Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Лабораторная работа №4 (Язык искусственного интеллекта PROLOG)**

**Дисциплина**: Интеллектуальные системы

Выполнил студент гр. 13541/1 Смирнов М.И.

(подпись)

Руководитель Сазанов А.М.

(подпись)

“ ” 2017 г.

Санкт – Петербург

2017

1. **Демонстрационный проект «family1».**

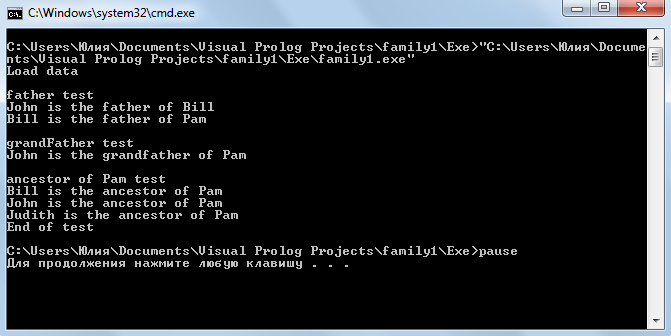
Текст программы «family1»:

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*                          Copyright (c) Sabu Francis Associates  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ implement main     open core      constants         className = "main".         classVersion = "$JustDate: 2007-09-24 $$Revision: 1 $".      clauses         classInfo(className, classVersion).      domains         gender = female(); male().      class facts - familyDB         person : (string Name, gender Gender).         parent : (string Person, string Parent).      class predicates         father : (string Person, string Father) nondeterm anyflow.     clauses         father(Person, Father) :-             parent(Person, Father),             person(Father, male()).      class predicates         grandFather : (string Person, string GrandFather) nondeterm (o,o).     clauses         grandFather(Person, GrandFather) :-             parent(Person, Parent),             father(Parent, GrandFather).      class predicates         ancestor : (string Person, string Ancestor) nondeterm (i,o).     clauses         ancestor(Person, Ancestor) :-             parent(Person, Ancestor).         ancestor(Person, Ancestor) :-             parent(Person, P1),             ancestor(P1, Ancestor).      class predicates         reconsult : (string FileName).     clauses         reconsult(FileName) :-             retractFactDB(familyDB),             file::consult(FileName, familyDB).      clauses         run():-             console::init(),             stdIO::write("Load data\n"),             reconsult("..\\fa.txt"),             stdIO::write("\nfather test\n"),             father(X, Y),                 stdIO::writef("% is the father of %\n", Y, X),             fail.         run():-             stdIO::write("\ngrandFather test\n"),             grandFather(X, Y),                 stdIO::writef("% is the grandfather of %\n", Y, X),             fail.         run():-             stdIO::write("\nancestor of Pam test\n"),             X = "Pam",             ancestor(X, Y),                 stdIO::writef("% is the ancestor of %\n", Y, X),             fail.         run():-             stdIO::write("End of test\n"). end implement main  goal     mainExe::run(main::run). |

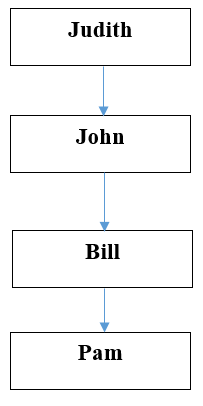
Содержимое файла «fa.txt»:

|  |
| --- |
| clauses  person("Judith",female()).  person("Bill",male()).  person("John",male()).  person("Pam",female()).  parent("John","Judith").  parent("Bill","John").  parent("Pam","Bill"). |

Результат запуска программы:



Генеалогическое дерево:



Онтология:

1. Определяем степень родства между людьми и, таким образом, построим генеалогическое дерево.
2. Входными данными будут являться люди с указанными именами и полом, а также с указанными родственными связями:

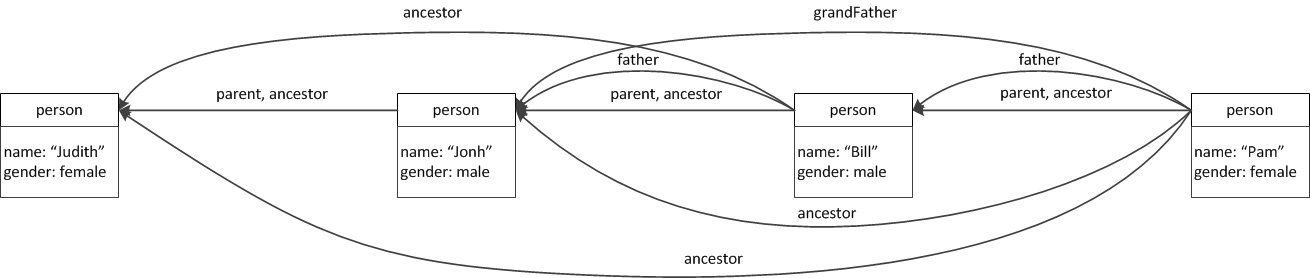
А) Определяем кто чей отец – должен быть родителем и мужчиной.

Б) Определяем кто чей дедушка – должен быть родителем родителя и мужчиной.

В) Определяем предка по восходящей линии (ancestor) для конкретного человека (для Pam).

По итогу получим: результаты отцовских тестов, тестов на дедушку, и тестов на старших родственников для указанного человека.

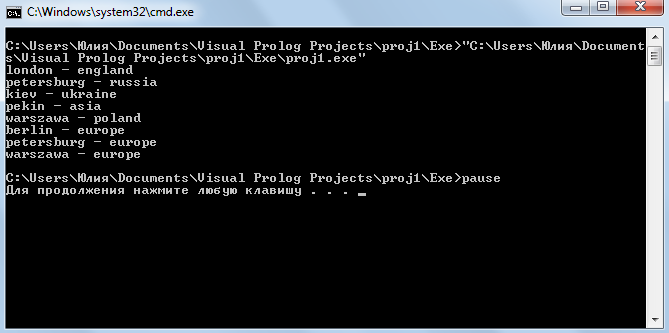
Концептуальная карта (семантическая сеть), описывающая данный пример:



1. **Примеры из пособия.**

Листинг 1:

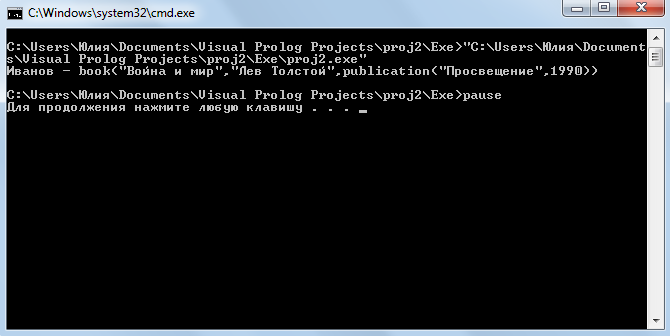
|  |
| --- |
| implement main     open core      constants         className = "main".         classVersion = "".      clauses         classInfo(className, classVersion).          class predicates         situ : (string Gorod, string Strana) nondeterm anyflow.     clauses             situ ("london", "england").             situ ("petersburg", "russia").             situ ("kiev", "ukraine").             situ ("pekin", "asia").             situ ("warszawa", "poland").             situ ("berlin", "europe").             situ (X, "europe"):-                 situ (X, "russia").             situ (X, "europe"):-                 situ (X, "poland").          clauses         run():-             console::init(),             situ(X, Y),             stdIO::writef("% - %\n", X, Y),             fail.         run().     end implement main          goal         mainExe::run(main::run). |



Данная программа является примером синтаксиса языка ПРОЛОГ. В результате выводятся все данные, имеющие отношение к предикату sity.

Листинг 2:

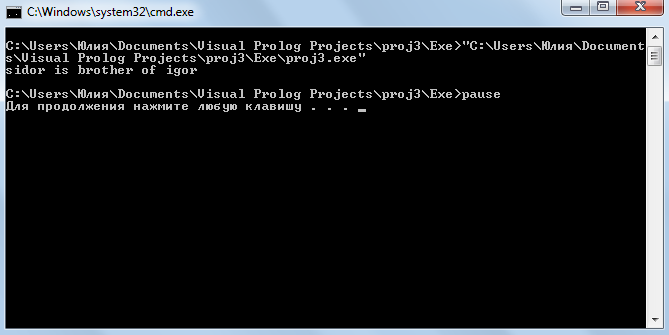
|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "". clauses     classInfo(className, classVersion).  domains     collector=symbol.     title=symbol.     author=symbol.     publisher=symbol.     year = integer.     personal\_library = book(title, author, publication).     publication = publication(publisher, year).  class predicates     collection : (collector[out], personal\_library[out]).      clauses     collection("Иванов", book("Война и мир", "Лев Толстой", publication("Просвещение",1990))).          clauses     run():-         console::init(),         collection(X, Y),         stdIO::writef("% - %\n", X, Y),         fail.     run(). end implement main  goal     mainExe::run(main::run). |



Данная программа показывает, как в ПРОЛОГе можно использовать составные объекты. Составные объекты позволяют описывать иерархические структуры, в которых описание одного предиката включает в себя описание других предикатов.

Листинг 3:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "". clauses     classInfo(className, classVersion).  domains     person = symbol. class predicates     otec : (person, person) nondeterm anyflow.     man : (person) nondeterm.     brat : (person[out], person[out]) nondeterm. clauses     man(X):-         otec(X,\_).     brat(X,Y):-         otec(Z,Y),         otec(Z,X),         man(X),         X<>Y.     otec("ivan","igor").      otec("ivan","sidor").      otec("sidor","lisa").  clauses     run():-         console::init(),         brat(X, Y),         stdIO::writef("% is brother of %\n", X, Y),         fail.     run(). end implement main  goal     mainExe::run(main::run). |

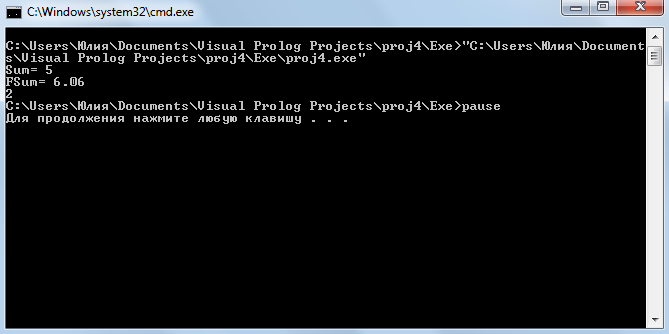


Эта программа решает примитивную задачу о семейных отношениях. Мы задаем исходные факты: 1)Иван – отец Игоря, 2) Иван – отец Сидора, 3) Сидор – отец Лизы. И ставим задачу определить, есть ли брат у Игоря.

В результате мы находим двух братьев – Сидора и Игоря.

Листинг 4:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  class predicates     add : (integer, integer).     fadd : (real, real).     maximum : (real, real, real[out]) nondeterm. clauses     add(X, Y):-         Z = X + Y,         stdIO::write("Sum= ", Z),         stdIO::nl.     fadd(X, Y):-         Z = X + Y,         stdIO::write("FSum= ", Z),         stdIO::nl.     maximum(X, X, X).     maximum(X, Y, X):-         X>Y.     maximum(X, Y, Y):-         X<Y. clauses     run():-         console::init(),         add(2, 3),         fadd(3.5, 2.56),         maximum(2, -4, Z),         stdIO::write(Z),         fail.     run(). end implement main  goal     mainExe::run(main::run). |

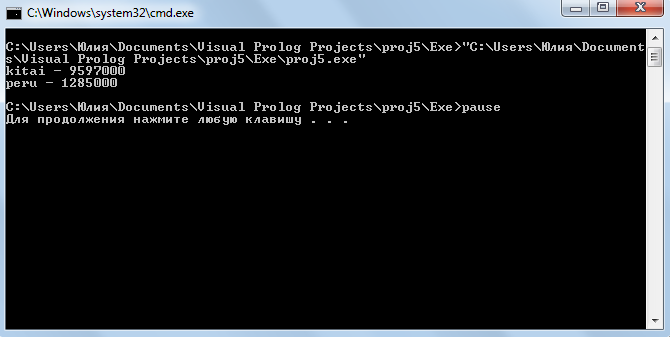


В языке ПРОЛОГ используется ряд встроенных функций для вычисления арифметических выражений. Для описания любых операций арифметики можно также использовать собственные предикаты.

В данной задаче мы находим сумму целых чисел, сумму вещественных чисел и максимум из трех вещественных чисел.

Листинг 5:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  domains     nazvanie = symbol.     stolica = symbol.     naselenie = integer.     territoria = real.  class predicates     strana : (nazvanie[out], naselenie[out], territoria[out], stolica[out]) multi.  clauses     strana("kitai", 1200, 9597000, "pekin").     strana("belgia", 10, 30000, "brussel").     strana("peru", 20, 1285000, "lima").  clauses     run():-         console::init(),         strana(X, \_, Y, \_), Y > 1000000,         stdIO::writef("% - %\n", X, Y),         fail.     run(). end implement main  goal     mainExe::run(main::run). |



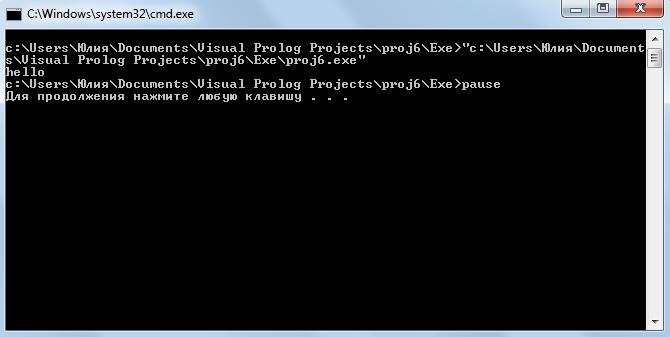
Данная программа является примером использования переменных в запросе. Мы находим страну, территория которой больше 1000000. В результате выводим название страны и её площадь.

Листинг 6:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  clauses     hello():-         console::init(),         stdIO::write("hello").          end implement main  goal     main::hello(). |

«main.cl»:

|  |
| --- |
| class main     open core  predicates     classInfo : core::classInfo.     % @short Class information  predicate.      % @detail This predicate represents information predicate of this class.     % @end predicates     hello : ().  end class main |



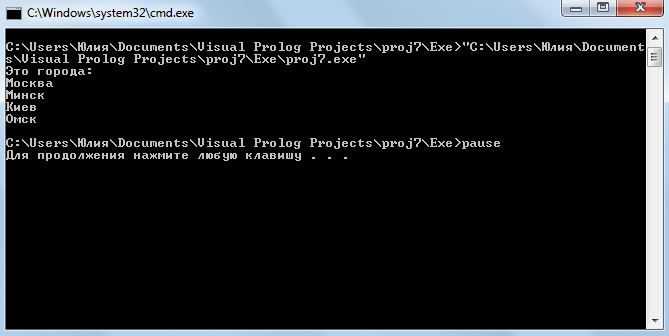
Данный пример показывает, что мы можем использовать свои предикаты в качестве стандартных.

Листинг 7:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  class predicates     gorod : (symbol[out]) multi.  clauses     gorod("Москва").     gorod("Минск").     gorod("Киев").     gorod("Омск").     show():-         gorod(X),         stdIO::write(X),         stdIO::nl(),         fail.     show(). end implement main  goal     console::init(),     stdIO::write("Это города:"),     stdIO::nl(),     main::show(). |

«main.cl»:

|  |
| --- |
| class main     open core  predicates     classInfo : core::classInfo.     % @short Class information  predicate.      % @detail This predicate represents information predicate of this class.     % @end predicates     show : ().  end class main |

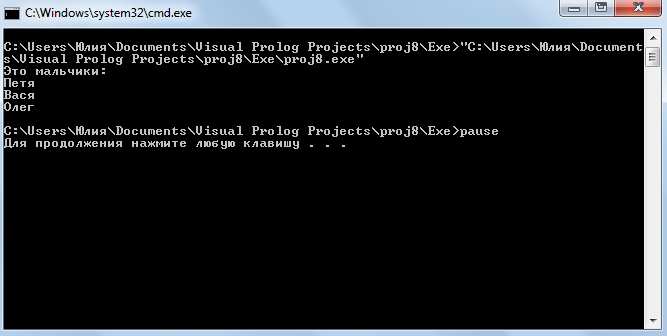
В данном примере мы используем предикат fail. Он вызывает искусственное неуспешное завершение поиска, что позволяет получить все возможные решения задачи.

Листинг 8:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).      domains     person = symbol.  class predicates     deti : (person[out]) multi.     make\_cut : (person) determ.  clauses     deti("Петя").     deti("Вася").     deti("Олег").     deti("Маша").     deti("Оля").     deti("Наташа").     show():-         deti(X),         stdIO::write(X),         stdIO::nl(),         make\_cut(X),         !.     show().     make\_cut(X):-         X = "Олег". end implement main  goal     console::init(),     stdIO::write("Это мальчики:"),     stdIO::nl(),     main::show(). |

«main.cl»:

|  |
| --- |
| class main     open core  predicates     classInfo : core::classInfo.     % @short Class information  predicate.      % @detail This predicate represents information predicate of this class.     % @end predicates     show : ().  end class main |



В этой программе мы используем предикат отсечения cut. Он позволяет получить доступ только к части данных, устраняя дальнейшие поисковые действия. На консоль выводятся только имена мальчиков, так как после имени Олег мы прекращаем поиск.

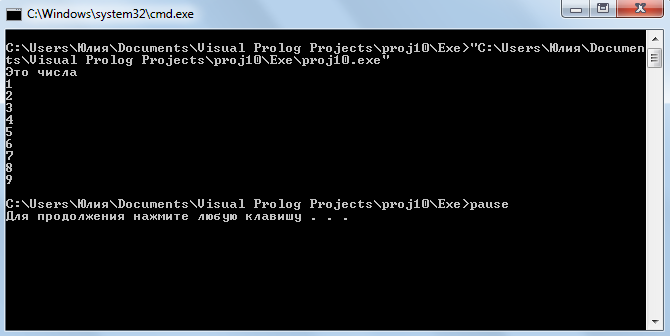
Листинг 9:

|  |
| --- |
| implement main     open core constants     className = "main".     classVersion = "". clauses     classInfo(className, classVersion).  class predicates     buy\_car : (symbol[out], symbol[out]) determ.     car : (symbol[out], symbol[out], integer[out]) multi.     color : (symbol, symbol) determ.  clauses     car("москвич", "синий", 12000).     car("жигули", "зеленый", 26000).     car("вольво", "синий", 24000).     car("волга", "синий", 20000).     car("ауди", "зеленый", 20000).     color("синий", "темный").     color("зеленый", "светлый").     buy\_car(Model, Color):-         car(Model, Color, Price),         color(Color, "светлый"),         !,         Price < 25000.     run():-         console::init(),         buy\_car(X, Y),         stdIO::writef("% - %\n", X, Y),         fail.     run(). end implement main  goal     mainExe::run(main::run). |

Мы ищем машину светлого цвета дешевле 25000. Эта программа не найдет ни одного решения, поскольку после дорогих зеленых «жигулей» поиск заканчивается, и более дешевые «ауди» не будут найдены.

Листинг 10:

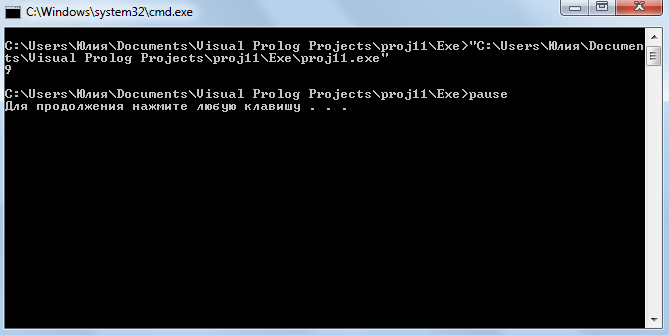
|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  domains     number = integer.      class predicates     write\_number : (number) nondeterm.      clauses     write\_number(10).     write\_number(N):-         N < 10,         stdIO::write(N),         stdIO::nl(),         write\_number(N + 1).     run():-         console::init(),         stdIO::write("Это числа"),         stdIO::nl(),         main::write\_number(1),         fail.     run(). end implement main  goal     mainExe::run(main::run). |



Данный пример отображает использование рекурсии в ПРОЛОГе. В разделе clauses даны два описания предиката write\_number. Если в процессе решения первое описание неуспешно, то используется второе описание. Программа печатает цифры от 1 до 9.

Листинг 11:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  class predicates     summa : (integer, integer[out]).  clauses     summa(X, Y):-         X < 10,         Y = X,         !.     summa(X, Y):-         X1 = X div 10,         summa(X1, Y1),         Y = Y1 + X mod 10.     run():-         console::init(),         summa(3042, Y),         stdIO::write(Y),         stdIO::nl(). end implement main  goal     mainExe::run(main::run). |

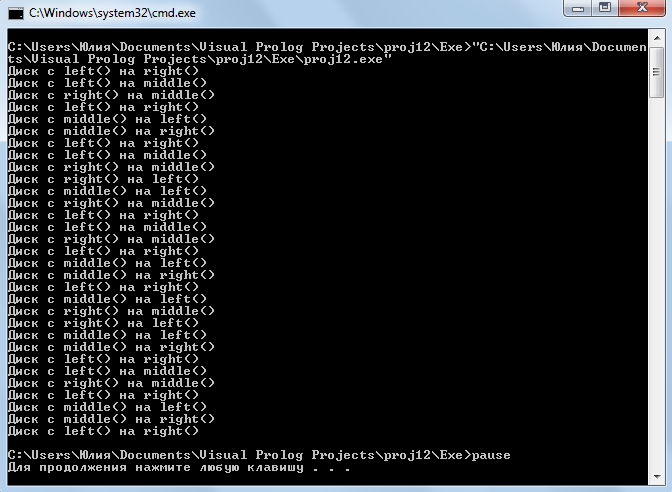


Данная программа печатает сумму всех цифр введенного числа.

Использование предиката ! в описании нерекурсивного правила позволяет избежать здесь переполнения стека.

Листинг 12:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).      domains     loc = right; middle; left.  class predicates     hanoi : (integer).     move : (integer, loc, loc, loc).     inform : (loc, loc).  clauses     hanoi(N):-         move(N, left, middle, right).     move(1, A, \_, C):-         inform(A, C),         !.     move(N, A, B, C):-         move(N - 1, A, C, B),         inform(A, C),         move(N - 1, B, A, C).     inform(Loc1, Loc2):-         stdIO::write("Диск с ", Loc1, " на ", Loc2),         stdIO::nl().     run():-         console::init(),         hanoi(5). end implement main  goal     mainExe::run(main::run). |

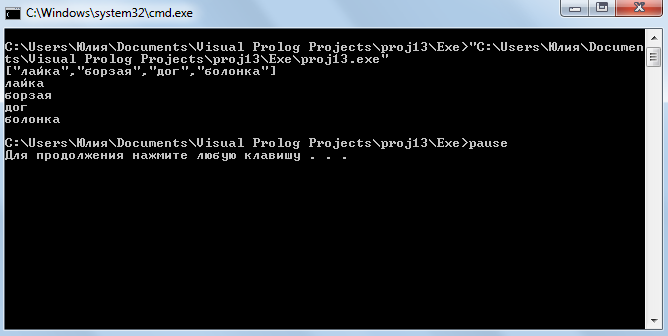


Данная программа решает задачу «Ханойская башня».

Требуется переместить диски с первого на третий стержень за некоторую последовательность ходов, каждый из которых заключается в перекладывания верхнего диска с одного из стержней на другой стержень. При этом больший диск никогда нельзя ставить на меньший диск.В результате мы получаем описание действий, необходимых для решения данной задачи.

Листинг 13:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  domains     dog\_list = symbol\*.      class predicates     dogs : (dog\_list[out]).     print\_list : (dog\_list).  clauses     dogs(["лайка", "борзая", "дог", "болонка"]).     print\_list([]).     print\_list([X|Y]):-         stdIO::write(X),         stdIO::nl(),         print\_list(Y).     run():-         console::init(),         dogs(X),         stdIO::write(X),         stdIO::nl(),         print\_list(X). end implement main  goal     mainExe::run(main::run) |



В данной программе мы используем списки. Описываем породы собак с помощью списков.

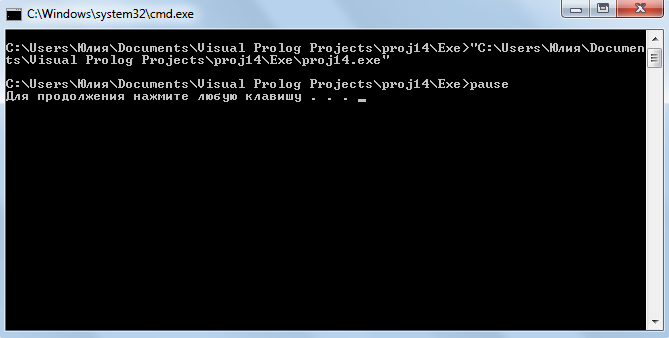
Операция разделения списка на голову и хвост обозначается с помощью вертикальной черты:

[Head|Tail]

С помощью этой операции можно реализовывать рекурсивную обработку списка.

Листинг 14:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  domains     dog\_list = symbol\*.      class predicates     find\_it : (symbol, dog\_list) nondeterm.  clauses     find\_it(X, [X|\_]).     find\_it(X, [\_|Y]):-         find\_it(X, Y).     run():-         console::init(),         find\_it("болонка", ["лайка", "дог"]),         stdIO::write("да"),         fail.     run(). end implement main  goal     mainExe::run(main::run). |



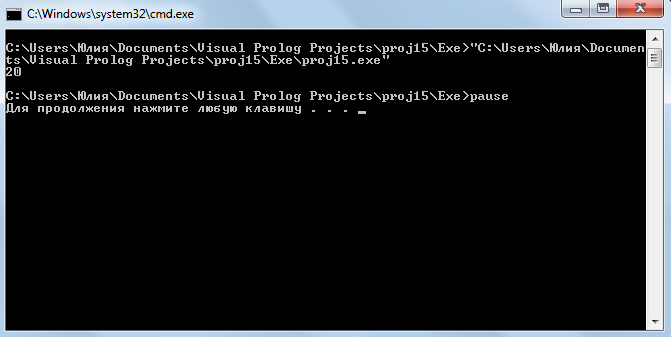
Первое правило описывает ситуацию, когда искомый элемент Х совпадает с головой списка.

Второе правило используется при неуспехе первого правила и описывает новый вызов первого правила, но уже с усеченным списком, в котором нет первого элемента и т.д. Если в списке нет элементов (пустой список), то второе правило оказывается неуспешным.

Программа не напечатает Yes, поскольку болонки нет в списке собак.

Листинг 15:

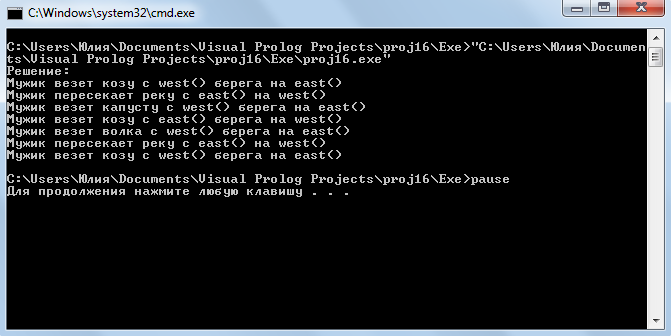
|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  domains     spisok = integer\*.      class predicates     summa\_sp : (spisok, integer[out]).  clauses     summa\_sp([], 0).     summa\_sp([H|T], S):-         summa\_sp(T, S1),         S = H + S1.     run():-         console::init(),         summa\_sp([2, 5, 3, 10], Sum),         stdIO::write(Sum),         stdIO::nl(). end implement main  goal     mainExe::run(main::run). |



Этот пример производит подсчет суммы всех элементов списка чисел.Числа, указанные в списке: 2, 5, 3, 10. Их сумма равна 20, что совпадает с результатом, выведенным на консоль.

Листинг 16:

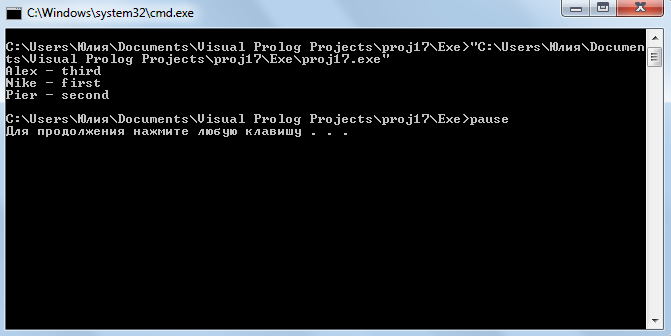
|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  domains     loc = east; west.     state = state(loc, loc, loc, loc).     path = state\*.  class predicates     go : (state, state).     path : (state, state, path, path[out]) determ.     move : (state, state[out]) nondeterm.     opposite : (loc, loc) determ anyflow.     unsafe : (state) nondeterm.     member : (state, path) nondeterm.     write\_path : (path) determ.     write\_move : (state, state) determ.  clauses     go(S, G):-         path(S, G, [S], L),         stdIO::write("Решение:"),         stdIO::nl(),         write\_path(L),         fail.     go(\_,\_).     path(S, G, L, L1):-         move(S, S1),         not(unsafe(S1)),         not(member(S1, L)),         path(S1, G, [S1|L], L1),         !.     path(G, G, T, T):-         !.     move(state(X, X, G, C), state(Y, Y, G, C)):-         opposite(X, Y).     move(state(X, W, X, C),state(Y, W, Y, C)):-         opposite(X, Y).     move(state(X, W, G, X),state(Y, W, G, Y)):-         opposite(X, Y).     move(state(X, W, G, C),state(Y, W, G, C)):-         opposite(X, Y).     opposite(east, west).     opposite(west, east):-         !.     unsafe(state(F, X, X, \_)):-         opposite(F, X).     unsafe(state(F, \_, X, X)):-         opposite(F, X).     member(X, [X|\_]).     member(X, [\_|L]):-         member(X, L).     write\_path([H1, H2|T]):-         !,         write\_move(H1, H2),         write\_path([H2|T]).     write\_path([]).     write\_move(state(X, W, G, C), state(Y, W, G, C)):-         !,         stdIO::write("Мужик пересекает реку с ", X, " на ", Y),         stdIO::nl().     write\_move(state(X, X, G, C), state(Y, Y, G, C)):-         !,         stdIO::write("Мужик везет волка с ", X, " берега на ", Y),         stdIO::nl().     write\_move(state(X, W, X, C), state(Y, W, Y, C)):-         !,         stdIO::write("Мужик везет козу с ", X, " берега на ", Y),         stdIO::nl().     write\_move(state(X, W, G, X), state(Y, W, G, Y)):-         !,         stdIO::write("Мужик везет капусту с ", X, " берега на ", Y),         stdIO::nl().   clauses     run():-         console::init(),         go(state(east, east, east, east), state(west, west, west, west)). end implement main  goal     mainExe::run(main::run). |



Данный пример показывает, как используются списки и механизм рекурсии при решении известной задачи о мужике, волке, козе и капусте.

Листинг 17:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  class predicates     name : (symbol) determ.     name : (symbol[out]) multi.     mesto : (symbol) determ.     mesto : (symbol[out]) multi.     prizer : (symbol, symbol) nondeterm.     solution : (symbol[out], symbol[out], symbol[out], symbol[out], symbol[out], symbol[out]) determ.  clauses     name("Alex").     name("Pier").     name("Nike").     mesto("first").     mesto("second").     mesto("third").     prizer(X, Y):-         name(X),         mesto(Y),         X = "Pier",         not (Y = "second"),         not (Y = "third");         name(X),         mesto(Y),         X = "Nike",         not (Y = "third");         name(X),         mesto(Y),         not (X = "Pier"),         not (X = "Nike").     solution(X1, Y1, X2, Y2, X3, Y3):-         name(X1),         name(X2),         name(X3),         mesto(Y1),         mesto(Y2),         mesto(Y3),         prizer(X1, Y1),         prizer(X2, Y2),         prizer(X2, Y3),         Y1 <> Y2,         Y2 <> Y3,         Y1 <> Y3,         X1 <> X2,         X2 <> X3,         X1 <> X3,         !.     run():-         console::init(),         solution(X1, Y1, X2, Y2, X3, Y3),         stdIO::writef("% - %\n% - %\n% - %\n", X1, Y1, X2, Y2, X3, Y3),         fail.     run(). end implement main  goal     mainExe::run(main::run). |

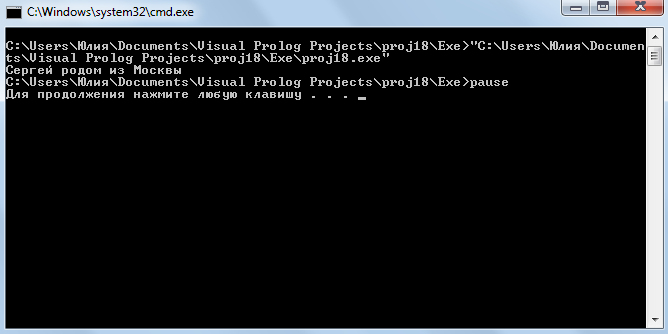


Решаем с помощью ПРОЛОГа следующую логическую задачу:

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

Листинг 18:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  domains     name = symbol.  class predicates     student : (name) determ.     student : (name[out]) multi.     gorod : (name) determ.     gorod : (name[out]) multi.     velo : (name, name) determ.     fact : (name, name) determ anyflow.     fact1 : (name, name) determ anyflow.     rodom : (name, name) nondeterm.     rodom : (name[out], name) nondeterm.     rodom\_penza : (name) nondeterm.  clauses     student(X):-         X = "Сергей";         X = "Борис";         X = "Виктор";         X = "Григорий";         X = "Леонид".     gorod(Y):-         Y = "Пенза";         Y = "Львов";         Y = "Москва";         Y = "Харьков";         Y = "Рига".     fact("Сергей", "Рига").     fact("Борис", "Пенза").     fact("Виктор", "Москва").     fact("Григорий", "Харьков").     velo(X, Y):-         student(X),         gorod(Y),         fact(X, Y),         !;         student(X),         gorod(Y),         not (fact(X, \_)),         not (fact(\_, Y)).     fact1("Борис", "Рига").     fact1("Виктор", "Львов").     rodom\_penza(X):-         student(X),         not (fact1(X, \_)),         gorod(U),         not (U = "Пенза"),         velo(X, U),         rodom("Леонид", U).     rodom(X, Z):-         student(X),         gorod(Z),         fact1(X, Z),         !;         student(X),         not (X = "Леонид"),         Z = "Пенза",         rodom\_penza(X),         !;         student(X),         gorod(Z),         not (fact1(\_, Z)),         X = "Леонид",         not (Z = "Пенза"),         student(K),         not (fact1(K, \_)),         velo(K, Z);         student(X),         not (X = "Леонид"),         gorod(Z),         not (Z = "Пенза"),         not (fact1(\_, Z)),         not (fact1(X, \_)),         gorod(Y),         not (Y = Z),         velo(X, Y),         not (rodom("Леонид", Z)),         not (rodom("Леонид", Y)).     run():-         console::init(),         rodom(X, "Москва"),         stdIO::writef("% родом из Москвы", X),         fail.     run(). end implement main  goal     mainExe::run(main::run). |



Решение ещё одной логической задачи:

«Пятеро студентов едут на велосипедах.

Их зовут Сергей, Борис, Леонид, Григорий и Виктор.

Велосипеды сделаны в пяти городах: Риге, Пензе, Львове, Харькове и Москве.

Каждый из студентов родился в одном из этих городов, но ни один из студентов не едет на велосипеде, сделанном на его родине.

Сергей едет на велосипеде, сделанном в Риге.

Борис родом из Риги, у него велосипед из Пензы.

У Виктора велосипед из Москвы.

У Григория велосипед из Харькова.

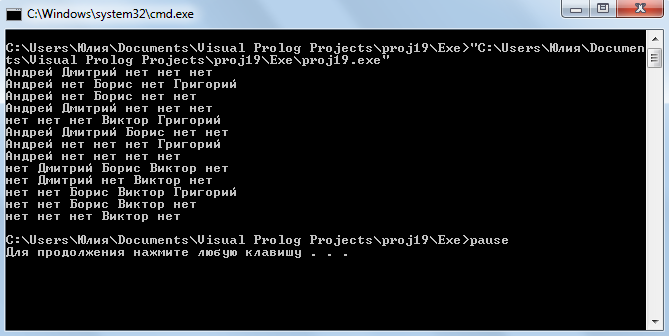
Виктор родом из Львова.

Уроженец Пензы едет на велосипеде, сделанном на родине Леонида.

Кто из студентов родом из Москвы ?»

Листинг 19:

|  |
| --- |
| implement main     open core constants     className = "main".     classVersion = "". clauses     classInfo(className, classVersion). domains     s = symbol.    class predicates     st\_A : (s[out]) multi.     st\_D : (s[out]) multi.     st\_B : (s[out]) multi.     st\_V : (s[out]) multi.     st\_G : (s[out]) multi.     ogr1 : (s, s, s, s, s) determ.     ogr2 : (s, s, s, s, s) determ.     spisok : (s[out], s[out], s[out], s[out], s[out]) nondeterm.     norm1 : (s, s, s, s, s) determ.     norm2 : (s, s, s, s, s) determ.     norm3 : (s, s, s, s, s) determ.     norm4 : (s, s, s, s, s) determ. clauses     st\_A(A):-         A = "Андрей";         A = "нет".     st\_D(D):-         D = "Дмитрий";         D = "нет".     st\_B(B):-         B = "Борис";         B = "нет".     st\_V(V):-         V = "Виктор";         V = "нет".     st\_G(G):-         G = "Григорий";         G = "нет".     ogr1("Андрей", \_, \_, "нет", \_).     ogr1("нет", \_, \_, "Виктор", \_).     ogr2(\_, "Дмитрий", \_, \_, "нет").     ogr2(\_, "нет", \_, \_, \_).     norm1("Андрей", "Дмитрий", "нет", \_, \_).     norm2("Андрей", "нет", "Борис", "нет", \_).     norm3(\_, "Дмитрий", "нет", "нет", \_).     norm4(\_, "нет", "нет", "Виктор", "Григорий").     spisok(A, D, B, V, G):-         st\_A(A),         st\_D(D),         st\_B(B),         st\_V(V),         st\_G(G),         norm1(A, D, B, V, G),         ogr1(A, D, B, V, G),         ogr2(A, D, B, V, G);         st\_A(A),         st\_D(D),         st\_B(B),         st\_V(V),         st\_G(G),         norm2(A, D, B, V, G),         ogr1(A, D, B, V, G),         ogr2(A, D, B, V, G);         st\_A(A),         st\_D(D),         st\_B(B),         st\_V(V),         st\_G(G),         norm3(A, D, B, V, G),         ogr1(A, D, B, V, G),         ogr2(A, D, B, V, G);         st\_A(A),         st\_D(D),         st\_B(B),         st\_V(V),         st\_G(G),         norm4(A, D, B, V, G),         ogr1(A, D, B, V, G),         ogr2(A, D, B, V, G);         st\_A(A),         st\_D(D),         st\_B(B),         st\_V(V),         st\_G(G),         not (norm1(A, D, B, V, G)),         not (norm2(A, D, B, V, G)),         not (norm3(A, D, B, V, G)),         not (norm4(A, D, B, V, G)),         ogr1(A, D, B, V, G),         ogr2(A, D, B, V, G). clauses     run():-         console::init(),         spisok(A, D, B, V, G),         stdIO::writef("% % % % %\n", A, D, B, V, G),         fail.     run(). end implement main goal     mainExe::run(main::run). |



Данная программа решает следующую задачу:

«Пять студентов должны посещать лекции всю неделю, но по определенным ими установленным правилам, а именно:

1. Если пришли Андрей и Дмитрий, то Бориса быть не должно, но если Дмитрий не пришел, то Борис должен быть, а Виктор быть не должен.

2. Если Виктор пришел, то Андрея быть не должно и наоборот.

3. Если Дмитрий пришел, то Григория быть не должно.

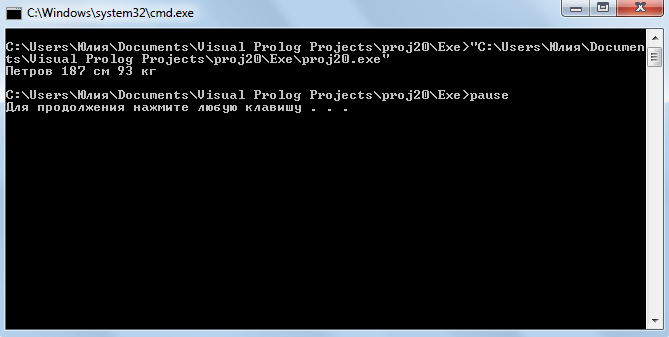
4. Если Бориса нет, то Дмитрий должен быть, но если нет также и Виктора, а если Виктор есть, Дмитрия быть не должно, но должен быть Григорий.

5. Каждый день студенты должны приходить в разных сочетаниях.

Какие это сочетания?»

Листинг 20:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  domains     name = symbol.     rost = integer.     ves = integer.  class facts     dplayer : (name, rost, ves).  class predicates     player : (name[out], rost[out], ves[out]) multi.     assert\_database : ().      clauses     player("Михайлов", 180, 87).     player("Петров", 187, 93).     player("Харламов", 177, 80).     assert\_database():-         player(N, R, V),         assertz(dplayer(N, R, V)),         fail.     assert\_database().  clauses     run():-         console::init(),         assert\_database(),         dplayer(N, R, V),         R > 180,         stdIO::writef("% % см % кг\n", N, R, V),         fail.     run(). end implement main  goal     mainExe::run(main::run). |

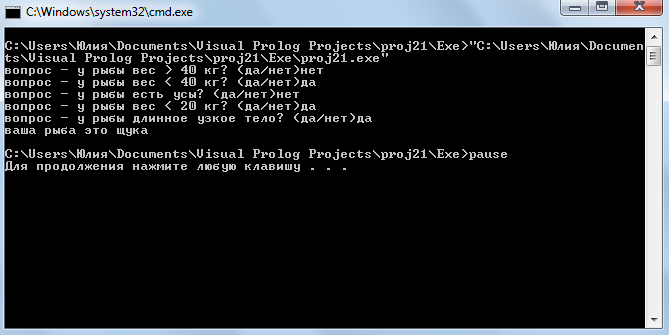


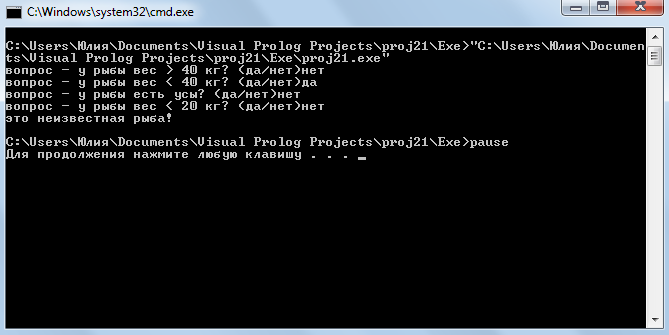
Факты, описанные в разделе clauses, можно рассматривать, как статическую базу данных (БД). Эти факты являются частью кода программы и не могут быть оперативно изменены.

В данной программе осуществляется поиск человека, ростом выше 180 см. Поэтому мы нашли Петрова.

Листинг 21:

|  |
| --- |
| implement main     open core  constants     className = "main".     classVersion = "".  clauses     classInfo(className, classVersion).  class facts     xpositive : (string, string).     xnegative : (string, string).  class predicates     expertiza : ().     vopros : (string, string) determ.     fish\_is : (string[out]) nondeterm.     positive : (string, string) determ.     negative : (string, string) determ.     remember : (string, string, string) determ.     clear\_facts : ().  clauses     expertiza():-         fish\_is(X),         !,         stdIO::write("ваша рыба это ", X, " "),         stdIO::nl,         clear\_facts().     expertiza():-         stdIO::write("это неизвестная рыба!"),         stdIO::nl,         clear\_facts().     vopros(X, Y):-         stdIO::write("вопрос – ", X, " ", Y, "? (да/нет)"),         R = stdIO::readLine(),         remember(X, Y, R).     positive(X, Y):-         xpositive(X, Y),         !.     positive(X, Y):-         not (negative(X, Y)),         !,         vopros(X, Y).     negative(X, Y):-         xnegative(X, Y),         !.     remember(X, Y, "да"):-         assertz(xpositive(X, Y)).     remember(X, Y, "нет"):-         assertz(xnegative(X, Y)),         fail.     clear\_facts():-         retract(xpositive(\_, \_)),         fail.     clear\_facts():-         retract(xnegative(\_, \_)),         fail.     clear\_facts().     fish\_is("сом"):-         positive("у рыбы", "вес > 40 кг").     fish\_is("сом"):-         positive("у рыбы", "вес < 40 кг"),         positive("у рыбы", "есть усы").     fish\_is("щука"):-         positive("у рыбы", "вес < 20 кг"),         positive("у рыбы", "длинное узкое тело").     fish\_is("окунь"):-         positive("у рыбы", "вес < 20 кг"),         positive("у рыбы", "широкое тело"),         positive("у рыбы", "темные полосы").     fish\_is("плотва"):-         positive("у рыбы", "вес < 20 кг"),         positive("у рыбы", "широкое тело"),         positive("у рыбы", "серебристая чешуя").  clauses     run():-         console::init(),         expertiza(). end implement main  goal     mainExe::run(main::run). |





Рассматриваем пример простой экспертной системы, которая решает задачу определения вида экземпляра пойманной рыбы.Программа реализует заданное дерево поиска решения. Ответы на заданные вопросы позволяют продвигаться по ветвям этого дерева к одному из вариантов решения.