

**Лекция 8.**  
**Логическое программирование**  
**на Прологе**

## **Логическое программирование.**

Рассуждая логически, логическое программирование - программирование, основанное на логике. Часто его определяют как использование языка логики в качестве языка программирования, вместе с использованием логического вывода в качестве вычислительного механизма. Это очень хорошее определение, но оно очень широкое. Логическое программирование, определенное таким образом, включает в себя то же функциональное программирование как и многие другие стили программирования.

Исторически сложилось так, что термином "логическое программирование" обозначают использование в качестве языка программирования некоторого подмножества чистой логики первого порядка. Это означает, что для выражения концепции действия применяется математическое понятие отношения или предиката.

Часто встречается еще более узкое толкование понятия логического программирования, когда язык ограничивается применением хорновских предложений, а механизм вывода - определенным вариантом метода резолюций. На этих принципах создан Пролог и сходные с ним языки программирования. В основе этих языков лежит возможность интерпретировать одно и то же выражение, как логическое утверждение

A, если  $B_1$  и  $B_2$  и ...  $B_n$

и как определение процедуры

```
чтобы выполнить A
  выполнить  $B_1$ ;
  выполнить  $B_2$ ;
  . . .
  выполнить  $B_n$ .
```

Но такое узкое понятие следует признать слишком ограничительным.

Многие языки программирования заслуживающие названия "логических" не вписываются в эту схему. С другой стороны, тот же Пролог содержит элементы логики второго порядка. А некоторые свойства Пролога при казуистическом подходе вообще не позволяют отнести его к декларативным языкам.

Тем не менее, Пролог остается наиболее распространенным языком логического программирования. и именно его мы будем использовать. На примере Пролога мы рассмотрим специфические свойства логических программ, которые делают логическое программирование хорошо подходящим для некоторых применений, таких как дедуктивные базы данных, синтаксический анализ (особенно неоднозначных грамматик) и сложные комбинаторные задачи.

Дальнейшее развитие логических языков продолжается в двух главных направлениях, которые можно назвать "алгоритмическим" и "переборным". Первое ориентировано главным образом на решение задач, для которых известны эффективные алгоритмы и требуются более тонкие средства управления, чем присутствующие в Прологе. Второе ориентировано на задачи, для которых нет эффективных алгоритмов, и где существенное место занимает перебор вариантов. Языки этого направления заменяют несколько примитивный перебор с возвратами, характерный для Пролога, более изощренными методами.

## Пролог

Разнообразие диалектов Пролога не столь велико, как у Лиспа и различия между ними не так существенны. Для определенности будем использовать [SWI-Prolog](#), который, как почти все реализации в основном следует знаменитому эдинбургскому Прологу для DEC-10.

После запуска интерпретатор Пролога выдает приглашение вводить *запросы*, которые также называют *вопросами* или *целями*. Как правило, запрос представляет собой вызов процедуры и записывается как имя процедуры, за которым в круглых скобках следуют аргументы, разделенные запятыми. В конце запроса ставится точка. Процедуры в Прологе называют *предикатами*. Некоторые из них встроены в язык и их можно вызывать немедленно.

```
?- true.  
Yes  
?- fail.  
No
```

Встроенные предикаты `true` и `fail` представляют соответственно тождественно истинное и тождественно ложное высказывание. Ответ на первый вопрос всегда "да", а на второй "нет". Говорят, что запрос `true` заканчивается *успехом*, а `fail` - *неудачей*. Точно так же любой запрос завершается либо успехом, либо неудачей.

```
?- integer(1).
```

*Yes*

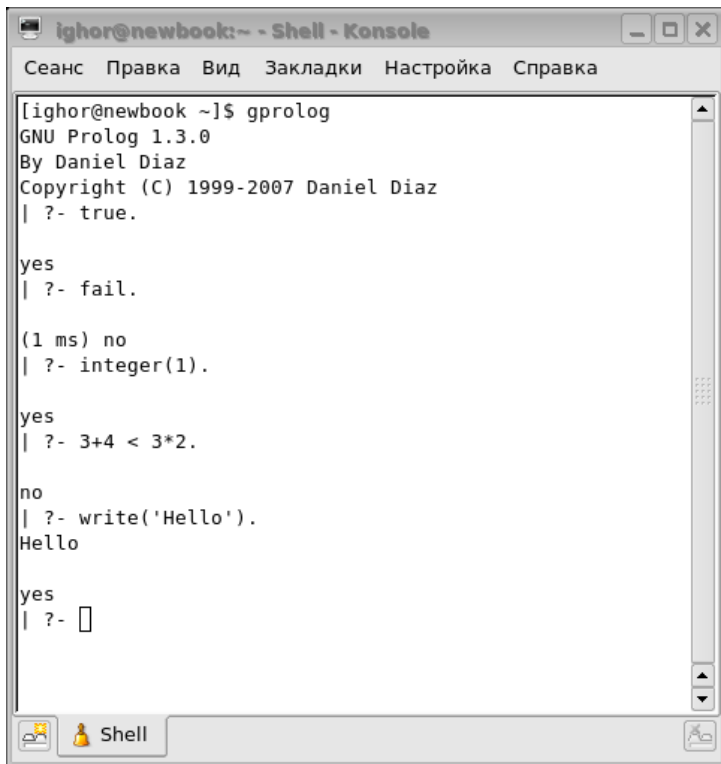
```
?- 3+4 < 3*2.
```

*No*

```
?- write('привет').
```

привет

*Yes*



```
ighor@newbook:~ - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка

[ighor@newbook ~]$ gprolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- true.

yes
| ?- fail.

(1 ms) no
| ?- integer(1).

yes
| ?- 3+4 < 3*2.

no
| ?- write('Hello').
Hello

yes
| ?- []
```



Обратите внимание: между именем предиката и открывающей скобкой не должно быть пробелов. Второй запрос записан в традиционной инфиксной форме - некоторые предикаты допускают такую запись. Выполнив последний запрос, Пролог отвечает Yes, хотя мы вроде бы ни о чем не спрашивали. Таким образом он просто сообщает о благополучном завершении запроса.

Есть еще одна возможность - запрос может завершиться ошибкой.

```
?- abracadabra.
```

```
ERROR: Undefined procedure: abracadabra/0
```

```
?- 3/0 > 1.
```

```
ERROR: //2: Arithmetic: evaluation error: `zero_divisor'
```

```
ighor@newbook:~ - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка

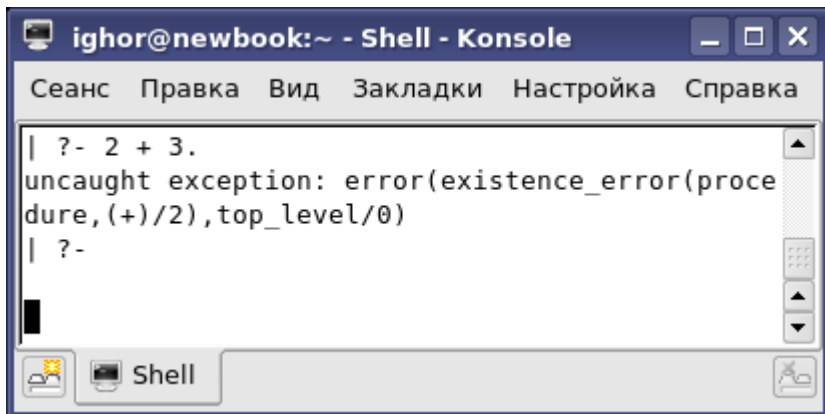
| ?- abracadabra.
uncaught exception: error(existence_error(procedure,abracadabra/0),top_level/0)
| ?- 3/0 > 1.
uncaught exception: error(evaluation_error(zero_divisor),(>)/2)
| ?- 
```

Поскольку системе не известно ничего по поводу `abracadabra`, возникает исключительная ситуация. Во втором случае ошибка возникла в процессе арифметических вычислений. В принципе, ошибка - тоже неудача, но неудача особого рода, после которой дальнейшее выполнение запроса не имеет смысла.

Имея некоторый опыт работы с функциональными языками, мы можем попытаться найти сумму чисел

?- 2 + 3.

*ERROR: Undefined procedure: (+)/2*



The image shows a terminal window titled "ighor@newbook:~ - Shell - Konsole". The window has a menu bar with "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The main area contains the following text:

```
| ?- 2 + 3.  
uncaught exception: error(existence_error(procedure,(+)/2),top_level/0)  
| ?-  
█
```

At the bottom of the window, there is a taskbar with a "Shell" tab and a search icon.

и получаем сообщение об ошибке. Наш запрос не имеет формы предложения.

Попробуем по другому.

```
?- write(2+3).
```

```
2+3
```

```
Yes
```

The image shows a terminal window with a dark blue title bar containing the text "ighor@newbook:~ - Shell - Konsole <2>". Below the title bar is a menu bar with the items "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The main area of the terminal displays the following text:

```
[ighor@newbook ~]$ gprolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- write(2+3).
2+3

yes
| ?- █
```

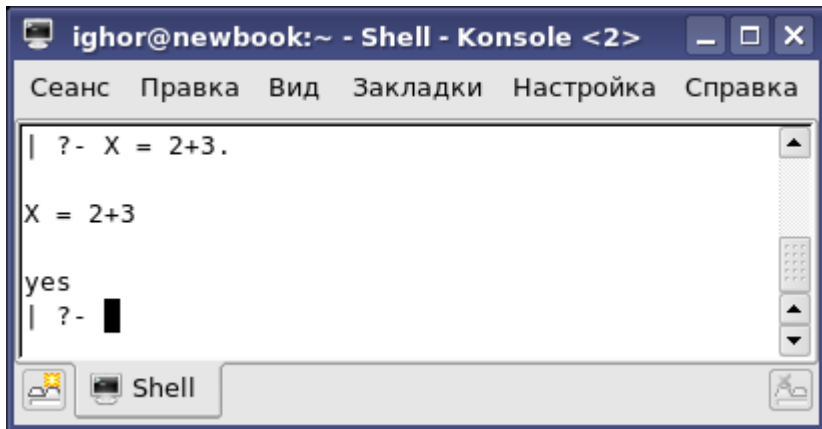
At the bottom of the terminal window, there is a taskbar with a "Shell" icon and a "Shell" label. On the right side of the terminal window, there are standard window controls: a vertical scrollbar and two arrow buttons (up and down).

Система выводит выражение, даже не пытаясь что-нибудь вычислить. Может быть так?

?- X = 2+3.

X = 2+3

Yes



```
igor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- X = 2+3.
X = 2+3
yes
| ?- █
```

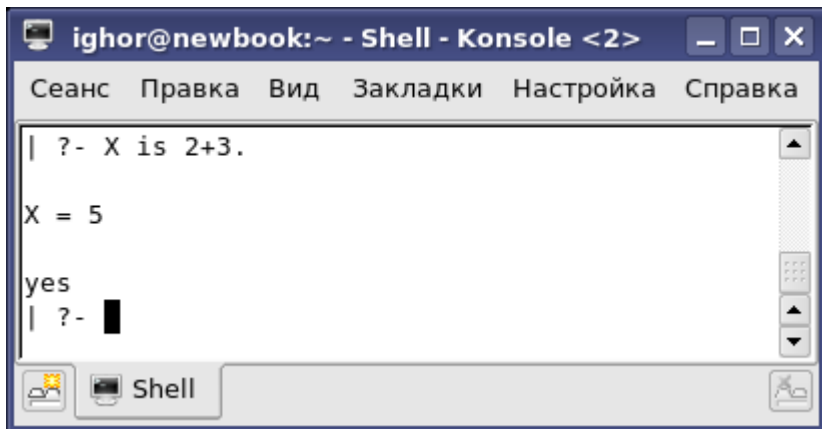
Снова неудача. Пролог разрабатывался прежде всего как язык обработки символьной информации и выражения вида "2+3" - это его объекты данных (они называются *термами*). Именно это выражение и присваивается переменной X. Но надо же как то выполнять арифметические вычисления. Для этого предназначены специальные *арифметические предикаты*.

```
?- X is 2+3.
```

```
X = 5
```

```
Yes
```





The image shows a terminal window titled "ighor@newbook:~ - Shell - Konsole <2>". The window has a menu bar with "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal content is as follows:

```
| ?- X is 2+3.  
  
X = 5  
  
yes  
| ?- █
```

At the bottom of the terminal, there is a taskbar with a "Shell" button and a search icon.

Наконец то! Предикат `is` вычисляет значение выражения и присваивает его переменной. Запрос завершается успехом и значение переменной выводится в качестве ответа.

Другие арифметические предикаты (  $:=$ ,  $\neq$ ,  $<$ ,  $=<$ ,  $>$ ,  $>=$  )  
вычисляют оба своих аргумента и сравнивают их значения.

?-  $3 + 2 := 5$ .

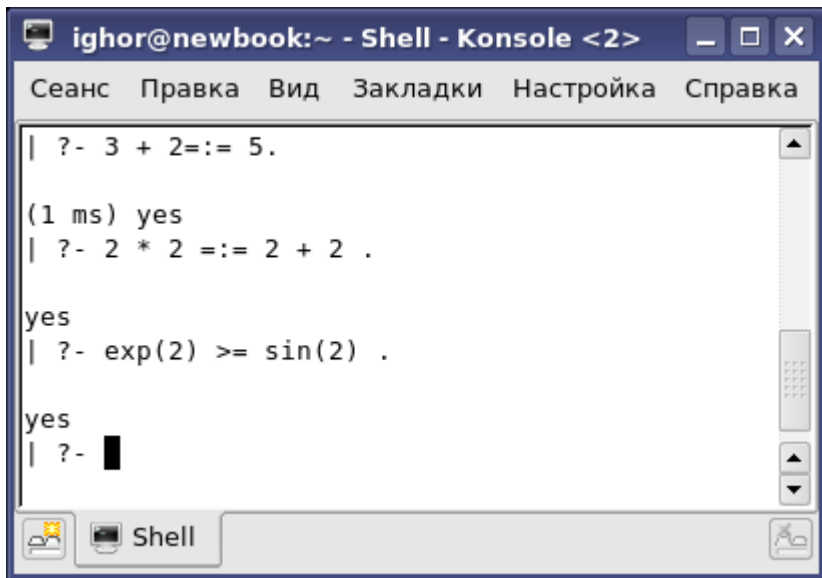
Yes

?-  $2 * 2 := 2 + 2$  .

Yes

?-  $\exp(2) >= \sin(2)$  .

Yes



```
ighor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- 3 + 2== 5.
(1 ms) yes
| ?- 2 * 2 == 2 + 2 .
yes
| ?- exp(2) >= sin(2) .
yes
| ?- █
```

Другая разновидность примитивных объектов - *атомы*. Чаще всего они записываются как последовательности букв и цифр, начинающейся со *строчной* буквы. Это отличает их от переменных, которые начинаются с *прописной* буквы. Атомы можно сравнивать посредством предикатов `==`, `\==`, `@<`, `@=<`, `@>`, `@>=`.

Арифметические предикаты для этого не подходят, поскольку они попытаются вычислить арифметические значения атомов, что приведет к ошибке. А вот числа (но не значения выражений) можно сравнивать и теми и другими.

```
?- a @< b.
```

```
Yes
```

```
?- 1 @< 2.
```

```
Yes
```

```
?- 2 < a.
```

```
ERROR: Arithmetic: `a/0' is not a function
```

```
?- 2 < e.
```

```
Yes
```

The image shows a terminal window with a dark blue title bar containing the text "ighor@newbook:~ - Shell - Konsole <2>" and standard window control buttons. Below the title bar is a menu bar with the items "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The main area of the terminal is white with black text. It contains the following text:  
| ?- a @< b.  
  
yes  
| ?- 1 @< 2.  
  
yes  
| ?- 2 < a.  
uncaught exception: error(type\_error(evaluable,  
a/0), (<)/2)  
On the right side of the terminal area, there are vertical scroll arrows and a keyboard icon. At the bottom of the window, there is a taskbar with a "Shell" icon and a "Shell" label, and a printer icon on the far right.

Почему последний запрос успешен, догадайтесь сами.

```
?- X is 2+3.
```

```
X = 5
```

```
Yes
```

```
?- Y is X+1.
```

```
ERROR: Arguments are not sufficiently instantiated
```

```
ighor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка

| ?- X is 2+3.
X = 5
yes
| ?- Y is X+1.
uncaught exception: error(instantiation_error,(
is)/2)
| ?- █
```

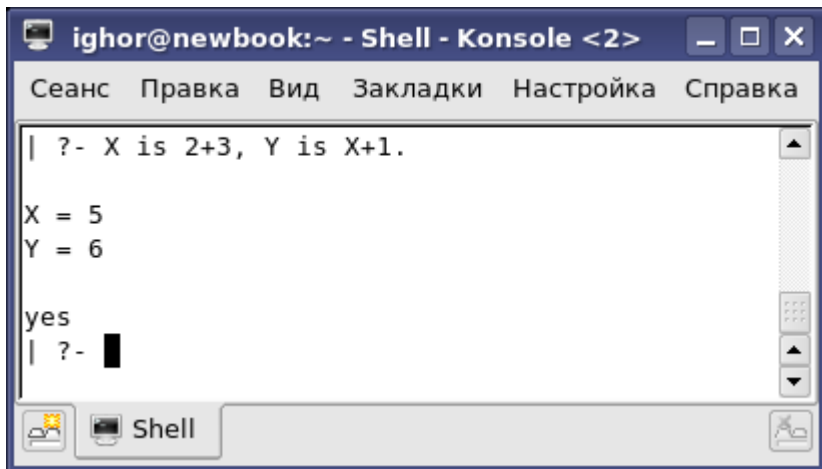
В сообщении об ошибке указано что переменная(имеется в виду X) не конкретизирована. Но мы же только что присвоили ей значение! Дело в том, что это уже другая переменная. Область действия переменной распространяется только на один запрос. Но запросы могут быть и сложными, состоящими из нескольких целей.

?- X is 2+3, Y is X+1.

X = 5

Y = 6

Yes



The screenshot shows a terminal window titled "ighor@newbook:~ - Shell - Konsole <2>". The window contains the following text:

```
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- X is 2+3, Y is X+1.
X = 5
Y = 6
yes
| ?- █
```

The window also features a menu bar with options: Сеанс, Правка, Вид, Закладки, Настройка, Справка. At the bottom, there is a taskbar with a "Shell" icon and a system tray icon.



Пока наша возможность задавать вопросы ограничена. Чтобы задавать более интересные вопросы, надо загрузить какую-нибудь программу. Это можно процедурой `consult`, для которой придумано сокращение. Например, если программа находится в файле `'my_program.pl'`, то ее можно загрузить, введя

```
?- consult(my_program) .
```

или

```
?- [my_program] .
```

А команда

```
?- [my_program, another_program, yet_another_program] .
```

загрузит сразу все указанные программы. Но прежде чем загружать программу надо ее написать.

## Программы.

Программы на Прологе часто называют "базами данных" или даже "базами знаний" в том смысле, что они представляет собой совокупности предложений, определяющих отношения между объектами предметной области или свойства этих объектов. Свойства и отношения в Прологе, как и в логике, называют предикатами.

Каждый предикат определяется последовательностью *предложений*, которую называют *процедурой*. Традиционно выделяют два типа предложений: *факты* и *правила*. Для начала разберемся с первой разновидностью. Предложение-факт состоит из имени предиката, за которым в круглых скобках следуют аргументы. Такое предложение описывает утверждение о конкретных объектах. Программа, состоящая только из фактов в сущности является реляционной базой данных.

Возьмем в качестве примера фрагмент базы данных небольшой компании NoLycosm&Shity Ltd. Одноместный предикат `employee` устанавливает свойство "быть работником".

```
employee (anthony) .  
employee (barbara) .  
employee (charles) .  
employee (diana) .  
employee (edward) .  
employee (frederic) .  
employee (gregory) .  
employee (herbert) .  
employee (isabella) .  
employee (john) .
```

Вообще то, имена собственные положено писать с прописной буквы, но в Прологе имена отношений и объектов должны начинаться со строчной буквы или заключаться в кавычки. Как правило, предпочитают обходиться без кавычек. Программа может содержать предикаты с одинаковыми именами, но с разным количеством аргументов. Чтобы различать их употребляют имена состоящие из имени предиката и его аргументности. Полное имя только что определенного предиката: `employee/1`.

Предикат `salary/2` устанавливает оклад каждого работника.

```
salary (anthony, 500) .  
salary (barbara, 200) .  
salary (charles, 180) .  
salary (diana, 150) .  
salary (edward, 150) .  
salary (frederic, 140) .  
salary (gregory, 150) .  
salary (herbert, 100) .  
salary (isabella, 90) .  
salary (john, 160) .
```

Предикат `boss/2` устанавливает описывает организационную структуру компании. Утверждение `boss(A,B)` означает, что B - руководитель A.

```
boss(barbara, anthony).  
boss(charles, anthony).  
boss(diana, barbara).  
boss(edward, barbara).  
boss(frederic, charles).  
boss(gregory, charles).  
boss(herbert, charles).  
boss(isabella, gregory).  
boss(john, gregory).
```

Загрузив программу, можно задавать вопросы, непосредственно к ней относящиеся.

```
?- [hh].
```

```
% hh compiled 0.00 sec, 0 bytes
```

```
Yes
```

```
?- employee(anthony).
```

```
Yes
```

```
?- employee(ronald).
```

```
No
```

```
?- boss(barbara, anthony).
```

```
Yes
```

```
ighor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка

| ?- consult(my).
compiling /home/ighor/my.pl for byte code...
/home/ighor/my.pl compiled, 28 lines read - 279
1 bytes written, 10 ms

(1 ms) yes
| ?- employee(anthony).

(1 ms) yes
| ?- employee(ronald).

no
| ?- boss(barbara, anthony).

yes
| ?- █
```

Это простейший тип вопросов. Система просто проверяет записано ли некоторое предложение в программе и отвечает "да" если это так или "нет" в противном случае. Таким образом, предложение считается ложным, если явно не записано обратного.

Более интересные вопросы получаются, если подставить в запросы переменные. Например, мы можем узнать зарплату Чарльза, спросив.

```
?- salary(charles, S) .
```

```
S = 180
```

```
Yes
```



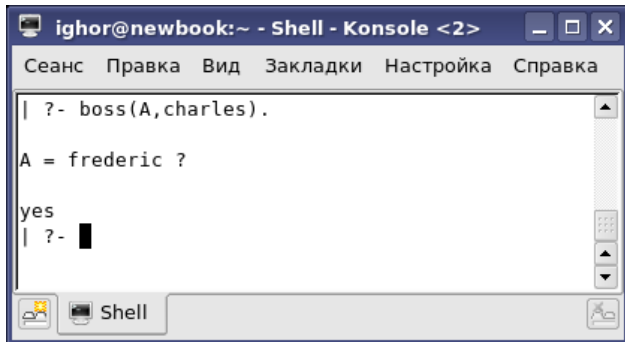
```
ighor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- salary(charles,S).
S = 180
yes
| ?- █
```

Отвечая на этот вопрос Пролог сопоставляет его со всеми фактами из базы данных, пытаясь подобрать подходящее значение переменной. Некоторые вопросы допускают несколько вариантов ответа. Такие вопросы, называют *недетерминированными*.

```
?- boss(A,charles) .
```

```
A = frederic
```

```
Yes
```



The screenshot shows a terminal window titled "igor@newbook:~ - Shell - Konsole <2>". The window contains the following text:

```
Сеанс  Правка  Вид  Закладки  Настройка  Справка  
| ?- boss(A,charles).  
  
A = frederic ?  
  
yes  
| ?- █
```

The window also features a menu bar with options: Сеанс, Правка, Вид, Закладки, Настройка, Справка. At the bottom, there is a taskbar with a "Shell" icon.

Но это не единственный ответ. Отвечая на вопросы с переменными система, выдав значение ждет нажатия на Enter. На самом деле, она интересуется нужны ли нам другие ответы. Нажимая на Enter мы говорим "спасибо, достаточно" и система отвечает Yes. Чтобы получить другие ответы надо ввести ";".

```
?- boss(A, charles) .
```

```
A = frederic ;
```

```
A = gregory ;
```

```
A = herbert ;
```

```
No
```

```
igor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- boss(A,charles).
A = frederic ? ;
A = gregory ? ;
A = herbert ? ;
no
| ?- █
```

Последнее *No* означает, что других ответов нет. Ответы выдаются в том же порядке, в котором они встречаются в программе. Вопрос с двумя переменными вернет все пары подходящие значений.

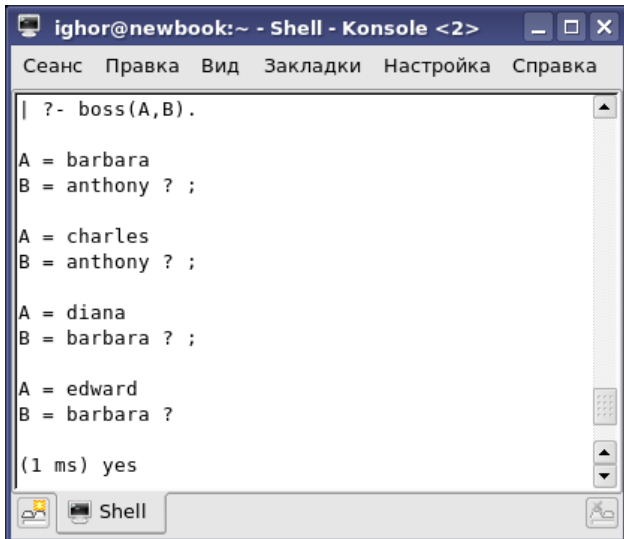
```
?- boss (A,B) .
```

```
A = barbara  
B = anthony ;
```

```
A = charles  
B = anthony ;
```

```
A = diana  
B = barbara
```

```
Yes
```



```
igor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- boss(A,B) .
A = barbara
B = anthony ? ;
A = charles
B = anthony ? ;
A = diana
B = barbara ? ;
A = edward
B = barbara ?
(1 ms) yes
```

В в тех случаях, когда требуется указать переменную но её значение для нас несущественно можно использовать так называемую *анонимную переменную*, состоящую из единственного символа подчеркивания. Отвечая на вопрос, содержащий анонимную переменную, Пролог подбирает для нее значение, но не выводит его.

```
?- boss(A,_) .
```

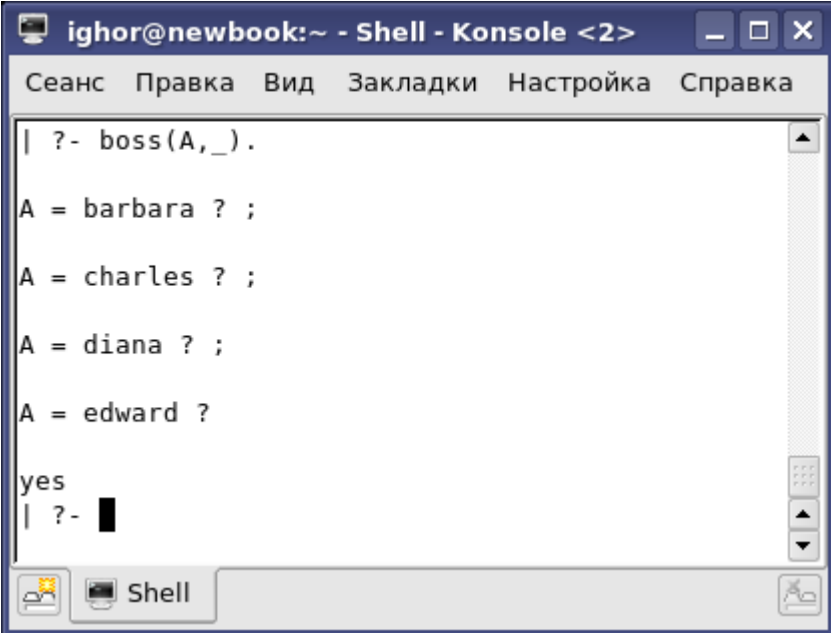
```
A = barbara ;
```

```
A = charles ;
```

```
A = diana ;
```

```
A = edward
```

```
Yes
```



```
ighor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- boss(A,_).
A = barbara ? ;
A = charles ? ;
A = diana ? ;
A = edward ?
yes
| ?- █
```



До сих пор мы задавали довольно простые вопросы. Чтобы ответить на них достаточно было просмотреть одно отношение. Но как мы знаем, простые вопросы можно объединять в более сложные. Запятая при этом получает логический смысл конъюнкции предложений. Например, спросим не получает ли кто то больше своего начальника

?- boss(A,B), salary(A,SA), salary(B,SB), SA > SB.

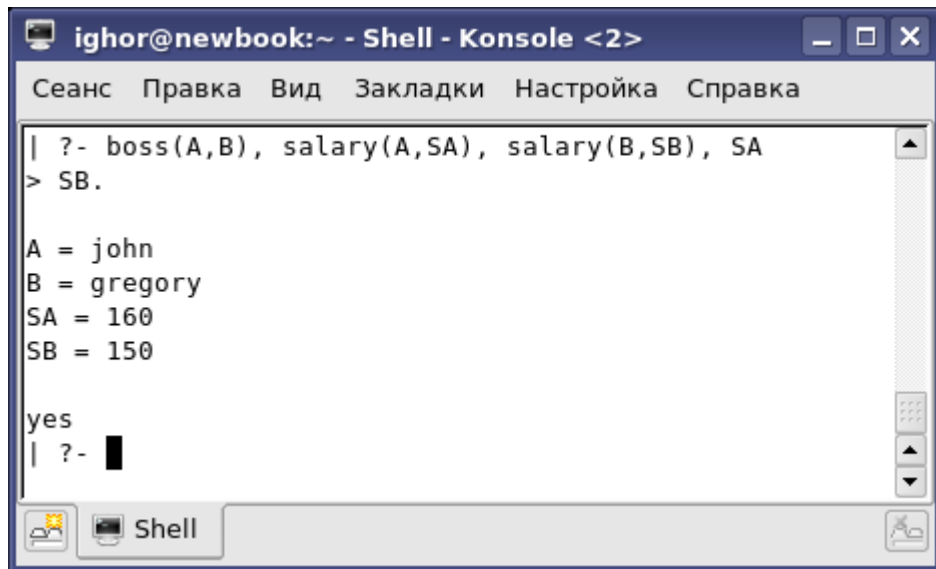
A = john

B = gregory

SA = 160

SB = 150 ;

No



The image shows a terminal window titled "ighor@newbook:~ - Shell - Konsole <2>". The window has a menu bar with "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal content is as follows:

```
| ?- boss(A,B), salary(A,SA), salary(B,SB), SA  
> SB.  
  
A = john  
B = gregory  
SA = 160  
SB = 150  
  
yes  
| ?- █
```

The terminal window includes standard window controls (minimize, maximize, close) and a taskbar at the bottom with a "Shell" icon.

и обнаружим не порядок: оклад Джона больше, чем у Грегори.

Кроме конъюнкции для соединения предложений можно применять и дизъюнкцию, которая обозначается точкой с запятой ";".

?- salary(X,S), (S<100;S>200) .

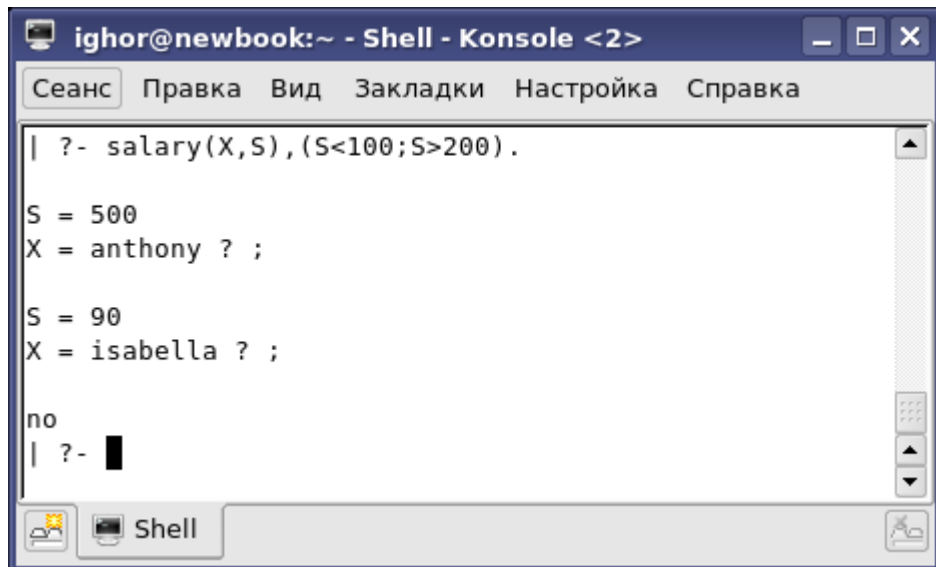
*X = anthony*

*S = 500 ;*

*X = isabella*

*S = 90 ;*

*No*



The image shows a terminal window with a dark blue title bar containing the text "ighor@newbook:~ - Shell - Konsole <2>". Below the title bar is a menu bar with the items "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The main area of the terminal displays the following text:

```
| ?- salary(X,S),(S<100;S>200).  
  
S = 500  
X = anthony ? ;  
  
S = 90  
X = isabella ? ;  
  
no  
| ?- █
```

At the bottom of the terminal window, there is a taskbar with a "Shell" icon and a "Shell" label. On the right side of the terminal window, there are standard window controls: a scroll bar, a numeric keypad icon, and up/down arrow icons.

Отрицание обозначается "\+" (или `not`, но эта форма, хотя и привычная, не рекомендуется).

```
?- \+ employee(X).
```

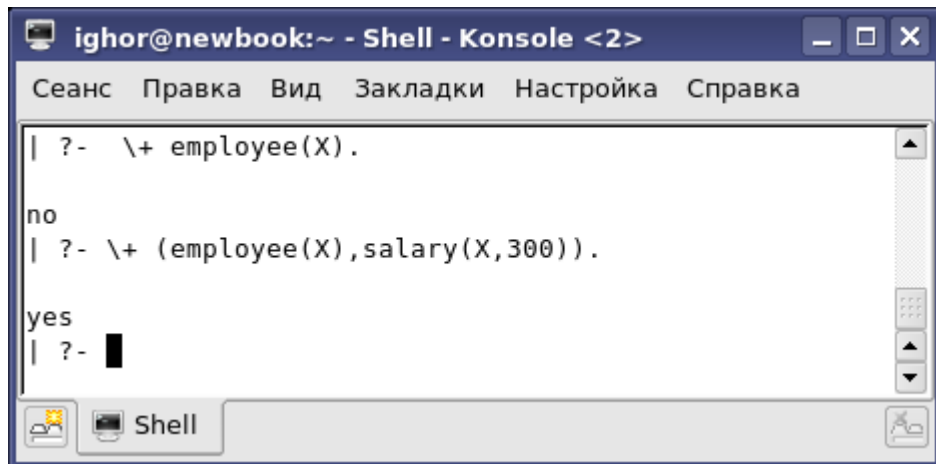
*No*

Запрос означает "не существует ни одного работника", что, конечно, ложно.

```
?- \+ (employee(X), salary(X, 300)).
```

```
X = _G157
```

*Yes*



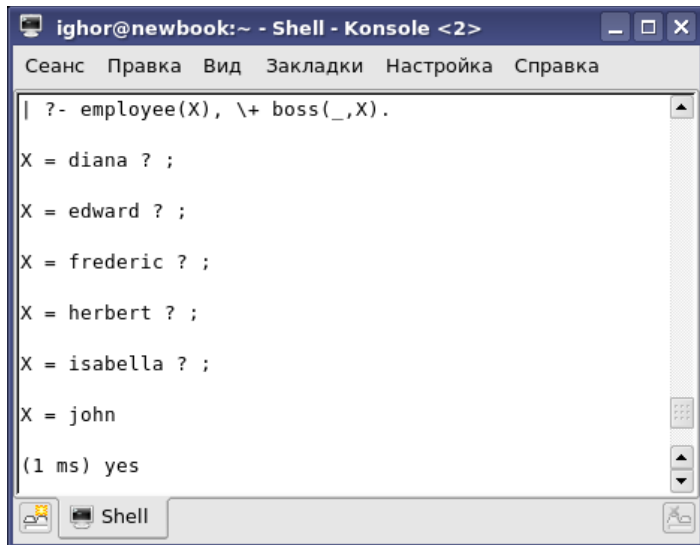
```
ighor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- \+ employee(X).
no
| ?- \+ (employee(X),salary(X,300)).
yes
| ?- █
```

А этот запрос "не существует ни одного работника с окладом 300".  
Таких работников, действительно, нет. Переменная X осталась свободной (не получила никакого значения), поэтому в качестве значения выводится нечто маловразумительное.

Как видно, отрицание позволяет формулировать общие (общеотрицательные в терминологии классической логики) суждения.  
Выясним кто не является начальником и кто не имеет начальников.

```
?- employee(X), \+ boss(_,X).
```

```
X = diana ;  
X = frederic ;  
X = herbert ;  
X = isabella ;  
X = john ;  
No
```



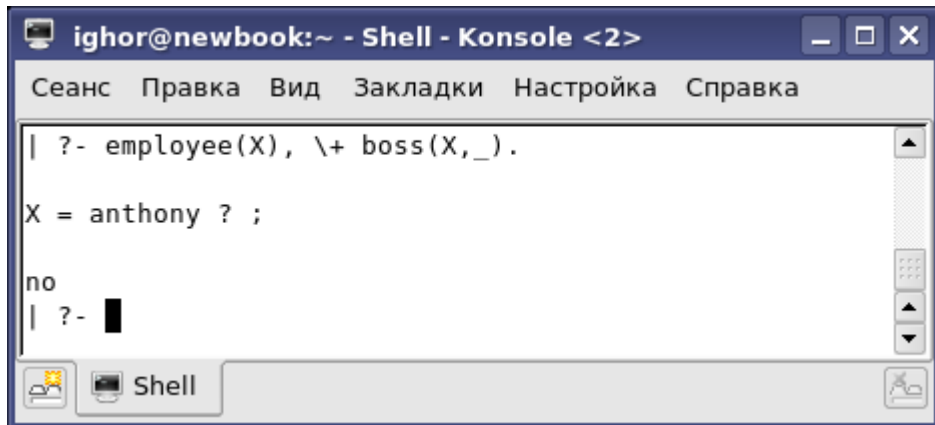
```
ighor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка
| ?- employee(X), \+ boss(_,X).
X = diana ? ;
X = edward ? ;
X = frederic ? ;
X = herbert ? ;
X = isabella ? ;
X = john
(1 ms) yes
```



?- employee(X), \+ boss(X,\_).

X = anthony ;

No



The image shows a terminal window titled "ighor@newbook:~ - Shell - Konsole <2>". The window has a menu bar with "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal content is as follows:

```
| ?- employee(X), \+ boss(X,_).  
  
X = anthony ? ;  
  
no  
| ?- █
```

The terminal interface includes a scroll bar on the right and a taskbar at the bottom with a "Shell" icon.

## Найдем самого высокооплачиваемого работника.

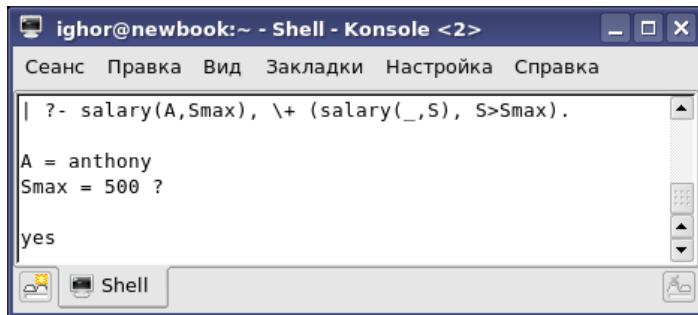
```
?- salary(A,Smax), \+ (salary(_,S), S>Smax).
```

```
A = anthony
```

```
Smax = 500
```

```
S = _G161
```

Yes



The screenshot shows a terminal window titled "igor@newbook:~ - Shell - Konsole <2>". The window contains the following text:

```
Сеанс  Правка  Вид  Закладки  Настройка  Справка  
| ?- salary(A,Smax), \+ (salary(_,S), S>Smax).  
  
A = anthony  
Smax = 500 ?  
  
yes
```

The window also features a menu bar with options: Сеанс, Правка, Вид, Закладки, Настройка, Справка. At the bottom, there is a "Shell" tab and a keyboard icon.

Интересное применение отрицания - конструкция  $(\neg(Q, \neg P))$ , которая проверяет, что все решения Q будут решениями для P. Например

```
?- \+ (employee(X), \+ salary(X,_)).
```

```
X = _G157
```

```
Yes
```

Это буквально означает "не существует работника, не имеющего оклада" или "все работники имеют оклады". В SWI-Prolog есть встроенный предикат `forall`, имеющий тот же смысл.

The screenshot shows a terminal window with a dark blue title bar containing the text "ighor@newbook:~ - Shell - Konsole <2>". Below the title bar is a menu bar with the items "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The main text area contains the following Prolog code and output:

```
| ?- \+ (employee(X), \+ salary(X,_)).  
yes  
| ?- forall(employee(X), salary(X,_)).  
uncaught exception: error(existence_error(procedure, fo  
rall/2),top_level/0)  
| ?- █
```

At the bottom of the window, there is a taskbar with a "Shell" tab and a small icon on the right.

```
?- forall(employee(X), salary(X,_)).
```

```
X = _G157
```

```
Yes
```

```
ighor@newbook:~ - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка

[ighor@newbook ~]$ pl
Welcome to SWI-Prolog (Multi-threaded, Version 5.6.35)
Copyright (c) 1990-2007 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [my].
% my compiled 0.01 sec, 2,824 bytes

Yes
?- forall(employee(X), salary(X,_)).

Yes
```

Кроме фактов - безусловно истинных предложений, в программы можно включать и условные предложения - *правила*, которые определяют новые предикаты в терминах уже существующих. Каждое правило состоит из двух частей, разделенных символом ":-". Левая часть правила - *заголовок* описывает предикат, а правая - *тело* определяет условия, при которых он истинен. Телом может быть любой запрос, который и выполняется при вызове предиката, определенного в заголовке. Например, можно оформить в виде правил, вопросы, которые мы задавали.

```
all_paid :- forall(employee(X), salary(X,_)).
```

```
costly(A) :- boss(A,B), salary(A,SA), salary(B,SB), SA > SB.
```

```
noboss(A) :- employee(A), \+ boss(_,A).
```

```
boss(B) :- boss(_,B).
```

Дополнив программу новыми правилами, сможем задавать новые вопросы. При этом не интересующие нас переменные оказываются спрятанными внутри правил и не выводятся.

```
?- all_paid.
```

```
Yes
```



```
igor@newbook:~ - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка

?- [my2].
Warning: (/home/igor/my2.pl:1):
    Redefined static procedure employee/1
Warning: (/home/igor/my2.pl:11):
    Redefined static procedure salary/2
Warning: (/home/igor/my2.pl:21):
    Redefined static procedure boss/2
% my2 compiled 0.00 sec, 1,840 bytes

Yes
?- all_paid.

Yes
```

?- costly(A) .

A = john ;

No

?- noboss(A) .

A = diana ;

A = frederic ;

A = herbert ;

A = isabella ;

A = john ;

No

?- costly(A).

A = john ;

No

?- noboss(A).

A = diana ;

A = edward ;

A = frederic ;

A = herbert ;

A = isabella ;

A = john ;

No



?- boss(charles).

Yes

?- boss(B).

*B = anthony ;*

*B = anthony ;*

*B = barbara ;*

*B = barbara ;*

*B = charles ;*

*B = charles ;*

*B = charles ;*

*B = gregory ;*

*B = gregory ;*

No

```
ighor@newbook:~ - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка
?- boss(charles).
Yes
?- boss(B).
B = anthony ;
B = anthony ;
B = barbara ;
B = barbara ;
B = charles ;
B = charles ;
B = charles ;
B = gregory ;
B = gregory ;
No
```

Обратите внимание, что в последнем запросе каждый ответ выдается несколько раз, ровно столько, сколько он встречается в программе. Нам еще придется столкнуться с этой особенностью - Пролог выдает ответ столько раз, сколькими способами он может его найти.

Предикаты, которые мы определили очень похожи на представления, применяемые в реляционных СУБД. Представление или вид (view) хранится в базе как запрос, но обращаться с ним можно как с таблицей. Точно так же предикат определенный правилом, ничем не отличается от предиката определенного перечислением фактов.

## Например, определив

```
employee(anthony, none, 500).  
employee(barbara, anthony, 200).  
employee(charles, anthony, 180).  
employee(diana, barbara, 150).  
employee(frederic, charles, 140).  
employee(gregory, charles, 150).  
employee(herbert, charles, 100).  
employee(isabella, gregory, 90).  
employee(john, gregory, 160).
```

## и затем

```
employee(A) :- employee(A,_,_).  
salary(A,S) :- employee(A,_,S).  
boss(A,B) :- employee(A,B,_), B \== none.
```

получим предикаты, неотличимые от тех, которые определили в начале.

Предикат вовсе не обязательно должен определяться одним правилом. Определим отношение `near`, означающее, что два сотрудника находятся рядом на служебной лестнице. Это возможно в двух случаях либо первый начальник второго, либо наоборот.

```
near(A,B) :- boss(A,B) .
```

```
near(A,B) :- boss(B,A) .
```

Конечно, это можно записать и одним предложением

```
near(A,B) :- boss(A,B) ; boss(B,A) .
```

но в более сложных случаях такая запись оказывается слишком громоздкой.



Разница между правилами и фактами довольно условна. Факты можно рассматривать как частный случай правил, а именно как правила с тождественно истинным телом. Наша база могла бы начинаться и так

```
near(A,B) :- boss(A,B).  
employee(barbara) :- true.  
...
```

До сих пор наши запросы находились в рамках реляционной алгебры. Теперь мы выйдем за эти границы. Отношение `boss` устанавливает непосредственного начальника работника. Но у этого же работника могут быть и другие начальники - начальник его начальника, начальник начальника его начальника и т.д. Определим отношение `chief`, обобщающее отношение `boss`.

```
chief(A,C) :- boss(A,C).  
chief(A,C) :- boss(A,B), chief(B,C).
```

Это рекурсивно определенное отношение позволяет находить всех начальников.

```
?- chief(john,X) .
```

```
X = gregory ;
```

```
X = charles ;
```

```
X = anthony ;
```

```
No
```

```
ighor@newbook:~ - Shell - Konsole
Сеанс  Правка  Вид  Закладки  Настройка  Справка

| ?- [my4].
compiling /home/ighor/my4.pl for byte code...
/home/ighor/my4.pl compiled, 13 lines read - 2300 bytes written, 15 ms
warning: /home/ighor/my4.pl:1: redefining procedure employee/3
        /home/ighor/my3.pl:1: previous definition
warning: /home/ighor/my4.pl:10: redefining procedure employee/1
        /home/ighor/my3.pl:10: previous definition
warning: /home/ighor/my4.pl:11: redefining procedure salary/2
        /home/ighor/my3.pl:11: previous definition
warning: /home/ighor/my4.pl:12: redefining procedure boss/2
        /home/ighor/my3.pl:12: previous definition

yes
| ?- chief(john,X).

X = gregory ? ;

X = charles ? ;

X = anthony ? ;

(1 ms) no
```

так же, как и всех подчиненных

?- chief(A, charles) .

A = *frederic* ;

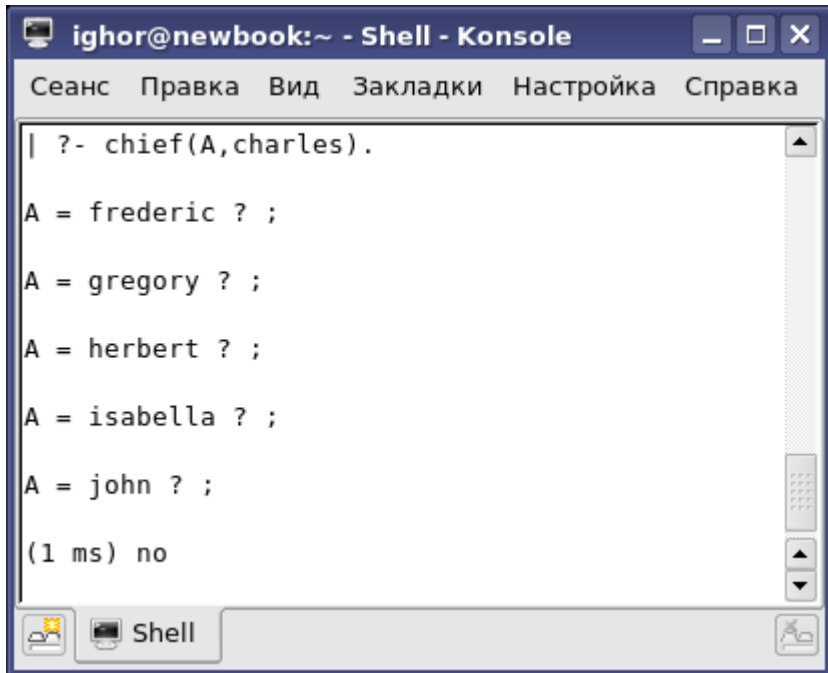
A = *gregory* ;

A = *herbert* ;

A = *isabella* ;

A = *john* ;

No



The image shows a terminal window with a dark blue title bar containing the text "ighor@newbook:~ - Shell - Konsole" and standard window control buttons. Below the title bar is a menu bar with the items "Сеанс", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The main area of the terminal is white and contains the following text:

```
| ?- chief(A,charles).  
  
A = frederic ? ;  
A = gregory ? ;  
A = herbert ? ;  
A = isabella ? ;  
A = john ? ;  
  
(1 ms) no
```

On the right side of the terminal area, there is a vertical scrollbar and a numeric keypad. At the bottom of the window, there is a taskbar with a "Shell" icon and a label "Shell".

С математической точки зрения мы построили транзитивное замыкание отношения `boss` - типичный пример задачи, которую нельзя решить средствами реляционной алгебры.

Добавим в базу данных отношение `birthday`, описывающее дату рождения. Поскольку у нас нет специального типа данных для дат, нам придется кодировать отдельно год, месяц и число.

```
birthday (anthony, 1970, 8, 12) .
```

Но Пролог позволяет нам объединить три элемента в одну структуру, которую назовем `date`.

```
birthday (anthony, date (1970, 8, 12)) .
```

В запросах можно обращаться к структуре в целом или к отдельным ее компонентам.

```
?- birthday(anthony,D).
```

```
D = date(1970, 8, 12)
```

```
Yes
```

```
?- birthday(anthony,date(Y,_,_)).
```

```
Y = 1970
```

```
Yes
```

```
?- birthday(A,date(_,12,31)).
```

```
A = frederic
```

```
Yes
```

Имя `date`, как и порядок элементов выбрано произвольно. Можно было бы использовать и другие, например записывать дату как `12/8/1970` или `1970-8-12`. Как мы уже могли убедиться, эти выражения, хотя и выглядят как арифметические, будут рассматриваться как структуры пока мы не применим к ним предикат `is`. И все же порядок год, месяц, число оказывается очень удобным. Дело в том, что предикаты типа `@<` позволяют сравнивать структуры, причем сначала выполняется сравнение первых компонент, затем если, они равны, вторых и т.д.

```
?- date(1970,8,20)@< date(1970,10,1).
```

*Yes*



```
igor@newbook:~ - Shell - Konsole <2>
Сеанс  Правка  Вид  Закладки  Настройка  Справка

[igor@newbook ~]$ gprolog
GNU Prolog 1.3.0
By Daniel Diaz
Copyright (C) 1999-2007 Daniel Diaz
| ?- [my5].
compiling /home/igor/my5.pl for byte code...
/home/igor/my5.pl compiled, 0 lines read - 385
bytes written, 8 ms

yes
| ?- birthday(anthony,D).

D = date(1970,8,12)

yes
| ?- birthday(anthony,date(Y,_,_)).

Y = 1970

yes
| ?- birthday(A,date(_,12,31)).

no
| ?- date(1970,8,20)< date(1970,10,1).

yes
```

***Задание для размышления.***

Определите отношение  $\text{younger}(A,B)$ , означающее что A моложе B, для двух способов задания birthday.

### **Задание для размышления.**

Пусть заданы три отношения

1. `haunt(Drinker, Bar)` - Drinker посещает бар Bar;
2. `serves(Bar, Beer)` - в баре Bar подают пиво сорта Beer;
3. `likes(Drinker, Beer)` - Drinker любит пиво Beer.

Определите следующие предикаты

1. `happy(Drinker)` - пиво, которое любит Drinker подают по крайней мере в одном из баров, которые он посещает.
2. `goodBar(Drinker, Bar)` - здесь подают хотя бы один сорт хорошего пива.
3. `veryGoodBar(Bar)` - все завсегдатаи этого бара найдут здесь пиво по вкусу.
4. `veryHappy(Drinker)` - в любом баре, которые посещает Drinker, подают подходящее пиво.
5. `verySad(Drinker)` - ни в одном из баров, где бывает Drinker, не подают приличного пива.