, в результате теряется эффективность, например, когда желательно найти только одно решение. В других случаях можно оказаться необходимым продолжать поиск решений, даже если целевое решение уже согласовано. Visual Prolog даёт 2 инструментальных средства, которые дают возможность управлять механизмом поиска с возвратом: предикат fail, который используется для инициализации поиска с возвратом, и предикат cut (или отсечение обозначается «!») – для запрета возможности возврата.

Декларативная семантика Пролог – программы может быть правильной, но процедурная ошибочной. Тогда программа не может добраться до ответа, т.к. выбирает неверный путь. Рекомендуется сначала сосредоточится на декларативных аспектах задачи, затем пропустить программу на машине, и, если она окажется неправильной, попытаться изменить порядок следования предложений и целей.

Использование предиката fail

Visual Prolog начинает поиск с возвратом, когда вызов завершается неудачно. В определенных ситуациях бывает необходимо инициализировать выполнение поиска с возвратом, чтобы найти другие решения. Visual Prolog поддерживает специальный предикат fail, вызывающий неуспешное завершение, а $\Rightarrow$ инициирует возврат.

Пример:

```
  domains
  name = symbol
  predicates
    father (name, name)
    everybody
    clauses
       father (peter, ann ).
       father ( john, alice).
       father (alex, eric).
       everybody: - father (X, Y), write (X, “is”, Y, “ ‘s father\n”), fail.
   goal
         father (X, Y).
   goal
         everybody.
```

Test Goal найдет все решения для father (X, Y).

Но если скомпилировать программу и запустить с помощью Project $\rightarrow$ Run (F9), то Пролог найдет I решение. Отличие 2-х целей в предикате fail.

Предикат fail не может быть согласован (он всегда неуспешен), поэтому Visual Prolog вынужден повторять поиск с возвратом.

При поиске с возвратом он возвращается к последнему обращению, который может произвести множественные решения. Такое обращение называют недетерминированным. Детерминированное обращение может

произвести только одно решение. Предикат write не может быть вновь согласован, (он не может предложить новых решений), поэтому Visual Prolog должен выполнить откат дальше на этот раз к первой подцели в правиле.

Обратите внимание, что помещать подцель после предиката fail бесполезно, он все время завершается неудачно.

Предикат cut (отсечение !)

Visual Prolog предусматривает возможность отсечения, которая используется для прерывания поиска с возвратом.

Отсечение помещается в программу таким же образом, как и подцель в теле правила. Когда процесс проходит через отсечение, удовлетворяется обращение к cut и выполняется обращение к очередной подцели (если таковая имеется).

Однажды пройдя через отсечение, уже невозможно произвести откат к подцелям, расположенным в обрабатываемом предложении перед отсечением, и также невозможно возвратиться к другим предложениям, определяющим обрабатывающий предикат (предикат, содержащий отсечение).

Существуют 2 основных случая применения отсечения:

- Если вы заранее знаете, что определенные подцели правила никогда не приведут, к осмысленным решениям, бектрекинг в этом случае будет, лишней тратой времени - применяют отсечение. Программа станет быстрее и экономичнее. Такой прием называют зеленым отсечением.
- Если отсечения требует сама логика программы для исключения из рассмотрения альтернативных подцелей. Это – красное отсечение.

Пример:

Перевести числовые значения отметок в их словесные значения:
более 86 баллов – otl, более 73 - hor, более 60 – ud, менее 60 – neud.

Вариант программы без использования отсечения:

```
domains
        i=integer
        s=symbol
predicates
nondeterm   perevod ( i,s )
clauses
        perevod  ( Z, "otl" ):- Z>= 87.
        perevod  ( Z, "hor" ):- Z>= 74, Z <87.
        perevod  ( Z,  "ud"):- Z >= 61, Z < 74.
        perevod  ( Z , "neud" ):- Z<61.
goal
        perevod ( 60,X ).
```

Лучший вариант программы с использованием отсечения:

```
domains
        i=integer
        s=symbol
```

```
predicates
        nondeterm   perevod ( i,s )
clauses
        perevod ( Z, _ ) :- Z < 0,!, fail .
        perevod ( Z, _ ) :- Z > 100,!,fail.
        perevod ( Z, "otl" ):- Z>= 87,!.
        perevod ( Z, "hor" ):- Z>= 74,!.
        perevod ( Z,  "ud"):- Z >= 61,!.
        perevod ( _ , "neud").
goal
        perevod ( 80,X ).
```

Здесь удовлетворение одного условия исключает все последующие варианты и предусмотрена защита от некорректных данных (два первых варианта) при запросе perevod (200, X) предикат fail остановит механизм возврата и на запрос будет дан отрицательный ответ предикат “!” отсекает все последующие откаты.

— Отсечение отбрасывает все расположенные после него предложения.

— Конъюнкция целей стоящих перед отсечением, приводит не более чем к одному решению.

— Отсечение не влияет на цели расположенные правее него.

— Если конъюнкция не выполняется, то поиск переходит к последнему выбору, сделанному перед выбором предложения с отсечением.

Предикат not (отрицание)

Использование оператора отрицания позволяет упрощать конструкции.

Например,

```
predicates
nondeterm likes (symbol ,symbol).
hates (symbol , symbol ).
clauses
likes (bill, X):- likes (sue, X), not (hates (bill, X)).
likes (sue ,ann).
likes (sue ,alice).
hates (bill , ann ).
goal
likes ( bill , X ).
```

Ответ:

```
X=alice
1 solution
```

Предикат not должен использоваться со связанными переменными. Некорректно: likes (bill , X): - not (hates (bill , X)), likes (sue, X). При вызове подцели со свободными переменными Prolog возвратит сообщение об ошибке: “Free variables not allowed in *not* or *retractall* .”

Замечание: Если не может быть доказана истинность подцели отрицания, то предикат будет успешным.

Задания для самостоятельной работы

1. Определить знак введенного числа.
2. Предположим, что средний налогоплательщик США, женатый горожанин с двумя детьми и зарабатывающий не менее $500 и не более $2000 в месяц. Определите специальный предикат налогоплательщика, заданный целью `special_taxpayer(X),` который будет успешно достигать цели только тогда, когда у X будет нарушено одно из условий среднего налогоплательщика.

   Используйте предикат обрезки, если Вы не предполагаете использовать бэктрекинг.
3. Дан список сотрудников. Также дан список легковых и грузовых автомобилей с датой выпуска и их владельцев. При одном запуске программы:

   а) Вывести список сотрудников, не имеющих машины.

   б) Вывести список сотрудников, имеющих машины, позднее 1999 года выпуска.

   в) Вывести фамилию одного сотрудника, имеющего легковую машину, позднее 1999 года выпуска.

   Ответ должен быть оформлен с заголовками списков, примерный вид представлен ниже.

   The list of the employees who aren't having the machines:
   Bill
   Smitt
   The list of the employees having the machines, before 1999 of release:
   Fred Mazda(car) 1998
   Dany BMW(truck) 1997
   The employee, having the automobile machine, after 1999 of release:
   John audi(car) 2000
   No

Лабораторная работа №

Списки

Предложим, что нужен список о преподавателях и предметах, которые они ведут. Тогда можно написать программу

```
predicates
        teacher (symbol, symbol, symbol).
clauses
        teacher (mary, mayson, english1).
        teacher (mary, mayson, english2).
        teacher (nike, brown, history).
        teacher (nike, brown, sociology).
```

Повторяем имя преподавателя для каждого предмета. Здесь было бы удобно создать такой аргумент для предиката, который содержит одно или несколько значений. Такой тип называется списком.

Список – это объект, содержащий конечное число других объектов. Отличие от массивов нет необходимости заранее объявлять размер.

Наша программа имеет вид:

```
domains
pregmets = symbol *      % объявляем домен - список
predicates
        teacher (symbol    FirstName, symbol   LastName, predmets    Pregmets)

clauses
        teacher (mary, mayson,[english1, english2]).
        teacher (nike, brown,[first, sociology]).
```

в symbol* * обозначает, что predmets – список идентификаторов.
если объявляем список целых integer – list = integer *

Каждая составляющая списка называется элементом, они разделяются запятыми и заключаются в [].

Список, содержащий числа, записывается [1,2,3]

Элементы списка могут быть любые, включая другие списки, однако, все элементы должны принадлежать одному домену.

```
domains
     element = element *
     element = integer ; real ; symbol %  неверно
```

Чтобы объявить список из целых, действительных, символьных значений надо определить один тип, включающий все эти три типа с функторами которые покажут, к какому типу относится тот или иной элемент

```
domains
    elements = element *
    element =  i (integer); r (real); s (symbol) % i, r, s – функторы составных объектов.
```

Повтор и рекурсия.

В прологе нет явных операторов цикла как for, do/while или while, не существует прямого способа выражения повтора.

Пролог обеспечивает 2 вида повторения – откат, с помощью которого выполняется поиск многих решений в одном запросе, и рекурсию, в которой процедура вызывает сама себя.

Поиск с возвратом является хорошим способом определить все возможные решения целевого утверждения. Но даже если задача не имеет множества решений, можно использовать поиск с возвратом для выполнения итераций.

Повтор, определяемый пользователем.

Определяем правило вида

repeat.

repeat: - repeat.

(вместо repeat можно использовать любое название предиката).

Такое правило задает бесконечный цикл. Цель предиката repeat – допустить бесконечность поиска с возвратом(бесконечное число откатов).

Пример: ввод и печать введённых символов до тех пор, пока не нажата клавиша Enter .

```
predicates
        nondeterm repeat
        nondeterm typer
clauses
        repeat.
        repeat:- repeat.
        typer:- repeat,
        readchar(C),
        write (C),
        C = '\r', !.
goal
        typer, nl.
```

Правило typer работает следующим образом.

1. Выполняет repeat, который ничего не делает, но ставит точку отката.
2. Присваивает переменной C значение символа.
3. Печатает C.
4. Проверяет соответствует ли C коду возврата каретки.
5. Если соответствует, то завершение. Если нет - возвращается к точке отката и ищет альтернативы. Так как ни write, ни readchar не являются альтернативами, постоянно происходит возврат к repeat, которое всегда имеет альтернативное решение.
6. Теперь обработка слова продвигается вперёд: считывает следующий символ, печатает его и проверяет на соответствие коду возврата каретки.

C теряет своё значение после отката перед вызовом предиката readchar(**C**).

Поиск с возвратом может повторять операции сколько угодно, но он не способен запомнить что-либо из одного повторения в другое.

```
predicates
nondeterm repeat
nondeterm typer
clauses
repeat.
repeat:- repeat.
typer:- repeat,
readchar(C),
upper_lower(C,G),      %преобразует введенные символы в символы нижнего регистра
```

```
write (C,G),
C = '\r', !.
goal
typer, nl.
```

Рекурсия.

Другой способ организации циклических вычислений в Прологе - рекурсия. Рекурсивная процедура – это процедура, которая вызывает сама себя. При этом задача разделяется на тривиальные(граничные) случаи и общие – через более простые.

Рекурсивная формулировка правила содержит в своём теле ссылку на заголовок этого же правила.

При описании рекурсии первое правило должно задавать исходное утверждение. Определение понятия “предок” через понятие “родитель”

```
предок ( X, Z ) : - родитель ( X, Z ).
предок(X, Z) : - родитель (X, Y),
                 предок ( Y, Z ).
```

Пример: вычислить факториал числа n.

Алгоритм задачи.

Найти n!

Если n=1, то 1!=1. Иначе найти (n-1)!и умножить на n.

```
predicates
factorial ( unsigned, real )
clauses
factorial ( 1, 1.0 ):-!.
factorial ( N, ResN ):-
M=N-1, factorial (M, ResM),
ResN=ResM*N.
goal
factorial ( 3, X ), write ( "fact 3=", X ).
```

Выполнение программы начинается с попытки достижения цели factorial (3, X). X – неопределён, хвостовая часть цели (оператор вывода) засылается в стек.

Переходим на первый вариант одноимённого правила: fact (1, 1): !.

3 < > 1, сопоставление заканчивается неуспехом, выбирается второе правило.

factorial (3, X)

↓ N = 3, ResN = X

M=N-1

factorial (2, ResM)

↓ M = 2, U= ResM

ResN=ResM*3— в стек

↓ N_1 = 2, $ResN_1$ = U

↓ M_1=1

U = 2* $ResM_1$. — в стек

↓ $N_2 = 1$
factorial (1, U1)
↓ N = 1 U1 = 1.0
и начинается выгрузка стека
↓
U = 2* ResM $_1$ =2*1 = 2
↓
ResN=ResM*3=2*3 = 6

Здесь при составлении аргументов подстановка N = 3, ResN = X
При обработке тела правила получаем M=2 и новый вызов factorial (2, ResM) и хвостовая часть ResN=ResM*3. (ResN=ResM*X.)засылается в стек, возвращаемся на fact (1, 1.0) –неуспех
и далее на fact (N, ResN) с новыми значениями $N_1 = 2$, $ResN_1 = U = 2*$ ResM $_1$. Далее в теле $N_2 = 2-1=1$, fact (1 , U1).
возвращаемся на factorial (1, 1.0) –успех и начинается выгрузка стека
Возвращаемся на factorial (2, 1*2), далее factorial (3, 2*3)
factorial (3, 6) и выполняется вывод результата.

Отсечение в factorial (1, 1) стоит для того, чтобы при вызове factorial (1, X) не вычислялся factorial (N, ResN), что привело бы к зацикливанию при вычислении факториала от отрицательных элементов.

Предикаты, стоящие правее рекурсивного вызова, не влияют на организацию рекурсии – они выполняются после выхода из неё и получают значения из стека, куда эти переменные попадают в ходе рекурсии. Производимые при этом вычисления называются хвостовыми. Информация хранится в области памяти, называемой стековым фреймом (stack frame) или просто стеком, который создаётся при вызове правила. Когда выполнение правила завершается, память, занятая стековым фреймом освобождается.

Рекурсия может потребовать много системных ресурсов, так как при рекурсивном вызове новые копии используемых значений помещаются в стек и сохраняются там, пока правило не завершится – успешно или неуспешно.

К бесконечным вычислениям приводит логический круг в определении, например:

родитель (X, Y): -ребёнок (Y, X).
ребёнок(Y, X): - родитель(X, Y).

Преимущества рекурсии:

Рекурсия имеет 3 преимущества:

- может выражать алгоритмы, которые нельзя удобно выразить никаким другим способом;
- логически проще метода итерации;
- широко используется в обработке списков.

Рекурсия – хороший способ для описания задач, содержащих в себе подзадачу такого же типа.
Но у рекурсии есть один большой недостаток – она съедает память.

Оптимизация хвостовой рекурсии.

Всякий раз, когда одна процедура вызывает другую, надо сохранять информацию о выполнении вызывающей процедуры, чтобы после выполнения вызванной процедуры возобновить выполнение на том же месте, где была остановлена.Это означает, что если процедура вызывает себя 100 раз , 100 различных состояний должно быть одновременно сохранено во фреймовом стеке. Максимальный размер стека на 16-разрядных платформах под DOS составляет 64 кбайт(3000-4000 стековых фреймов). На 32-разрядных платформах стек теоретически может достичь до нескольких гигабайт, но здесь проявятся другие системные ограничения, прежде чем стек переполнится.

Чтобы избежать использования большого стекового пространства применяют оптимизацию хвостовой рекурсии(tail recursion optimization) или оптимизацией последнего вызова (last-call optimization).

Чтобы задать хвостовую рекурсию, нужно чтобы

— Вызов являлся самой последней подцелью предложения;
— Ранее в предложении нет точек.

Пример хвостовой рекурсии, которая вызывает себя без резервирования стека и поэтому не истощает память

```
predicates
count(integer)
clauses
count(N) :-
write(N), nl,
NewN=N+1,
count(NewN).
Goal
nl, count (0).
```

Программа будет печатать целые числа, начиная с 0 и остановится от целочисленного переполнения, но остановки от истощения памяти не произойдет.

Задания для самостоятельной работы

1. **Программа запрашивает номер дня недели и выводит в ответ одно из сообщений: «Рабочий день», «Суббота», «Воскресенье» или «Номер дня недели должен быть от 1 до 7».**

2. **Квадратное уравнение имеет вид $ax^2+bx+c=0$. Вводя коэффициенты a, b, с найти решение для определенного квадратного уравнения.**

3. **Ряд 1, 1, 2, 3, 5, 8, 13, …. представляет собой ряд, называемый числами Фибоначчи. Каждый последующий член равен сумме двух предыдущих членов. Вычислить n-ый член ряда.**

4. **В программе вычисления факториала, данной в лекции, рекурсию преобразовать в хвостовую рекурсию. Использовать аргумент предиката в качестве параметра цикла.**

Лабораторная работа №

Списки и рекурсия.

Обработка списков, т.е. объектов, которые содержат произвольное число элементов – мощное средство VP. (Массивы без объявления размера)

Объявление:

```
domains
list = integer*
```

означает список элементов целого типа.

Элементы списка могут быть любые, включая другие списки, однако, все элементы должны принадлежать одному домену; если элементы разных типов нужно пользоваться функторами.

```
domains
element = element *
element = integer ; real ; symbol %  неверно
```

Чтобы объявить список из целых, действительных, символьных значений надо определить один тип, включающий все эти три типа с функторами, которые покажут, к какому типу относится тот или иной элемент

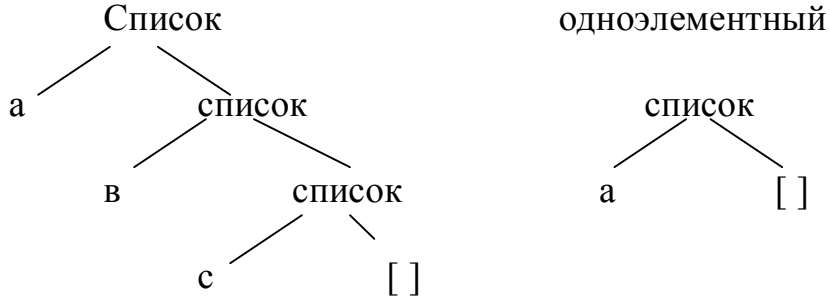
```
domains
elements = element *
```

element = i(integer); r(real); s(symbol) % i, r, s – функторы составных объектов.

Список является рекурсивным составным объектом. Он состоит из двух частей: головы – которой является первым элементом и хвоста, который является списком включающем все последующие элементы. Хвост списка – всегда есть список, голова списка – всегда элемент.

В списке [a, b, c] головой является a, хвостом – [b, c]. В одноэлементном списке [c] головой является c, а хвостом – []. Если выбирать первый элемент списка несколько раз, то обязательно дойдём до [].

Пустой список нельзя делить на голову и хвост. Структура списка имеет структуру дерева



Элементы разделяются запятыми.

Есть способ явно отделить голову от хвоста. Вместо разделения запятыми это можно сделать”|” (вертикальной чертой)

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

Головы и хвосты списков

Список	**Голова**	**Хвост**
`['a', 'b', 'c']`	`'a'`	`['b', 'c']`
`[ 1 ]`	`1`	`[] /* пустой список*/`
`[ ]`	`Не определено`	`Не определено`
`[[1, 2, 3], [2, 3, 4], []]`	`[1, 2, 3]`	`[[2, 3, 4], []]`

Несколько примеров унификации списков.

Унификация списков

Список 1	**Список 2**	**Связывание с переменной**
`[X, Y, Z]`	`[ann, bill,john]`	`X= ann, Y= bill, Z= john`
`[1]`	`[X | Y]`	`X=1, Y=[]`
`[1, 2, 3, 4]`	`[X, Y | Z]`	`X=1, Y=2, Z=[3,4]`
`[1, 2]`	`[3 | X]`	`Не совместимо`

Использование списков.

Список является рекурсивной составной структурой данных, поэтому нужны алгоритмы для его обработки. Главный способ обработки списка – это просмотр и обработка каждого его элемента, пока не будет достигнут конец. Алгоритму такого типа нужны 2 предложения. Первое из них говорит, что делать с обычным списком, который можно разделить на голову и хвост, второе указывает, что делать со вторым списком.

Примеры

1)Печать списков

```
domains
list = integer* % или другой тип
predicates
type_list (list)
clauses
type_list ([]).    % если список пустой, ничего не делать.
type_list ([H | T]): -
write (H), nl, type_list (T).
goal
type_list ([1, 2, 3]).
```

Здесь имеется 2 утверждения type_list, которые означают:

Печатать пустой список, т.е. ничего не делать.

Иначе печатать список, что означает печатать его голову (которая является одним элементом), затем печатать его хвост (список).
(Здесь мы видим хвостовую рекурсию).

2) Подсчет элементов списка (количества элементов).
Что такое длина списков?

1. Длина пустого списка [] равна 0.
2. Длина любого другого списка равна 1+ длина его хвоста.

```
domains
      list = integer*   % или char*
predicates
      length (list, integer)
clauses
      length ([], 0).
      length ([ _| T], L):-
      length (T, L1), L = L1 + 1.  % рекурсия ( не хвостовая рекурсия.)
     goal
           length ([1, 2, 3], L).
```

Алгоритм:
length ([1, 2, 3], L).
length ([2, 3], L1).
length ([3], L2).
length ([], 0).
L2 = 0+1 =1.
L1 = L2+1 = 2.
L = L1+1 = 3.

Правило ***length*** не является хвостовой рекурсией, поскольку рекурсивный вызов не является последним шагом. А можно ли получить хвостовую рекурсию?

Проблема заключается в том, что нельзя вычислить в ***length*** длину списка, пока не вычислите длину хвоста. Создадим предикат с тремя аргументами. Первый аргумент представляет список, который будет обрабатываться, пока не станет пустым. Второй – есть свободный аргумент, который будет подсчитывать результат вычисления длины списка. Третий есть счетчик, начинающий работу от 0 и увеличивающийся с каждым рекурсивным вызовом.
Когда список станет пустым, будет выполнена унификация счетчика с переменной, которая фиксирует результат вычисления.

```
DOMAINS
   list = integer*
PREDICATES
   length_of(list,integer,integer)
CLAUSES
   length_of([], Result, Result).
   length_of([_|T],Result,Counter):-
      NewCounter = Counter + 1,
      length_of(T, Result, NewCounter).
GOAL
    length_of([1, 2, 3], L, 0), % начало со счетчиком  Counter = 0
    write("L=",L), nl.
```

Получили хвостовую рекурсию. Использование аргумента в качестве параметра цикла. И логика ее вычисления более прозрачна, чем в предыдущем примере.

3. Преобразование одного списка в другой. Работаем со списком поэлементно, заменяя каждый элемент вычисленным значением.

```
domains
     list = real*
predicates
     positive (list, list)
clauses
     positive ([], []).   % граничные условия
      positive ([H| T], [H1| T1]):-     % отделить голову списка
     H1 = abs (H)*2,        % - вычислить новый элемент.
     positive (T, T1).          % вызвать элемент из оставшегося списка.
Goal          positive ([1, -5, 2, 6], Newlist).
Ответ   Newlist = [2, 10, 4, 12]
        1 Solution                  хвостовая  рекурсия.
```

4. Программа просматривает список из чисел и составляет новый список, отбрасывая отрицательные числа.

```
domains
  list = integer*
predicates
```

```
  discard_negatives(list,list)
clauses
  discard_negatives([],[]).
  discard_negatives([H|T],ProcessedTail):-
        H < 0,                          /* If H is negative, just skip it */
        !, discard_negatives(T, ProcessedTail).
  discard_negatives([H|T],[H|ProcessedTail]):-
        discard_negatives(T, ProcessedTail).
goal
  discard_negatives([2,-45,3,468],X).
```

Задания для самостоятельной работы

1. Вычислить при x=1, 2, 3, …,10:

$$y = x^2 + \cos(x)$$

$$z = \begin{cases} \sqrt{x^5 + y}\,, & \text{если } y \geq 10 \\ \cos(x) + \sin(x), & \text{если } y < 10 \end{cases}$$

2. С клавиатуры вводится рост и вес пользователя. Программа должна вычислить оптимальный вес пользователя по формуле: рост(в см)-110, затем сравнить с реальным и дать рекомендацию – похудеть или поправиться на «*x*» кг.
3. Из исходного списка чисел переписать в новый список четные числа.
4. Вывести все возможные варианты разбиения исходного списка на два новых списка.
5. Переписать список чисел в другой список, возведя каждый элемент в квадрат.
6. Посчитать произведение всех элементов списка чисел.
7. Ненулевые элементы исходного списка чисел переписать в новый список.
8. Посчитать количество отрицательных элементов списка чисел.
9. Из исходного списка чисел переписать в новый список элементы, кратные трем.
10. Даны показатели температуры воздуха на неделю. Найти среднее арифметическое значение t и количество дней, когда температура была ниже -10^0C.

Арифметические вычисления и сравнения.

Возможности вычислений и сравнений в Visual Prolog аналогичны возможностям других языков применения (C, Pascal, Basic)
Арифметические операции +, -, *, /, div, mod

Операнд 1	оператор	операнд 2	результат
целое	+, -, *	целое	целое
Целое или вещественное	+, -, *	Целое или вещественное	вещественное
Целое или вещественное	/	Целое или вещественное	вещественное

целое	div	целое	целое
целое	mod	целое	целое

Операции возведения в степень нет. Тип результата будет равен большему из размеров 2-х операндов по формуле $a^x = \exp(x*\ln(a))$

Порядок вычислений как обычно в арифметике, порядок можно изменить с помощью ( ).

Сравнение

Операторы отношения <, <=, =, >, >=, <> или ><

Сравнивать можно числа, выражение символы, строки и идентификаторы

5>3, x+5<y+1,(переменные должны быть связаны)

‘a’ <’b’ – символы (сравниваются коды 97<98)

“antony”>”antonia” - строки

p1 = peter, p2 = sally, p1>p2 – идентификаторы.

Когда сравниваются строки или идентификаторы, результат зависит от сравнения символов на соответствующих позициях. Сравнение до первых различных символов. Если конец достигнут, и пара различных символов не найдена, то - более короткая строка будет меньшей.

Функции и предикаты

x div y – частное от деления x на y

x mod y – остаток от деления x на y

abs (x) – модуль x

cos(x), sin(x), tan(x), arctan(x) x – в радианах, град = rad*180/π

exp (x) = e^x

ln (x) – натуральный логарифм

log (x) – десятичный логарифм

sqrt (x) = $\sqrt{x}$

round (x) – округленное значение x. Результат вещественный.

trunc (x) –усекает x. Результат вещественный.

val (domain, x) – явное преобразование между числовыми доменами

y = val (real, a*5+10)

Для генерации случайных чисел предусмотрены 2 предиката

random (X) – присваивает X случ. вещ. число от 0≤X<1

random (X, Y) – присваивает Y – случ. целое число 0≤Y<X

предикат randominit(S) инициализирует генератор случайных чисел

```
predicates
  person(integer,symbol) - nondeterm (i,o)
  rand_person(integer) - nondeterm (i)
clauses
  person(1,fred).
  person(2,tom).
  person(3,mary).
```

```
  person(4,dick).
  person(5,george).
  rand_person(0):-!.
  rand_person(Count):-random(5,N),
person(N,Name),
write(Name),nl,
NewCount=Count-1,
rand_person(NewCount).
goal
%time(Hours, Minutes, Seconds, RandomSeed)
%randominit(RandomSeed),
rand_person(3).
goal

randominit(24),
rand_person(7).
```

Запись и чтение в VP.

Запись осуществляется 3-мя стандартными предикатами

Это:

- Предикат write write (P1, P2,..)с производным числом параметров
- предикат nl, new line – переход на новую строку
- предикат writef

\ обратный слэш - управляющий символ. За ним следует специальный символ управления печатью:

‘\n’ – на начало новой строки

‘\t’ – табуляция

‘\r’ – возврат каретки

Предикат writef позволяет выполнить форматированный вывод; он имеет следующий вид: writef (format string, Arg1, Arg2,…)

Аргументы Arg1…- должны быть константами или связывающими переменными, принадлежащими стандартным доменам; сложные домены форматировать нельзя. Строка форматирования соединяет обычные символы и форматные спецификаторы. Обычные символы печатаются без изменений, а спецификаторы имеют вид

% - m. pf, где символы после % являются необязательными и имеют следующие значения:

- дефис показывает, что поле выравнивается слева (по умолчанию – справа)

m поле - десятичное число, определяющее минимальную длину поля.

p поле – десятичное число, описывает точность представления с плав. точкой

f поле - описывает формат поля, отличный от формата по умолчанию

спецификаторы формата поля f

f – вещественное с фиксированной точкой (123.4)

e – вещ. с экспоненциальным представлением (1, 234e2)

g – или f и e (что по умолчанию)

d – целые знаковые десятичные

u – целые без знаковые десятичные

o – целые восьмеричные

x – целые шестнадцатеричные

} если буквы заглавные то длинные целые long

c – символьные (char)

s – строки (string или symbol)

если буква формата не выбрана, то выберет автоматически подходящее.

```
goal
        A = "one",
        B = 330.12,
        C = 4.3333375,
        D = "one two three",
        writef("A = '%-7' \nB = '%8.1e'\n",A,B),
        writef("A = '%' \nB = '%8.4e'\n",A,B),nl,
        writef("C = '%-7.7g' \nD = '%7.7'\n",C,D),
        writef("C = '%-7.0f' \nD = '%0'\n",C,D),
        writef("char: %c, decimal: %d, octal: %o, hex: %x",'a','a','a','a').
```

Ответ

```
A = 'one    '
B = ' 3.3E+02'
A = 'one'
B = '3.3012E+02'

C = '4.333338'  (если будет не '%7.7', а '%7'то результат будет '4.3333375')
D = 'one two'
C = '4      '
D = 'one two three'
char: a, decimal: 97, octal: 141, hex: 61
A=one, B=330.12, C=4.3333375, D=one two three
1 Solution
```

Пример:

```
predicates
        nondeterm run
        nondeterm person(string,integer,real)
clauses
  person("Pete Ashton",20,11111.111).
  person("Marc Spiers",32,33333.333).
  person("Kim Clark",28,66666.666).

  run:-
```

```
        % Name   is left-justified, at least 15 characters wide
        % Age    is right-justified, 2 characters wide
        % Income is right-justified, 9 characters wide, with 2
        %        decimal places, printed in fixed-decimal notation

            person(N, A, I),
            writef("Name= %-15, Age= %2, Income= $%9.2f \n",N,A,I),
            fail.
    goal
      run.
```

Ответ

```
Name= Pete Ashton    , Age= 20, Income= $ 11111.11
Name= Marc Spiers    , Age= 32, Income= $ 33333.33
Name= Kim Clark      , Age= 28, Income= $ 66666.67
no
```

Чтение включает в себя несколько стандартных предикатов для чтения.
readln – для чтения всей строки символов.

readint, readreal, readchar – для чтения целых, вещественных и символьных значений.

readln - читает текстовую строку до 254 символов(плюс возврат кар) с клавиатуры и до 64Кбайт с других устройств. Если во время ввода с клавиатуры нажать <Esc>, то readln потерпит неудачу.

readint, readreal, readchar - принимает значение соответствующих типов, если вводимое значение не соответствует типу или нажав клавишу Esc, то ввод терпит неудачу.

Все эти предикаты могут быть переориентированы для чтения из файла. Другой более специализированный предикат, относящийся к категории чтения – это file_str – для чтения всего текстового файла в строку.

Predicate file_str - читает символ из файла в строку или создает файл и записывает в этот файл строку file_str (Filename, Text). Если переменная Text перед вызовом предиката свободна, то предикат читает из файла, пока не встретится конец файла. Читаемый файл не может превышать максимальную длину строки (64 кбайт в 16-битных платформах).

Например: file_str (“T. dat”, Mytext) свяжет Mytext с содержимым файла T.dat. Если после этой записи ввести file_str (“T. bak”, Mytext), то создастся файл с именем T.bak, куда записывается содержимое Mytext. Если файл существует, то произойдёт перезапись файла.

Рассмотрим примеры, как использовать стандартные предикаты чтения для работы с составными объектами и списками.

```
domains
        person = p(name, age, tel, job)
        age = integer
        tel, name, job = string
predicates
```

```
readperson(person)
run
clauses
        readperson (p(Name, Age, Tel, Job)):-
        write ("which name?"),   readln(Name),
        write ("Age?"), readint (Age),
        write ("Tel?"), readln (Tel),
        write ("Job?"), readln (Job).
        run :- readperson (P), nl, write (P), nl,
        write ("is this compound object (y/n)"),readchar(Ch),Ch ='y',!,write("end\n").
        run :- nl, write ("Try again"), nl, run.

Goal run.
```

2.Пример: чтение целых чисел и преобразование их в список.

Читает целое число на строке, пока не введем V не целое значение (например, символ) терпит неудачу и Visual Prolog выведет список на экран.

```
domains
        list = integer*
predicates
        readlist (list)
clauses
        readlist ([H| T]): - write("->"),
        readint (H), H<>1111, !, readlist (T).
        readlist ([]).
goal write ("spisok целых чисел),
        readlist (List), nl, nl,
        write ("The list is : ", List).
```

Классические предикаты VP.

Member дает возможность проверить

1) содержится ли элемент в списке и

2) содержится ли один список в другом (или добавлен ли один список к другому) соответственно.

Пример (Member – принадлежность элемента к списку).

У вас есть список имён и надо проверить содержится ли определённое имя в этом списке (предикат member(name, name list))

Если голова списка не совпадает с Name, то нужно проверить, можно ли найти Name в хвосте списка.

```
domains
        namelist = name*
        name = symbol
predicates
        member (name, namelist)
clauses
```

```
        member (Name, [Name|_]).
        member (Name, [_|Tail]): -
        member (Name, Tail).
goal
        member (john, [john, bill]).
        member (X, [john, bill]).
```

Пример (Member – принадлежность списка ко второму списку).

```
        domains
            namelist = symbol*
            predicates
        nondeterm member (namelist, namelist)
        clauses
            member ([Name|Tail], [Name|Tail]).
            member (Name, [_|Tail]):-
            member (Name, Tail).
        goal
            member (X, [john, bill]).
```

Ответ

```
X=["john","bill"]
X=["bill"]
2 Solutions
```

Предикат append задаёт отношение между 3 списками. Отношение сохраняется, если известен хотя бы один из списков.

Пример

```
domains
        list = integer*
predicates
        append(list, list, list)
clauses
        append ([], L, L).
        append ([H | L1], L2, [H | L3]): -        append (L1, L2, L3).
goal
        append ([1, 2, 3], [5, 6], L).
        append ([1, 2], [3], L), append (L, L, LL).    LL=[1,2,3,1,2,3]
```

Если один список пустой, то результатом, объединения первого и второго списков будет второй.

Если первый список непустой – то можно объединить первый и второй в третий список, сделав голову первого списка головой третьего списка.

Если задана цель append (L1, L2, [1, 2, 3]), то надо найти решения какие списки можно объединить, чтобы получить третий список?

Решения L1 = [], L2 = [1, 2, 3]
L1 = [1], L2 = [2, 3]
L1 = [1, 2], L2 = [3]
L1 = [1, 2, 3], L2 = []
4 solutions.

Если цель append [L, [3, 4], [1, 2, 3, 4]], то решение L1 = [1, 2].

Предикат append определил отношение между входным и выходным набором таким образом, что отношение применимо в обоих направлениях. Задавая отношение, задаём вопрос:

который выход соответствует данному входу?

или

который вход соответствует данному выходу?

Состояние аргументов при вызове предиката называется потоком параметров.

Аргумент, который присваивается или назначается в момент вызова, называется входным аргументом и обозначается буквой i, а свободный аргумент – это выходной аргумент и обозначается буквой o.

Предикат append имеет возможность работать с разными потоками данных, но не все предикаты таковы.

Определение:

Если предложение Пролога может быть использовано с различными потоками параметров оно называется обратимым.

Встроенный предикат findall

Мы рассмотрели, как организовывать повторы при нахождении решений с помощью бэктрекинга и рекурсии. Преимущество рекурсии в том, она

может передавать информацию (через аргументы) от одного рекурсивного вызова к другому

и может хранить информацию о порядке рекурсивных вычислений и промежуточных значениях аргументов.

Но есть вещи, где рекурсия не может конкурировать с бэктрекингом. Она не может представить все альтернативные решения для заданной цели, как это может сделать бэктрекинг. Рассмотрим задачу, где требуется найти все решения для заданной цели, при этом нужно найти все решения для составной структуры данных, рассматривая ее как единое целое.

Для этих целей можно воспользоваться встроенным предикатом ***findall,*** который использует цель, как один из своих аргументов, и реализует все решения, собирая их в список.

Предикат имеет вид ***findall(****VarName,* ***mypredicate,*** *ListParam)*, где

- Первый аргумент *VarName*, специфицирует, какой аргумент предиката будет коллекционироваться в списке.

- Второй аргумент ***mypredicate,*** указывает на предикат, из которого будут изыматься значения для коллекции.
- Третий аргумент – это переменная *ListParam,* которая собирает все специфицированные значения в список.

Пример: определить средний возраст группы людей.

```
DOMAINS
  name,address = string
  age = integer
  list = age*

PREDICATES
  nondeterm person(name, address, age)
  sumlist(list, age, integer)

CLAUSES
  sumlist([],0,0).
  sumlist([H|T],Sum,N):-
    sumlist(T,S1,N1),
    Sum=H+S1, N=1+N1.

  person("IVANOV", "PUSHKINA 22", 42).
  person("PETROV", "GOGOLYA 10", 36).
  person("SEMENOV", "MIRA 44", 51).

GOAL
    findall(Age,person(_, _, Age),L),
    sumlist(L,Sum,N),
    Ave = Sum/N,
    write("Average=", Ave),nl.
```

Здесь клоз ***findall*** создает список *L*, в котором собираются возрасты людей. Эти значения извлекаются из предиката ***person***.

Если Вы захотите отобрать людей с фиксированным возрастом, полезно использовать следующую подцель (возраст 42 года):

```
findall(Who, person(Who, _, 42), List)
```

Правда, здесь вам потребуется переопределить коллекционируемый список:

```
slist = string*
```

Составные списки

Мы уже знаем, что список из целых чисел можно объявить

```
integerlist = integer*
```

То же самое можно сделать для вещественных чисел; списка символов или строк.

Однако часто в сложных структурах объектов требуется задавать смешанные типы данных, примерно как такие:

```
[2, 3, 5.12, ["food", "goo"], "new"]  /* Не корректно! */
```

Составные списки состоят из элементов более чем одного типа данных. Чтобы специфицировать такие типы, Вам потребуются особые приемы, поскольку ***Visual Prolog требует, чтобы элементы списка принадлежали одному типу данных.*** Выход из положения – использовать функторы.

И в этом случае можно работать с различными типами элементов, ***поскольку домен может содержать более чем один тип данных, как аргументы к функтору.***

Рассмотрите следующий пример. Здесь даны неправильная и правильная декларации сложных списков

```
DOMAINS                    /* здесь l, i, c и s - функторы */
  llist = l(list); i(integer); c(char); s(string)
  list = llist*
```

Некорректное объявление списка:

```
  [ 2, 9, ["food", "goo"], "new",'C' ]
```

Корректное:

```
  [i(2), i(9), l([s("food"), s("goo")]), s("new"),c('C')]
```

Следующий пример с предикатом ***append*** показывает, как манипулировать такого рода списками.

Программа 6.8

```
DOMAINS
  llist = l(list); i(integer); c(char); s(string)
  list = llist*

PREDICATES
  append(list,list,list)

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

GOAL
    append([s(nike), l([s(bill), s(mary)])],[s(bill), s(sue)],Ans),
    write("FIRST  LIST: ", Ans,"\n\n"),
    append([l([s("This"),s("is"),s("a"),s("list")]),s(bee)], [c('c')],Ans2),
    write("SECOND LIST: ", Ans2, '\n').

    ANSWER

    FIRST  LIST: [s("likes"),l([s("bill"),s("mary")]),s("bill"),s("sue")]



    SECOND LIST: [l([s("This"),s("is"),s("a"),s("list")]),s("bee"),c('c')]

    Ans=[s("likes"),l([s("bill"),s("mary")]),s("bill"),s("sue")],
    Ans2=[l([s("This"),s("is"),s("a"),s("list")]),s("bee"),c('c')]

    1 Solution
```

Лабораторная работа №

Обработка строк в VP.

Visual Prolog поддерживает набор стандартных предикатов для эффективной обработки строк. Условно разделим их на две группы:

базовые предикаты управления строками

и предикаты преобразования типов строк в другие и наоборот.

Основные предикаты управления строкой

1. frontchar (String1, Char, String2)

```
                  /          |          \
          строка     1символ     остаток
                    1-ой строки  строки
```

Можно использовать для расщепления строки в последовательность символов или для создания строки из последующих символов (или для проверки символов в строке)

Пример:

```
domains
        charlist =char*
predicates
        string_list(string, charlist)
clauses
        string_list ("", []):- !.
        string_list (S, [H | T]):- frontchar(S, H, S1),
        string_list (S1, T).
goal
        string_list ("ABC",Z).
```

в результате Z = [‘A’, ‘B’, ‘C’].

2. fronttoken (String1, Lexema, String2)

Для разделения строки String1 на лексические символы (лексемы).

Лексемой является последовательность символов, если она представляет собой

1. имя в соответствии с синтаксисом Visual Prolog
2. число (предшествующий ему знак является отдельной лексемой).
3. отличный от пробела знак

Пример:

```
domains
    namelist = name*
    name = symbol
predicates
    string_fr(string, name list)
clauses
    string_fr(S, [ H|T ]):- fronttoken(S, H, S1),!,
    string_fr(S1, T).
    string_fr(_,[]).
goal
    string_fr(“bill fred dick”, X).
```

Ответ X = [bill, fred, dick].

3. frontstr (NumberofChars, String1, Startstr, Endstr)
где Startstr содержит NumberofChars первых символов из String1, а Endstr содержит остаток.

4. concat (String1, String2, String3) устанавливает что String3 является результатом сцепления String1 и String2.

5. str_len (string Arg, Length) определяет или проверяет длину строки или возвращает строку пробелов заданной длины.

6. isname (String) проверяет является ли аргумент допустимым именем согласно синтаксису VP.

7. subchar (String, Position, Char) возвращает символ на данной позиции строки.

subchar(“ABC”, 2, Char) свяжет Char с символом B

8. substring (Str_in, Pos, Len, Str_out) возвращает часть строки.
Str_out будет равен копии части строки Str_in, начиная с символа на позиции Pos и длиной Len.

substring (“ABCDE”, 2, 3, Substr)
результат Substr = “BCD”

9. searchchar (String, Char, Position) возвращает позицию первого появления заданного символа в строке.

searchchar (“ABCD”, ‘A’, Pos) свяжет Pos с 1

Предикат ищет только первое вхождение символа, чтобы найти последующие вхождения надо написать программу

```
Domains
 i=integer
 s= string
 c= char
predicates
nondeterm  poisk (s, c, i, i)  % Pos – номер позиции в исходной строке
nondeterm  p (s, c, i, i, i)      %Pos1-номер позиции в оставшейся части строки
clauses
    poisk (Str, Ch, Pos, Old):- searchchar (Str, Ch, Pos1),      %Pos1=3  =4
    p(Str, Ch, Pos, Pos1, Old).
    p(_, _, Pos, Pos1, Old):- Pos = Pos1 + Old.                  %  Pos=3 =7
    p(Str, Ch, Pos, Pos1, Old):- frontstr (Pos1, Str, _, Rest),
    Old1 = Old+ Pos1,                                            %Old1=3 =7
    poisk (Rest, Ch, Pos, Old1).                                 % print Pos=3  =7
goal
    poisk ("ktabulate", 'a', P,0),
    write (P, '\n'), fail.
```

10. searchstring (SourseStr, SearchStr, Pos) возвращает позицию первого появления строки в другой строке.

searchstring (“ABCDEF”, “BC”, Pos) свяжет Poc с 2

Аналогично как в предыдущем составить программу.

```
Domains
 i=integer
 s= string
predicates
nondeterm  poisk (s, s, i, i)
nondeterm  p (s, s, i, i, i)
clauses
        poisk (Str, Ch, Pos, Old):- searchstring (Str, Ch, Pos1),
        p(Str, Ch, Pos, Pos1, Old).
        p(_, _, Pos, Pos1, Old):- Pos = Pos1 + Old.
        p(Str, Ch, Pos, Pos1, Old):- frontstr (Pos1, Str, _, Rest),
        Old1 = Old+ Pos1,
        poisk (Rest, Ch, Pos, Old1).
goal
        poisk ("tabulabt", "ab", P,0),
        write (P, '\n'), fail.
```

Преобразование типов

Стандартные предикаты для образования типов char_int

1. char_int (Char, Integer) – преобразует символ в целое число или целое число в символ (этот предикат не является необходимым в новых версиях, так как происходит автоматическое преобразование типов, оставлен для совместимости со старыми версиями).
2. str_char (String, Char) – преобразует строку, содержащую один и только один символ в символ или символ в строку из одного символа.
3. str_int (String, Integer) преобразует строку, содержащую целое число, в целое число или целое число в его текстовое представление.
4. str_real (String, Real) преобразует строку в вещественное число или вещественное число в строку.
5. upper_lower (Upper, Lower) преобразует строку, содержащую символы верхнего регистра в строку соответствующих символов нижнего регистра и наоборот.

goal

Str1 = “STRING”, Str2 = “string”,

upper_lower (Str1, Str2).

6. tem_str (Domain, Term, String) является универсальным предикатом для преобразований, он совершает преобразование терма заданного домена в его строковое представление.

Domain задает какому домену принадлежит Term

term_str заменяет str_*

```
str_real (S, R) :- term_str (real, R, S).
term_str (real, R, "6.1").
```

Пример преобразования строки в список лексем, определение типа лексемы.

```
domains
tok = numb (integer); name (string); char (char)
toklist = tok*
predicates
```

```
nondeterm      scanner (string, toklist)
nondeterm      maketok (string, tok)
clauses
scanner ("", []).
scanner (Str, [Tok|Rest]):- fronttoken (Str, Sym, Str1),
maketok (Sym, Tok),
scanner (Str1, Rest).

maketok (S, name (S)):- isname (S).
maketok (S, numb (N)):- str_int (S, N).
maketok (S, char (C)):- str_char (S, C).
goal
        write ("Enter some text:"), nl,
        readln (Text), nl,
        scanner (Text, T_List),
        write (T_List).
```

Преобразование между symbol и string, а также между char, integer, real производятся автоматически при использовании предикатов

Пример:

```
predicates p (integer)
clauses
    p (X):- write ("The integer value is",X).
goal
    %X = 97.234, p (X).
    %X = 97, p (X).
    X = 'a', p (X).
```

Во всех трех случаях ответ будет один и тот же 97.

Задания для самостоятельной работы

1. Программа должна определить есть ли во введенном предложении слово «Prolog».
2. Написать программу, определяющую количество вхождений слова «язык» в строке – «Логический язык программирования – это язык Пролог».
3. Составить программу, определяющую может ли введенная строка являться идентификатором в языке Пролог.

Лабораторная работа №

Файловая система в Visual Prolog.

Visual Prolog использует текущее устройство чтения (по умолчанию клавиатура) и текущее устройство записи (по умолчанию экран дисплея).

Можно переназначить устройства ввода-вывода, например, читать из файла и записать в файл.

Независимо от типа устройства чтение и запись обрабатываются в Visual Prolog идентично.

Для доступа к файлу, сначала он должен быть открыт или для чтения, или для записи, добавления или модификации.

Файл, открытый для записи, добавления или модификации после завершения операции должен быть закрыт во избежание потерь изменений.

При работе с файлами в Visual Prolog действительное имя файла в OS связывается с символическим именем, и затем это символическое имя используется для направления ввода-вывода.

Символические имена файлов должны начинаться с маленькой буквы и должны быть объявлены в разделе

domains

file = file1; source

В программе разрешен только один домен file. Visual Prolog распознает пять встроенных альтернатив домена file: keyboard (клавиатура), stdin (стандартное устройство ввода), screen(запись на монитор), stdout (стандартное устройство вывода), stderr(стандартное устройство для вывода ошибок).

Они не должны встречаться в описании file, открывать и закрывать их не требуется.

Стандартные предикаты для открытия и закрытия файлов

Примечание: Вы должны использовать два знака наклонной черты влево, когда Вы задаете путь в программе; например:

```
"c:\\prolog\\prim.dat"
```

1. Предикат openread (SymbName , NameFOS) открывает файл NameFOS для чтения.
2. Предикат openwrite (SymbolicName, NFOS) открывает файл для записи, если файл уже существует, то содержимое файла уничтожается.
3. Предикат openappend (SymbN, NfOS) открывает файл для записи в конец файла.
4. Предикат openmodify (Symb, OS) открывает файл и для записи и для чтения. Если он существует, он не будет перезаписан.
5. Предикат closefile(SymbNameFile)закрывает указанный файл.
6. Предикат readdevice(SymbNf) переопределяет текущее устройство чтения, если переменная SymbNf определена и файл открыт для чтения. Если же переменная SymbNf является свободной, то присвоит переменной имя текущего активного устройства чтения.
7. Предикат writedevice(SymbNF) либо назначает, либо позволяет получить текущее устройство записи.

Пример:

Записать символы, набранные на клавиатуре, в файл proba.dat. Ввод прекращается при нажатии клавиши #.

```
domains
        file = myfile
predicates
        readloop
clauses
```

```
readloop:- readchar(X), X<>'#',!, write(X), readloop.
readloop.
goal
openwrite(myfile, "C:\\proba.dat"), writedevice(myfile), readloop,
closefile(myfile), writedevice(screen).
```

8. Предикат filepos(SymbNf, FilePosit, Mode) указывает позицию для чтения или записи.

Mode - целая величина, указывающая режим отсчета позиций и может принимать значения 0, 1, 2:

0 - позиция относительно начала файла,
1 - относительно текущей позиции,
2 - относительно конца файла.

Например: Запись текста в файл somefile.pro, начиная с 100 позиции, отсчет с начала файла

```
domains
file=myfile
goal
Text = "A text to be written in the file", openmodify(myfile, "somefile.txt"),
writedevice(myfile), filepos(myfile, 100, 0), write(Text), closefile(myfile).
```

9. Предикат eof(SymbNf) проверяет является ли позиция, полученная в процессе чтения, концом файла.

10. Predicate existfile(OSFN) проверяет существует ли файл.

```
open(File, Name): - existfile(Name),!, openread(File, Name).
open( _, Name): - write("Error: the file not found").
```

11. deletefile(OSFN).
12. renamefile(oldOSFN, NewOSFN).
13. disk(Path) – для изменения текущего диска/каталога.
14. copyfile(SourceName,DestinationName) можно указывать полное имя файла, т.е. путь к файлу.

Задания для самостоятельной работы

1. Записать символы, набранные на клавиатуре, в файл proba.dat и вывести их на экран монитора. Ввод прекращается при нажатии клавиши #.

2. Программа должна записать целые числа, введенные с клавиатуры, в файл number.dat на диск C: в папку Prolog.