Дисциплина: Функциональное и логическое программирование.

Лабораторная работа №2.

Механизм поиска решений.

Время: 180 мин.
Что нужно освоить:
1) каким образом задавать внутреннюю цель программы;
2) как работает бэктрекинг;
3) как уговорить Пролог совершить полный перебор всех вариантов реализации внутренней цели.

Introduction

Приступая к выполнению данной лабораторной работы вы должны уже разбираться в следующих вопросах:

- что такое декларативное программирование и в чем его отличие от процедурного;
- все программы Пролога представляют из себя базы знаний, включающие два вида предложений - факты и правила;
- факты содержат безусловные утверждения, а правила – условные;
- совокупность правил с одинаковым предикатом в заголовке принято называть процедурой;
- суть выполнения программы на языке Пролог сводится к поиску ответа на вопрос, предъявляемый к базе знаний;
- процесс поиска ответа на вопрос – это процесс последовательной унификации, то есть сопоставление вопроса с предложениями программы.

Однако в рамках первой лабораторной работы мы лишь обозначили некоторые возможности описания мира в Прологе через определение отношений на основе фактов и правил. Возможности логического программирования на языке Пролог расширяются за счет использования дополнительных механизмов, в частности, задание данных через структуры, списки, определение отношений на основе рекурсивных правил.

Сегодня мы продолжим изучать основы логического программирования – в ходе выполнения работы предстоит освоить:

- способы работы с внутренней и внешней целями;
- порядок включения программного кода в текст программы из другого файла;
- способы создания своих и использования встроенных логических функций.

Кроме того, вам предстоит сделать первые шаги в определении отношений на основе рекурсивных правил.

Часть 1. Следи за собой, будь осторожен !

И начнем мы с описания потенциальных возможностей совершения ошибки в логической программе. Зачастую такие ошибки выявляются только при расширении базы знаний. В этой части уточним также особенности работы Пролога с внутренней целью, в частности отличие процесса унификации внутренней цели от внешней.

Здесь уместным окажется материал из книги Норберта Винера «Творец и робот» / 1963 г.

> Джэкобс, английский писатель начала XX века, выдержка из повести «Обезьянья лапа».
>
> В повести описывается английская рабочая семья, собравшаяся к обеду на кухне. Сын вскоре уходит на фабрику, а старики родители слушают рассказы своего гостя, старшего сержанта, возвратившегося из Индии. Гость рассказывает об индийской магии и показывает талисман - высушенную обезьянью лапу. По его словам, индийский святой наделил этот, талисман магическим свойством исполнять по три желания любого из трех своих последовательных владельцев. Гость говорит, что это подходящий случай испытать судьбу. Он не знает первых двух желаний предыдущего владельца талисмана, но ему из-

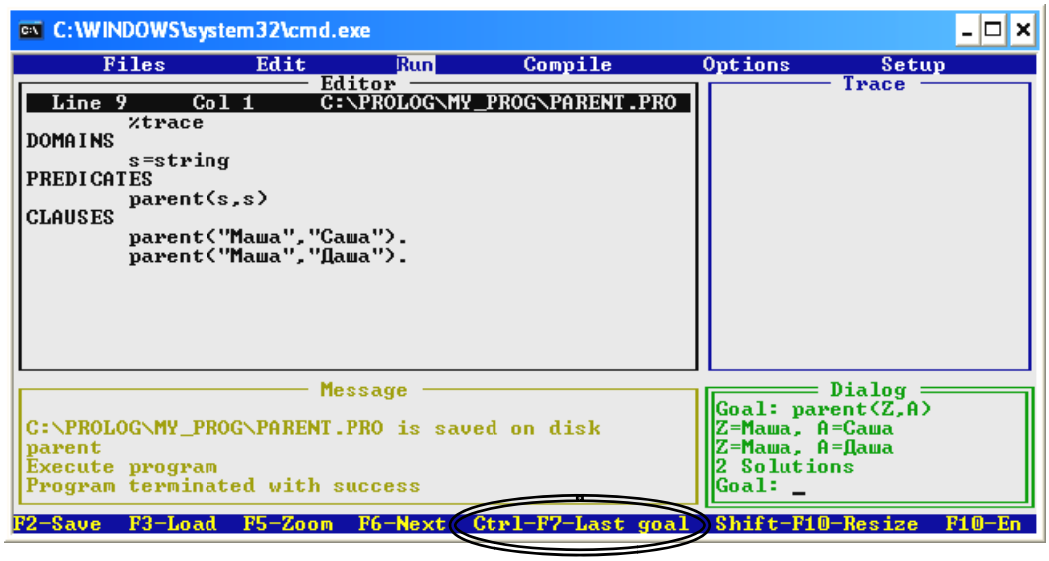
вестно, что последним желанием его предшественника была смерть. Сам он был вторым владельцем, но то, что он испытал, слишком страшно пересказывать. Гость уже намерен бросить волшебную лапу в камин, но хозяин выхватывает ее и, невзирая на предостережения, просит у нее двести фунтов стерлингов.

Через некоторое время раздается стук в дверь. Входит очень важный господин, служащий той фирмы, в которой работает сын хозяина. Со всей мягкостью, на которую он способен, господин сообщает, что в результате несчастного случая на фабрике сын хозяина погиб. Не считая себя ни в коей мере ответственной за случившееся, фирма выражает семье погибшего свое соболезнование и просит принять пособие в размере двухсот фунтов стерлингов. Обезумевшие от горя родители умоляют - и это их второе желание, - чтобы талисман вернул им сына... Внезапно все погружается в зловещую ночную тьму, поднимается буря. Снова стук в дверь. Родители каким-то образом узнают, что это их сын, но, увы, он бесплотен, как призрак. История кончается тем, что родители выражают свое третье и последнее желание: они просят, чтобы призрак сына удалился.

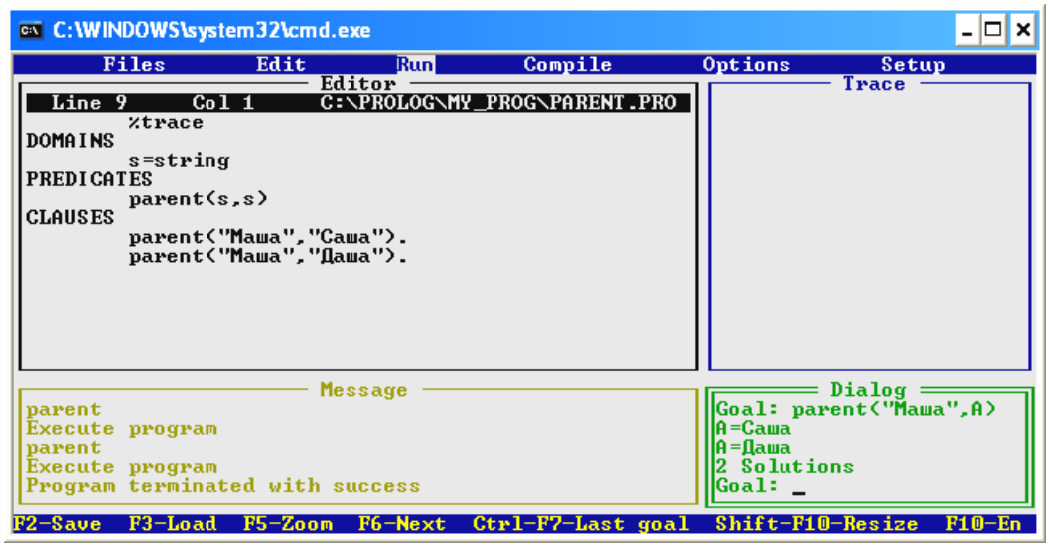
Эта история – напоминание о том, машины осуществляют любую деятельность в высшей степени буквально. Они будут делать именно то, что вы попросили, а не то что вы подразумевали, но не сумели точно сформулировать. Машина действует только в рамках заложенных в неё правил и все попутные ущербы, сторонние для её миропонимания, её не интересуют. На этом основаны некоторые фантастические фильмы о будущем мире с кибернетическими монстрами.

В логическом программировании существует риск получения неправильных или избыточных решений в том случае, когда программа первоначально тестировалась на ограниченном подмножестве фактов и правил, а в дальнейшем эксплуатируется на ином, более широком подмножестве. Рассмотрим пример.

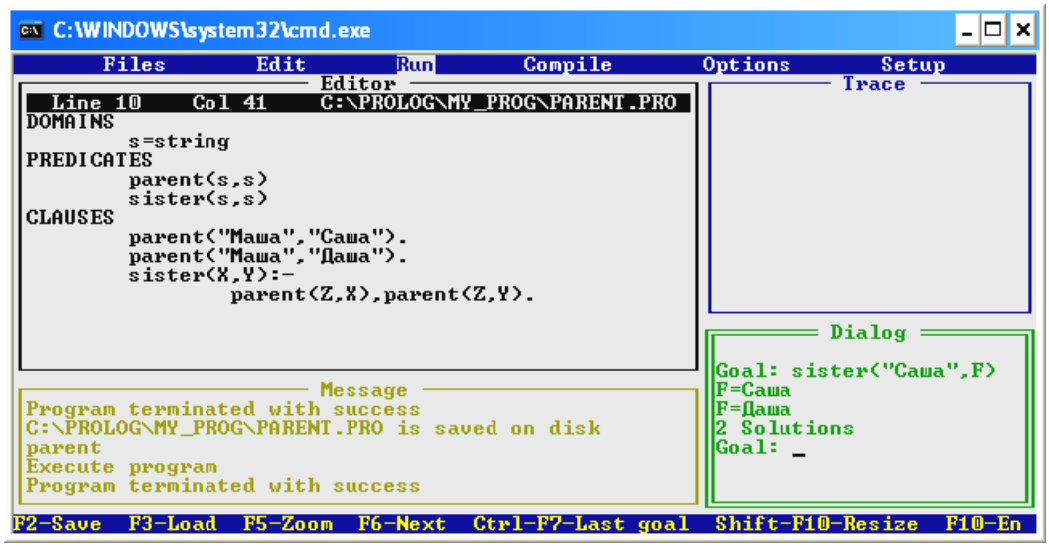
Пусть есть семья: мама Маша и её дочки Саша и Даша. Предположим, что для описания этой ситуации мы используем двухаргументный предикат parent (родитель). Первый аргумент этого предиката задает родителя, а второй его ребенка. Запустите систему Пролог, наберите приведенный ниже текст программы, сохраните его под именем parent.pro в папку C:\PROLOG\MY_PROLOG, запустите её на выполнение (Alt-R), в диалоговом окне задайте цель parent(Z,A). Объясните полученный результат (см. скриншот).



Давайте модифицируем цель. Нажмите **Ctrl-F7** для возвращения последней набранной цели и Z замените на “Маша”. Пусть Пролог выполнит реализацию этой цели. Объясните полученный результат (см. следующий скриншот).

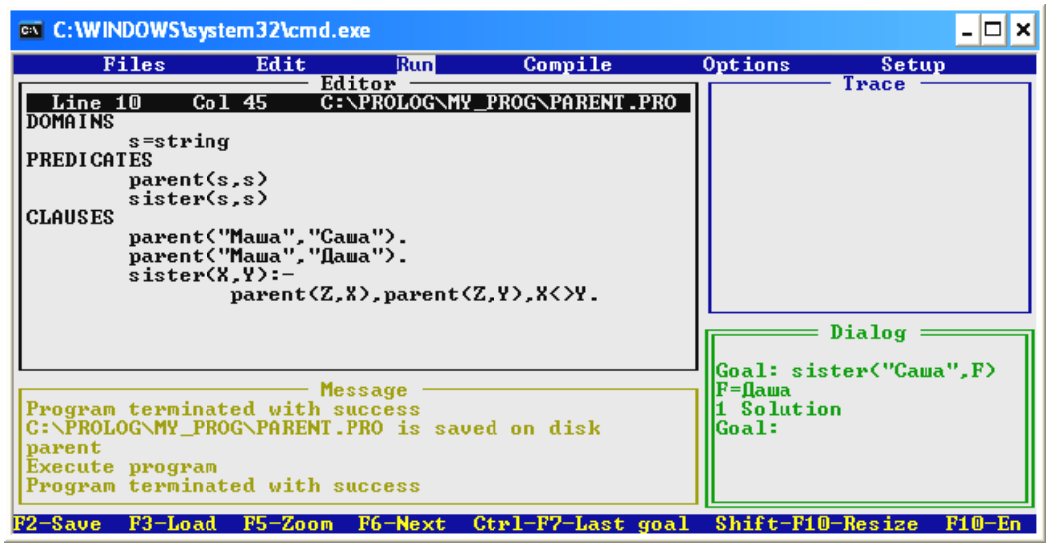


Дополним программу предикатом sister, задающим отношение сестра между двумя субъектами. Правило для выполнения этого предиката, предполагает выполнимость двух подцелей: первый и второй аргументы являются сестрами, если у них один родитель. После запуска видоизмененной программы для поиска цели - «Кто есть сестра Саши?» - мы получим как правильное решение («Даша») так и неправильное («Саша») (см. скриншот).

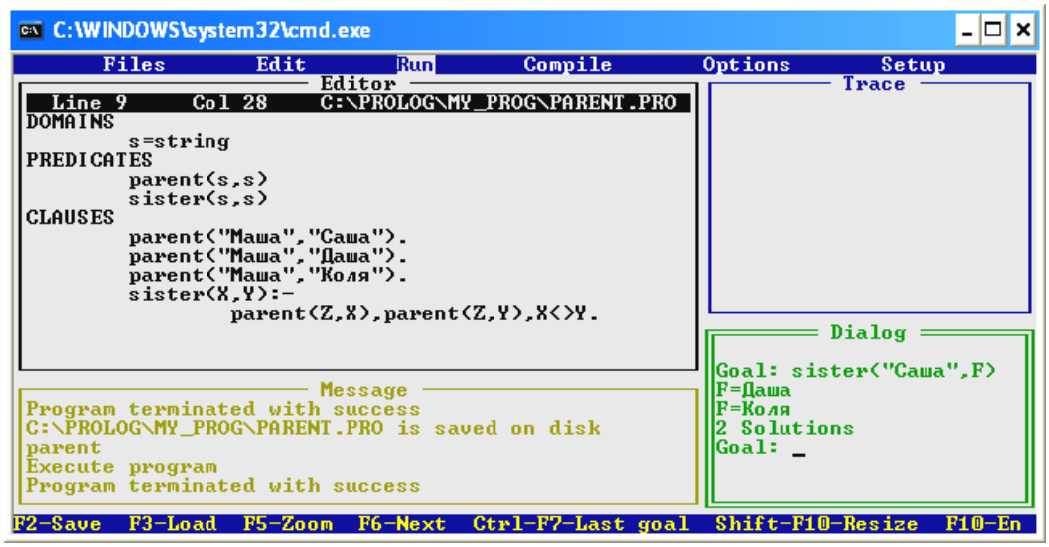


Но оно неправильное именно с нашей точки зрения. Мы же понимаем, что Саша не является сестрой самой себе. Сделайте трассировку программы, добавив в начало предикат trace, и объясните результат.

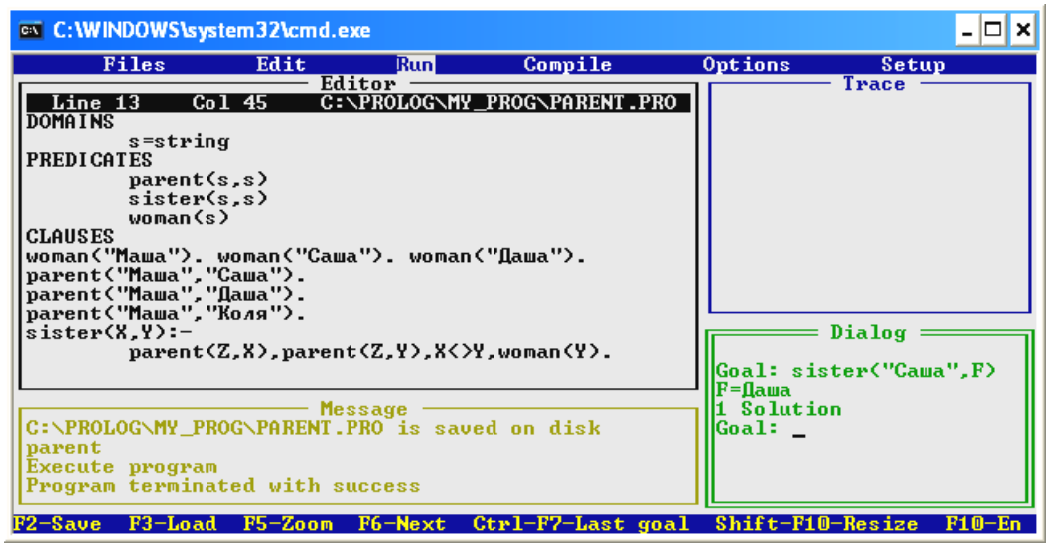
Пролог в этой ситуации не виноват, так как это мы допустили неточность. Внесем изменения, уточнив, что субъект не может быть сестрой самого себя. После запуска программы мы избавимся от неверного ответа (см. следующий скиншот).



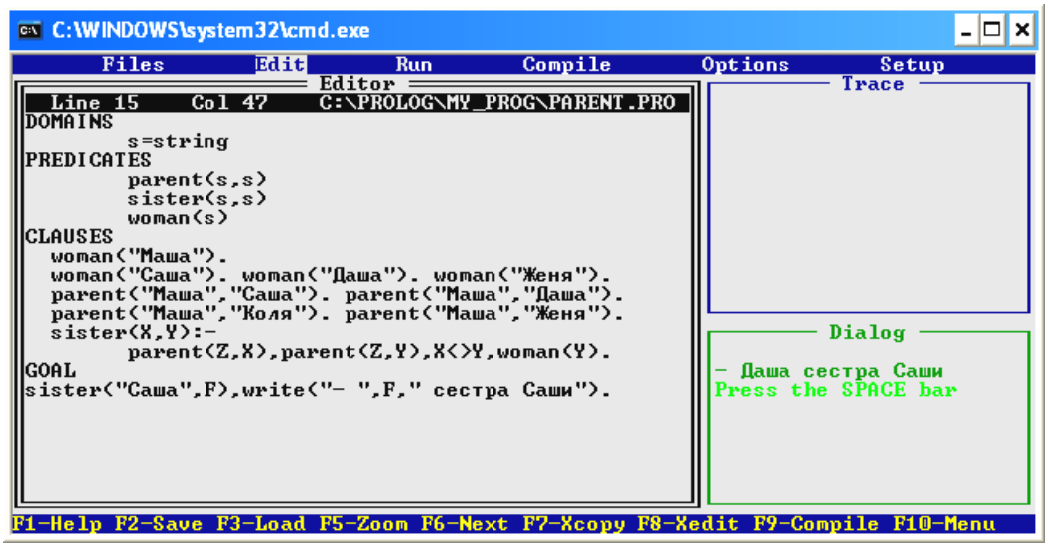
Однако, если ***в дальнейшем***, кто-то ***добавит*** в текст программы такой факт «у Маши есть сын Коля», то поиск сестры Саши опять даст один неверный результат (см. скриншот).



Для нас очевидно, что Коля не может быть сестрой не только Саши, но и вообще ничьей сестрой, но Прологу об этом никто не сказал. Придется добавить в описание правила, что сестра может быть только женского пола и перечислить тех, кто является женщинами (см. следующий скриншот).

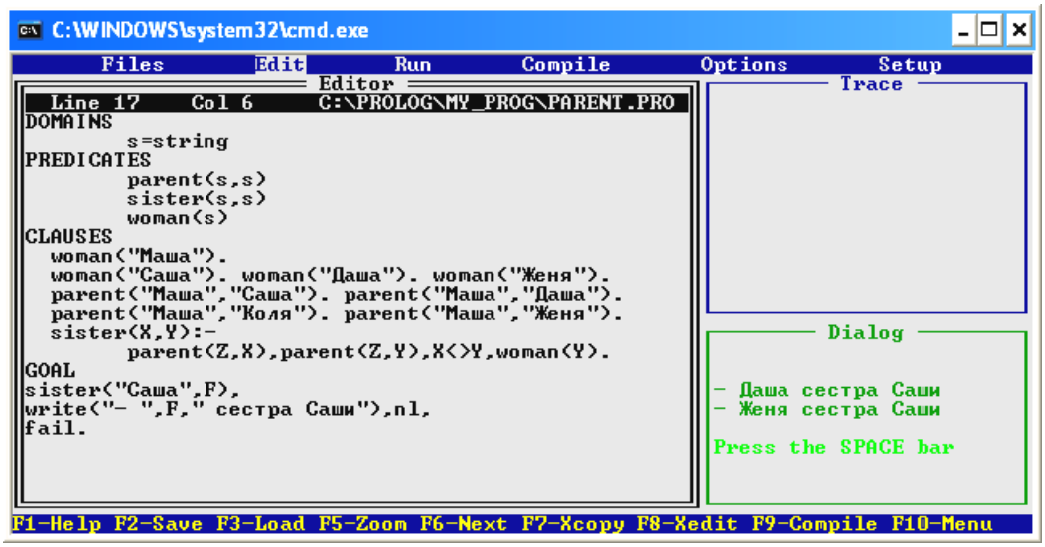


Но может возникнуть неприятность при переходе к внутренней цели. Пусть у Маши есть ещё одна дочь – Женя. В этом случае поиск цели sister(“Саша”,F) будет завершен сразу после отыскания первого же подходящего варианта, то есть “Даши”.



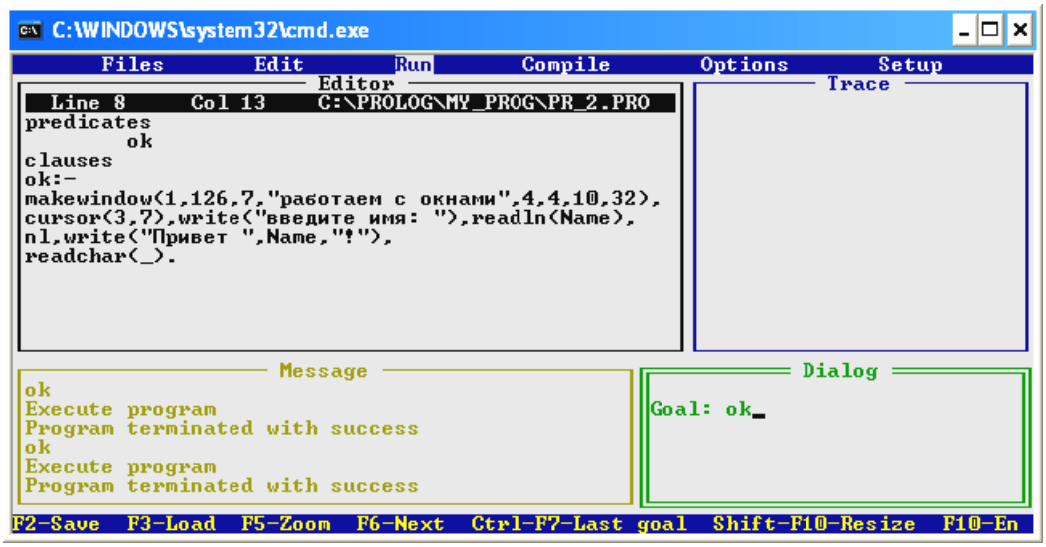
Тем не менее существует способ корректно принудить Пролог совершить полный перебор. Для того, чтобы заставить Пролог не останавливаться на достигнутом, необходимо каждый раз после отыскания приемлемого варианта унификации цели указывать ему, что эта цель, на самом деле, недостижима. В этом случае Пролог будет пытаться искать альтернативные варианты унификации. Чтобы текущий вариант не терялся, его нужно как-нибудь зафиксировать, например, вывести текущий ответ на экран.

Как именно реализуется подобный механизм? Необходимо в цель добавить стандартный предикат fail, который переводится как “потерпеть неудачу”. Этот предикат безусловный и когда Пролог добирается до него, то происходит то же самое, как если бы повстречалась такая запись “1=2”. Подцель fail (или “1=2”) невыполнима, что приводит Пролог в состояние, при котором он делает вывод, что общая цель недостижима и, если ещё есть альтернативные варианты унификации, то можно приступить к их рассмотрению. То есть, в данном случае, обнаружив что Даша есть сестра Саши, Пролог выводит этот результат на экран, делает переход на следующую строку (предикат nl – “new line”) и сталкивается с непреодолимой проблемой в виде fail, которая не дает ему остановиться в поиске (см. скриншот).



Попробуйте поставить вместо fail какую-нибудь заведомо невыполнимую подцель и проверьте работоспособность программы. Оцените результаты с помощью трассировки.

Внутренняя цель программы, конечно, несет больше преимуществ, чем проблем для программиста. Будем с этим постепенно разбираться. В частности давайте вспомним программу с прошлой лабораторной работы. Вот эту:



Здесь стандартный предикат readchar с анонимной переменной просто необходим, так как удерживает окно диалога с пользователем до того момента пока не получит какой-нибудь символ (т.е. нажатие любой клавиши).

В этой программе отсутствует внутренняя цель. Каждый раз, когда вы её запускаете, то приходится сообщать Прологу, что вам собственно от него нужно. Это не всегда удобно. Можно поступить иначе. Добавьте к тексту программы две строчки:

```
goal
ok.
```

И можете убрать предикат readchar. В нем теперь отпадает необходимость. Запустите программу в таком варианте и ощутите разницу.

Кроме того, только при наличии внутренней цели в программе её можно откомпилировать в отдельный исполняемый файл.

Часть 2. «Всё своё ношу с собой» или как оставить что-нибудь не нужное.

> Когда родной город Бианта (*герой древнегреческой мифологии*) был осажден войсками полководца Кира, жители стали убегать, захватывая самое ценное своё имущество. Один лишь Биант ничего не взял с собой. На вопрос удивленных горожан, почему он так уходит, Биант ответил: "Всё своё я ношу с собой".
>
> Греческая притча

Часть программы, например, базу знаний (совокупность фактов и правил), иногда хочется хранить отдельным файлом, чтобы можно было её использовать в любой другой программе. Для реализации этой цели можно использовать стандартный предикат include. Так как этому предикату надо указать что за файл ему нужно включить в программу, то возможны два варианта его использования.

Вариант №1.

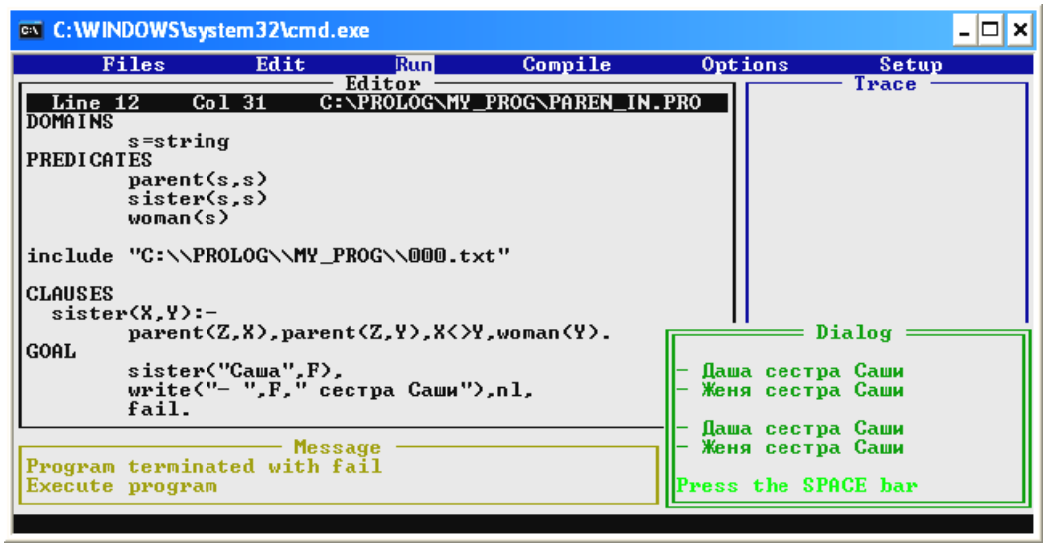
Формат использования: `include "диск\\путь к файлу\\имя файла"`, например, `include "C:\\Prolog\\MY_PROG\\000.txt"`. Обратите внимание, что в Прологе принято в описании пути использовать двойной обратный слеш.

Рассмотрим, как это работает на примере программы про многодетную семью из *части 1*. Все факты, касающиеся определения пола членов семьи и отношений родитель – литя вынесем в отдельный файл `000.txt`.

Содержимое файла `000.txt`:

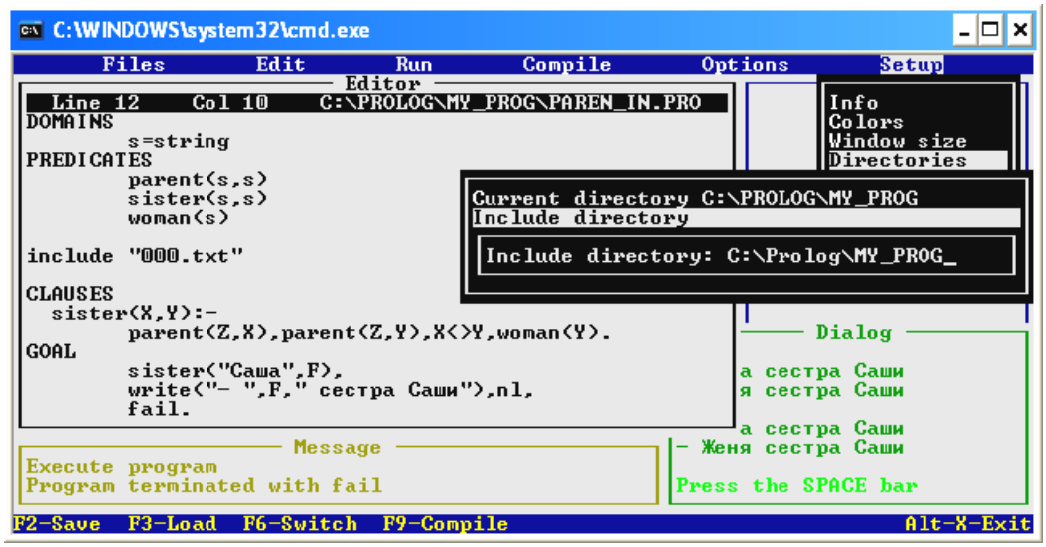
```
clauses
 % факты
woman("Маша").
woman("Саша").
woman("Даша").
woman("Женя").
parent("Маша","Саша").
parent("Маша","Даша").
parent("Маша","Коля").
parent("Маша","Женя").
```

Тогда текст программы с внесенными изменениями будет выглядеть следующим образом:



Вариант №2.

Формат использования: `include "имя файла"`, например, `include "000.txt"`. При этом Прологу следует отдельно указать где именно этот файл `000.txt` находится. Путь к файлу можно определить через меню (см. скриншот).



Подобным образом можно во внешнем файле хранить любую часть программы и включать её по мере необходимости. Попробуйте, к примеру, следующий вариант:

Текст программы:

```
include "111.txt"
GOAL
     ok.
```

Содержимое файла 111.txt:

```
DOMAINS
 s=string
PREDICATES
 parent(s,s)
 sister(s,s)
 woman(s)
 ok
CLAUSES
  woman("Маша"). woman("Саша").
  woman("Даша"). woman("Женя").
  parent("Маша","Саша"). parent("Маша","Даша").
  parent("Маша","Коля"). parent("Маша","Женя").
  sister(X,Y):-
 parent(Z,X),parent(Z,Y),X<>Y,woman(Y).
  ok:-
 sister("Саша",F),
 write("- ",F," сестра Саши"),nl,
 fail.
```

Часть 3. Логические функции.

Проведите кропотливую работу с данной частью лабораторной работы, и вы сделаете значительный шаг вперед на пути к логическому программированию.

Давайте составим программу построения таблицы истинности для разных логических функций (не используя стандартные предикаты логических функций). Начнем с функции «И».

Для компактности написания я не буду оформлять отдельного окна, а вывод организую непосредственно в системное окно Dialog. Вам же я рекомендую посвятить некоторое время конструированию дизайна окна вывода и таблицы в нем.

Итак, таблица будет представлена названием и заголовком:

```
GOAL
 write("таблица И"),nl,
 write("A B Z"),nl,
 i.
```

Здесь A и B есть входные переменные, а Z – значение логической функции. Предикат i следовательно предусмотрен для формирования четырех строк (0 0 0; 0 1 0; 1 0 0; 1 1 1). Обсудим его работу. Он должен делать возрастающий перебор всех вариантов сочетаний значений входных данных и выдавать соответствующее значение искомой функции:

```
i:-
 fact(A), fact(B),
 i(A,B,X),
 write(A," ",B," ",X), nl, fail.
```

Здесь первые две подцели обращаются к фактам определяющим возможные значения переменных A и B. Следовательно необходимо включить в программу описание этих фактов:

```
 fact(0). fact(1).
```

Далее, третья подцель обращается к предикату для определения значения логической функции «И» по двум входным переменным A и B. После определения значения оно

выводится на экран совместно с входными параметрами - `write(A," ",B," ",X).` Далее реализуется перенос строки и предикатом `fail` принудительно признается цель недостижимой, что приводит к рассмотрению альтернативных вариантов (для фактов `fact(A), fact(B)`).

Теперь организуем процедуру поиска значения логической функции «И» по двум входным переменным.

Самый простой вариант видимо выглядит так:

```
i(0,0,0).
i(0,0,1).
i(1,0,0).
i(1,1,1).
```

По сути это факты, перебирая которые пролог будет выдавать нам ответы. Однако, не всегда поставленная задача будет столь тривиальной. Давайте попробуем другие варианты оформления процедуры поиска значения функции.

Например, так:

```
i(A,B,1):-A>0,B>0,!.
i(A,B,0).
```

Или так:

```
i(A,B,X):-
      A+B=2,X=1,!;
      X=0.
```

Все это конечно интересно, но эти варианты создавались без учета последовательности унификации (сверху-вниз и слева-направо). Если перебор невелик, то можно и не уделять этому внимание. Но некорректно построенная процедура может стать камнем преткновения при большом количестве проверок и затянуть время унификации до неприлично больших величин.

Разберем подробнее. Итак, для исследуемой функции вариант с ответом «1» только один, но именно его мы и проверяем каждый раз в первом предложении процедуры. В то время как вариант с ответом «0» встречается в три раза чаще. Если его поставить на первое место, то отсечение будет срабатывать чаще, снижая тем самым общее число выполняемых действий:

```
i(A,B,X):-
      A+B<2,X=0,!;
      X=1.
```

Моя настоятельная рекомендация состоит в том, чтобы начиная с самых маленьких процедур вы учитывали тот факт, что в последующем они могут войти в другую более объемную программу и сильно испортить общую производительность.

Итак, советы общего плана по написанию программ:

1) не забывайте наглядно и однообразно оформлять все предложения в программе;

2) оставляйте комментарии (так **`% комментарий`** или так **`/* комментарий */`**);

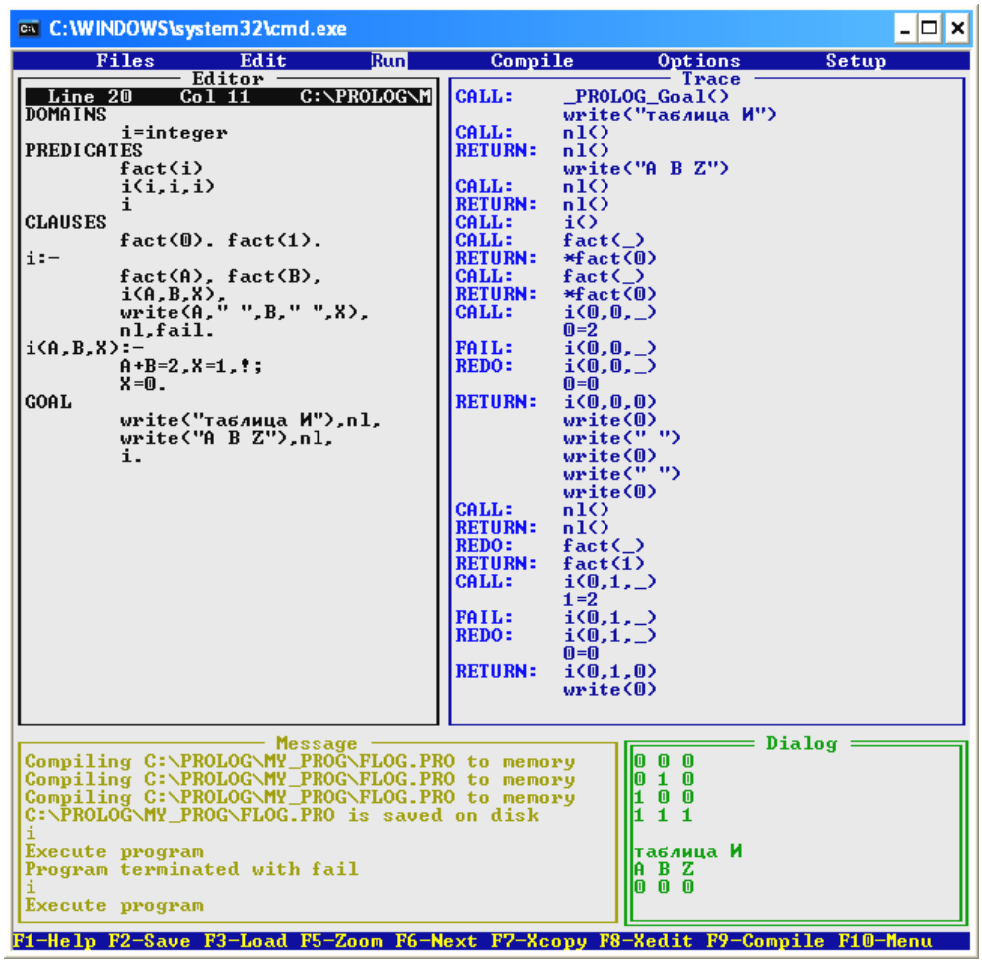
3) давайте идентификатор (имя) предикатам, так чтобы интуитивно был понятен их смысл;

4) учитывайте порядок сопоставления предложений в процедуре.

А теперь задание для самостоятельного исполнения.

Разработайте и испытайте процедуры поиска и вывода на экран таких логических функций как: «НЕ», «ИЛИ» *(цена: 4 балла)*. Их необходимо выполнить без использования стандартных (встроенных) предикатов.

В качестве помощи предлагаю использовать следующий скриншот:



Проверьте, как проходит пошаговое исполнение процесса унификации цели. Очевидно, что на данном скриншоте присутствует только часть трассировки.

Далее найдите в помощи пролога описание логических функций в исполнении стандартных предикатов. Разработайте программу и исследуйте предикаты: bitand, bitor, bitxor, bitnot, bitleft и bitright (*цена: 6 баллов*).

Ещё одно контрольное задание *(цена: 9 баллов)*.

На основе изученного материала самостоятельно разработайте следующую базу знаний.

Факты:

– пусть в виртуальном мире есть несколько объектов (например, фрукты, книги – разные, в количестве 5-8 шт.) – их наличие задать **фактами**;

– пусть в мире есть несколько субъектов (заданы именами, в количестве 4-5 субъектов) – **фактами** перечислены их предпочтения по отношению к объектам.

Правила:

1) найти по имени субъекта все объекты, входящие во множество его предпочтений;

2) найти по имени субъекта все объекты, не входящие во множество его предпочтений.

Запрос к базе знаний оформить в виде **внутренней цели**. При запуске программы на экран выводится сообщение с просьбой ввести имя субъекта и номер запроса (1 или 2, смотри описание правил выше), в ответ программа выводит на экран список объектов.