*Лабораторная работа №5*

# РЕШЕНИЕ НЕТРИВИАЛЬНЫХ ЗАДАЧ СРЕДСТВАМИ ЛОГИЧЕСКОГО ПРОГРАММИРВОАНИЯ

Время: 180 мин.

## Часть 1. Что такое хороший стиль логического программирования?

*(время на исполнение – 40 мин.)*

По материалам книги: Братко Иван. Алгоритмы искусственного интеллекта на языке Prolog, 3-е издание.: Пер. с англ. - М.: Издательский дом "Вильямс", 2004. - 640 с.

Вы, наверное, уже заметили, что логические программы существенно отличаются по своей сути от программ, написанных на процедурных языках программирования. Логические программы при грамотном подходе, как правило, являются более легкими для написания, понимания и отладки по сравнению с программами, написанными на традиционных языках программирования. Естественно, этому способствуют встроенные механизмы унификации и бэктрекинга и ориентация программ на рекурсивную обработку данных.

Более всего язык логического программирования подходит для поиска решений в комбинаторных задачах, решение задач планирования, обработка баз данных, интерпретация языков, разработка экспертных систем, обработка символьной информации. В данных областях применения особенности языка `Prolog` позволяют сокращать время разработки и отладки программ.

Эффективность логической программы, понимаемая как показатель расходования ресурсов памяти и времени выполнения, в большей степени зависит от опыта написания программ, от используемого стиля программирования, от грамотного расположения отношений в тексте программы и от продуманной постановки запросов к программе.

Некоторые из способов рационального написания логических программ мы рассмотрели в ходе изучения других приемов программирования. В частности, это:

– отсечение, как повышение эффективности поиска путем предотвращения ненужного перебора с возвратами и останова процесса обработки ненужных альтернатив на самых ранних этапах;

– поиск лучшего упорядочения предложений процедур и целей в телах предложений;

– использование для представления объектов в программе более подходящих структур данных, для того, чтобы можно было реализовать операции над объектами более эффективно;

– метод кэширования, основанный использовании динамических баз данных и подразумевающий добавление в базу промежуточных результатов, которые могут понадобиться при проведении дальнейших вычислений (тогда вместо повторения вычислений можно просто выполнить выборку результатов из базы фактов).

Использование рекомендаций, приведенных ниже в данном разделе, позволит вам писать более эффективные программы.

Предложения программы должны быть короткими. Тело предложения, как правило, должно содержать лишь несколько целей.

Процедуры должны быть небольшими, поскольку длинные процедуры являются сложными для понимания. Допускается применение длинных процедур, если они имеют некоторую единообразную структуру.

Для процедур и переменных должны использоваться мнемонические имена. Имена должны подчеркивать смысл отношений и роль объектов данных.

Важное значение имеет компоновка программ. Для повышения удобства чтения следует неизменно применять определенные правила расстановки пробелов, ввода пустых строк и отступов. Между предложениями должны находиться пустые строки. Исключение составляют случаи, когда предложения представляют собой многочисленные факты, касающиеся одного и того же отношения или предложения относятся к одной и той же процедуре. Каждая цель в предложении может быть помещена на отдельной строке.

Применяемые стилистические соглашения могут зависеть от конкретной программы, поскольку их выбор диктуется рассматриваемой задачей и личным вкусом. Но важно то, чтобы одни и те же соглашения неизменно использовались во всей программе.

Оператор отсечения должен использоваться с осторожностью. Эти операторы не следует применять в тех случаях, если без них можно легко обойтись. По возможности, лучше использовать зеленые операторы отсечения, а не красные операторы отсечения. Оператор отсечения называется зеленым, если после его удаления декларативное значения предложения не изменяется. Использование красных операторов отсечения должно быть ограничено такими четко определенными конструкциями, как **not** или выбор между несколькими взаимоисключающими вариантами. Ниже приведен пример конструкции последнего типа.

если **Condition,** то **Goal1,** иначе **Goal2**

Ее можно перевести на язык **Prolog** с помощью операторов отсечения следующим образом:

```
Condition, !,     % условие Condition является истинным?
Goal1;            % если да, то Goal1,
Goal2             % иначе Goal2
```

Процедура **not** также может стать причиной выполнения программой действий, неожиданных для пользователя, поскольку она тесно связана с оператором отсечения. Тем не менее, если нужно сделать выбор между процедурой **not** и оператором отсечения, то применение первой часто бывает более оправданным, чем создание каких-то непонятных конструкций с оператором отсечения.

Использование динамических баз данных иногда бывает очень удобным и значительно упрощает программирование, однако, нередко использование предикатов **assert** и **retract** способно затруднить понимание поведения программы. В частности, при использовании этих предикатов может оказаться, что одна и та же программа в разное время отвечает на одни и те же вопросы по-разному. Если в подобных случаях необходимо воспроизвести такое же поведение, как и прежде, то следует обеспечить полное восстановление предыдущего состояния программы, которая была модифицирована в результате внесения и извлечения предложений из базы данных с применением этих предикатов.

В результате использования точки с запятой смысл предложения может стать менее очевидным; иногда удобство программ для чтения может быть повышено путем разделения предложения, содержащего точку с запятой, на несколько предложений. Но возможно также, что это улучшение будет достигнуто и за счет увеличения длины программы или даже за счет снижения ее эффективности.

В качестве иллюстрации к нескольким рекомендациям, изложенным в данном разделе, рассмотрим отношение, предназначенное для слияния двух упорядоченных списков, при условии сохранения упорядоченности в итоговом списке:

```
merge(List1,List2,List3)
```

где **List1** и **List2** – упорядоченные списки, которые сливаются в список **List3**, например:

```
merge([2,4,7],[1,3,4,8],[1,2,3,4,4,7,8])
```

Ниже приведен пример реализации процедуры **merge**, выполненный с соблюдением некоторых требований к оформлению, но в плохом стиле с точки зрения использования знаков «;».

```
DOMAINS
     i=integer*
PREDICATES
     merge(i,i,i)
CLAUSES
merge(List1,List2,List3):-

     List1=[],!,List3=List2;  % если первый список пуст

     List2=[],!,List3=List1;  % если второй список пуст

     List1=[X|Rest1],List2=[Y|Rest2],
     X<Y,!,Z=X,                % Z=X – голова списка List3
     merge(Rest1,List2,Rest3),
     List3=[Z|Rest3];

     List1=[X|Rest1],List2=[Y|Rest2],
     Z=Y,                      % Z=Y – голова списка List3
     merge(List1,Rest2,Rest3),
     List3=[Z|Rest3].
```

А ниже представлена грамотно написанная версия этой процедуры.

```
merge([],List,List):-!.
merge(List,[],List).
merge([X|Rest1],[Y|Rest2],[X|Rest3]):-
                    X<Y,!,merge(Rest1,[Y|Rest2],Rest3).
merge(List1,[Y|Rest2],[Y|Rest3]):-
                    merge(List1,Rest2,Rest3).
```

Не смотря на то, что процедура расписана аж в четыре предложения, но при чтении такой процедуры её понимание дается легче. Существенно проще устраняются ошибки при написании программ в таком стиле.

| номер предложения | смысл предложения |
|---|---|
| 1 | если первый список пуст, то в третий список передаём второй и отсекаем лишние решения |
| 2 | если второй список пуст, то в третий список передаём первый |
| 3 | если голова первого списка меньше, чем второго, то именно она присоединяется к третьему списку (но на обратном ходе рекурсии), а голова второго передается дальше по рекурсии |
| 4 | иначе голова второго списка присоединяется к третьему (но на обратном ходе рекурсии), а голова первого передается дальше по рекурсии |

Хороший стиль также подразумевает наличие комментариев в тексте программы.

Комментарии должны, прежде всего, пояснять, для чего предназначена программа и как ее использовать, и только после этого представлять подробные сведения об используемом методе решения и других нюансах программирования. Основное назначение комментариев состоит в том, чтобы предоставить пользователю возможность эксплуатировать программу, понять ее и, возможно, модифицировать. Комментарии должны описывать в наиболее краткой из возможных форм всё, что для этого необходимо.

Широко распространенной ошибкой является недостаточное применение комментариев, но в программе может также оказаться и слишком много комментариев. Изложение сведений, очевидных и без комментариев при изучении кода самой программы, создает лишь ненужную нагрузку для тех, кто изучает эту программу.

Продолжительные комментарии должны предшествовать коду, к которому они относятся, а короткие комментарии следует чередовать с самим кодом.

Обязательным для комментирования являются:

– назначение программы, способы её использования (например, какая цель должна быть вызвана и каковы ожидаемые результаты), а также примеры использования;

– назначение предикатов в программе;

– параметры этих предикатов;

– обозначения входных и выходных параметров (напомню, что входными называются такие параметры, которые при вызове предиката имеют значения);

– основные ограничения программы.

Иногда возникает необходимость делать ссылки на предикаты. При этом имя предиката следует указывать вместе с его арностью. Например, ссылка на предикат **`merge(List1,List2,List3)`** может быть представлена как **`merge/3`**. Арность следует указывать, так как в одной программе можно использовать одно и тоже имя для обозначения разных отношений, если их арность различается. Например, предикат **`m(List1,List2)`** может задавать, что второй список содержит только четные числа из первого, а предикат **`m(List1,List2,List3)`** может задавать отношение, определяющее, что элементы первого и второго списка содержатся в третьем. Имя предиката одно и то же, но Пролог их распознает по количеству аргументов.

Типы входных/выходных параметров обозначаются путем указания перед именами параметров префикса "+" (входной) и "–" (выходной). Например, обозначение **`merge(+,+,–)`** указывает, что первые два параметра **`merge`** являются входными, а третий – выходным.

– – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –

Рассмотрим пример оформления программы поиска наибольшего общего делителя для двух чисел (*см. листинг ниже*). Есть различные способы организации процедуры поиска наибольшего общего делителя, но в данной программе мы воспользуемся самым примитивным – простым перебором на основе рекурсии. В данном случае нас интересует не столько изучение способов поиска наибольшего общего делителя, сколько порядок оформления программы.

В самом начале раздела предложений (**CLAUSES**) находятся комментарии, содержащие описание программы в такой последовательности:

1) назначение основного предиката;

2) примеры его использования;

3) входные и выходные аргументы (знак «**+**» – это входной аргумент, знак «**–**» – выходной, знак «**~**» – аргумент, меняющий своё значение по ходу рекурсии);

4) вспомогательный предикат;

5) краткий смысл его работы (знак «**->**» означает «передается», иными словами запись «**Арг4 -> Арг5**» означает, что аргументу №5 присвоили значение аргумента №4).

```
DOMAINS
     i=integer
PREDICATES
     nod(i,i,i)
     del(i,i,i,i,i)
CLAUSES
                                % nod – наибольший общий делитель двух чисел
                                % пример: nod(18,12,X), X=6
                                % пример: nod(12,180,X), X=12
                                % nod(+,+,–)
                                % del(+,+,~,~,–) – вспомогательный
                                % Арг3 – счетчик, Арг4 – текущий ответ
                                % если Арг3=N1 или N2, то Арг4 –> Арг5
nod(N1,N2,Z):-
     del(N1,N2,1,1,Z).

del(N1,N2,T,X,X):-
     T=N1,!;T=N2,!.             % если текущий делитель = N1 или N2, то X – ответ и останов
del(N1,N2,T,X,L):-
     D=T+1,                     % счетчик
     0=N1 mod D,
     0=N2 mod D,!,              % если, D делит нацело и N1 и N2,
     del(N1,N2,D,D,L).          % то это новый общий делитель – Арг4=D
del(N1,N2,T,X,L):-
     D=T+1,                     % счетчик
     del(N1,N2,D,X,L).          % иначе: проверяем следующее число D
```

В заключение замечу, что для поиска наибольшего общего делителя есть более эффективный способ, чем тот, что реализован выше:

```
CLAUSES
     nod(X,X,X):-!.
     nod(X,Y,D):-
               X<Y,!,
               Y1=Y-X,
               nod(X,Y1,D);
               nod(Y,X,D).
```

Попробуйте самостоятельно перевести смысл этой процедуры с языка Пролог на русский язык.

## Часть 2. Все совпадения случайны или наконец-то в бой.

*(время на исполнение – 140 мин.)*

Напишем на Прологе логическую игру для двух участников – компьютер и человек.

Правила игры.

Играют двое. Цель игры – угадать четырёхзначное число противника. Перед началом игры каждый из игроков загадывает четырёхзначное число. Ограничения – в четырёхзначном числе не должно быть повторяющихся цифр (например, так правильно – **9413**, так не правильно – **4304**), ноль может стоять на первой позиции (например, **0123**). Выигрывает тот, кто отгадает число противника за меньшее количество ходов, при равном количестве ходов присуждается ничья.

Что такое ход и как делать ходы? Ход это четырехзначное число из неповторяющихся цифр (ноль может стоять на первой позиции). В ответ на ваш ход противник должен обозначить угаданные у него цифры. Делается это так: противник сравнивает свое число с вашим ходом, если есть цифра которая присутствует в обоих числах, но ее положение в вашем ходе не соответствую загаданному противником, то она называется «коровой», а если цифра есть и стоит именно на позиции загаданной противником то «быком». Подсчитывается общее количество «коров» и «быков» и говорится ответ. К примеру, ответ один «бык» и две «коровы» следует понимать так: вы угадали три цифры из загаданного числа, но две из них стоят не на своем месте. Пусть ваш противник загадал

**1234**, а вы сходили

**0324**

**_ККБ**

Тогда противник, поразмыслив, должен дать вам такой ответ: один «бык» и две «коровы». После чего уже он делает свой ход, а вы даете ответ о количестве «быков» и «коров». При грамотной игре количество угаданных цифр будет возрастать при постепенном увеличении количества быков. Победа, естественно, будет засчитана при четырех «быках».

Для большего понимания разберем пример одной игры. Я сыграл с программой написанной в Excel`e на VBA, специально для сравнения процедурного и логического языков программирования. Результаты заносились в ячейки памяти (см. таблицу).

| номер хода | это моё число | | | | быков | коров | тут число компьютера | | | | быков | коров |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 9 | 7 | 2 | | | X | X | X | X | | |
| 1 | 2 | 3 | 8 | 0 | 0 | 1 | 1 | 2 | 3 | 4 | 0 | 2 |
| 2 | 3 | 6 | 9 | 5 | 0 | 1 | 5 | 6 | 7 | 8 | 1 | 0 |
| 3 | 8 | 9 | 1 | 4 | 1 | 1 | 5 | 3 | 2 | 0 | 1 | 2 |
| 4 | 8 | 5 | 4 | 7 | 0 | 1 | 5 | 0 | 4 | 3 | 2 | 1 |
| 5 | 0 | 9 | 7 | 1 | 2 | 1 | 5 | 1 | 0 | 3 | 4 | 0 |
| 6 | 1 | 9 | 7 | 2 | 4 | 0 | | | | | | |
| | эти ходы делал компьютер, пытаясь угадать число у меня | | | | тут я ему отвечал | | эти ходы делал я, пытаясь угадать число у компьютера | | | | тут он мне отвечал | |

Перед игрой я загадал число (для простоты я взял год моего рождения) – 1972. Первым ходом компьютер предложил 2380 – я ему отвечаю: 0 «быков» и 1 «корова», так как цифра 2 является «коровой», а «быков» нет.

Я, опять же для простоты собственных рассуждений сходил 1234 и получил ответ – 2 «коровы», «быков» нет. Не знаю о чем «думал» компьютер, когда делал последующие ходы, но могу лишь прокомментировать свои.

Второй мой ход – опять же исходя из простоты рассуждений – 5678. Ответ компьютера – 1 «бык» – не самый удачный для меня исход, так как появилось неопределенность относительно двух цифр: 0 или 9 должны быть в числе.

В последующих ходах я делал упор на 0 и не прогадал. С третьего хода я основывался в своих рассуждениях, что быком является 5 – тут тоже на моей стороне была удача. В третьем ходе я проверял цифры из первого хода – там было число 1234 с результатом 2 «коровы» – и я решил проверить двойку и тройку, поменяв их местами – 5320. В результате получил ответ от компьютера – 1 «бык» и 2 «коровы» – ура! я на правильном пути, уже три цифры из числа, правда, пока не ясно какие именно :(

Поразмыслив, я оставил в числе 5, 3 и 0, а двойку заменил на четверку – опять угаданы три цифры :|

Ну и пятым ходом, заменив четверку на единицу, я добился успеха!!! :)

Алгоритм угадал мое число ходом позже…

Все совпадения в тексте с условными знаками, используемыми другими авторами – совершенно случайны и не несут коммерческой выгоды :o

– – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –

Прежде чем писать логическую программу, сыграйте в эту игру с человеком, чтобы понять её суть и поразмыслить над будущим алгоритмом. Попробуйте самостоятельно определить, какие основные процедуры нужно будет заложить в логическую программу, чтобы она самостоятельно угадывала число. Проверьте свою готовность к решению нетрадиционных задач. Определите опытным путем какое максимальное число ходов до победы возможно при эффективной стратегии игры.

P.S. Если рядом не оказалось человека, достаточно развитого, чтобы сыграть с вами в эту игру, то можно спросить у преподавателя или в лаборантской кафедры ту самую программу, написанную в Excel`е на VBA, и потренироваться с ней.

– – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –

Я помогу вам написать эту программу на Прологе. Мы вместе создадим простейший дизайн, базовые процедуры и стратегию решения, но часть предикатов вы сконструируете сами.

Кроме всего, основываясь на своем опыте, вы сможете усовершенствовать, предложенный ниже алгоритм. Замечу сразу, что окончательный объем текста программы займет меньше места, чем мое объяснение правил этой игры и примеры к ним. А если исключить из текста комментарии и убрать описательные и оформительские разделы программы, оставив только процедуры, относящиеся непосредственно к сути игры, то останется не более 500 символов. Как раз как в абзаце, который вы сейчас читаете.

Последующий текст будет идти от лица компьютерной программы, её противник – человек – будет называться ИГРОК.

Прежде всего, определимся с архитектурой программы.

В разделе типов данных (DOMAINS) достаточно определить только два – i=integer и li=integer*, так как мы будем работать только с целыми числами или списками цифр.

Следующий раздел – описание предикатов (PREDICATES). Что нужно иметь в программе, чтобы играть с человеком – иными словами, какие нужны предикаты и процедуры.

Назначение основных предикатов:

1) принимать четырехзначное число от игрока ( xod_h(li) ) и преобразовывать его в список цифр ( num_li(i,li) );
2) генерировать своё четырехзначное число из неповторяющихся цифр как список цифр ( gen(li) );
3) проверять подходит ли это число для того, чтобы сделать ход ( usl(li) ):

Тут потребуется небольшое отступление.

_____ *начало отступления* _____

После некоторых размышлений о возможной формализации процесса угадывания в этой игре, я пришел к выводу, что компьютер вовсе не должен копировать логику человека. Иными словами, совсем необязательно перекладывать именно человеческий способ поиска решения в алгоритм. Для реализации угадывания по правилам этой игры можно использовать как минимум два разных подхода, кроме человеческого:

– один из них состоит в последовательном переборе всех подходящих четырехзначных чисел с неповторяющимися цифрами в диапазоне от 0123 до 9876 (этот способ реализован в программе, написанной на VBA);
– другой состоит в генерации случайного четырехзначного числа с неповторяющимися цифрами из того же диапазона (этот способ будет реализован на Прологе).

Оба способа предполагают последующую проверку сгенерированного числа на подходимость по количеству быков и коров к числам из предыдущих ходов. Для лучшего понимания рассмотрим пример для второго способа с генерацией случайного числа.

Пусть первым ходом компьютер сгенерировал 0274 (случайное четырехзначное число с неповторяющимися цифрами). На этот ход игрок дал ответ 0 быков и 2 коровы. Тогда в следующем ходу компьютер будет генерировать число до тех пор пока оно не будет подходить под все предыдущие (то есть пока только под число из первого хода – 0274). Иными словами, если число текущего хода именно и есть загаданное число, тогда и ответы предыдущих ходов к нему подходят. Так, например, если компьютер сгенерирует на втором ходу случайное число 1234, подразумевая, что оно и есть загаданное, то тогда ответ при предыдущем ходе (0274) был бы не 0 быков и 2 коровы, а 2 быка и 0 коров. Несостоявшиеся быки подчеркнуты в таблице.

| ход | X | X | X | X | быков | коров |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 7 | 4 | 0 | 2 |
| 2 | 1 | 2 | 3 | 4 | | |

Значит 1234 не подходит и нужно генерировать другое число. Пусть случайно вышло число 2465, проверка которого дает результат по ходу №1 именно – 0 быков и 2 коровы, поэтому этим числом можно ходить. Коровы подчеркнуты в таблице.

| ход | X | X | X | X | быков | коров |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 7 | 4 | 0 | 2 |
| 2 | 2 | 4 | 6 | 5 | | |

Пусть игрок дал на этот ход такой ответ – 2 быка и 2 коровы. На третьем шаге опять генерируем случайное число, пусть выпало такое – 2761 (см. табл. ниже). Необходимо его проверить на подходимость ко всем предыдущим числам. Проверка по первому ходу дает положительный результат, то есть если бы число 2761 было действительно загаданным числом, то при ходе 0274 ответ и был бы 0 быков и 2 коровы. Коровы подчеркнуты в таблице. Однако проверка второго хода дает отрицательный результат, а именно при загаданном числе 2761 во время второго хода был бы ответ 2 быка и 0 коров, что не соответствует действительности (2 быка и 2 коровы). Ячейки с быками взяты в рамочку – их действительно два, а коров нет совсем.

| ход | X | X | X | X | быков | коров |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 7 | 4 | 0 | 2 |
| 2 | 2 | 4 | 6 | 5 | 2 | 2 |
| 3 | 2 | 7 | 6 | 1 | | |

Итак, нужно генерировать другое число и оно вполне может оказаться таким – 2456. Проверка по первому ходу дает положительный результат, так как двумя коровами могли оказаться 2 и 4. Во втором ходе они уже стали быками, а еще двумя коровами оказались 6 и 5, которые и были поменяны местами на третьем ходу.

| ход | X | X | X | X | быков | коров |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 7 | 4 | 0 | 2 |
| 2 | 2 | 4 | 6 | 5 | 2 | 2 |
| 3 | 2 | 4 | 5 | 6 | | |

И так можно продолжать до тех пор, пока не будет получен ответ – 4 быка.

_____ *конец отступления* _____

4) к предикатам gen(li) и usl(li) нужен ещё обобщающий предикат, который, по сути, будет реализовывать ход компьютера xod_c(li) аналогично предиката для организации хода человека;
5) проверять является ли некоторая цифра элементом списка цифр ( mem(i,li) );
6) подсчитывать количество быков и коров ( bk(li,li,i,i) ) – первый список сравнивается со вторым и в третий аргумент заносится количество быков, а в четвертый – коров;
7) проверять условие – если у кого-то уже 4 быка, то останов ( ost(i,i) ) – первый аргумент – это количество быков после очередного хода у игрока, а второй – у компьютера;
8) нужен также предикат для осуществления принудительного повтора – repeat;
9) ну и, наконец, предикат, осуществляющий ход игры – start(li) – его задача, вести статистику игры, то есть выводить на экран последовательно ходы игрока и компьютера до достижения условия победы одного из участников. У предиката start(li) один входной аргумент – это список цифр числа, загаданного компьютером.

Сам процесс загадывания пройдет ещё в разделе цели (GOAL), там же будет оформлено основное окно, в котором и будет протекать игра (makewindow(1,112,7," БЫКИ - КОРОВЫ ",0,0,25,42)):

```
GOAL
        gen(ZC),                                          % компьютер загадал свое число
        makewindow(1,112,7," БЫКИ - КОРОВЫ ",0,0,25,42),cursor(1,1),
        write("      ЧЕЛОВЕК          ПРОЛОГ"),nl,
        write(" ваш ход   Б  К    мой ход   Б  К"),nl,
        start(ZC).                                        % начнем с загаданным числом
```

– – – – – – – – – – – – – – – – – – – – – – – – –

Кроме основного окна будут использоваться ещё два: makewindow(2,110,7," ВАШ ХОД ",17,3,5,13) – для организации диалога по принятию хода (ход – это четырёхзначное число) от человека, и makewindow(3,111,7," НУ, ЧТО? ",17,19,6,21) – для организации диалога по принятию ответа от игрока на наш (компьютерный) ход.

Четырёхзначные ходы игрока можно не запоминать, а просто выводить на экран вместе с ответами о количестве быков и коров – для удобства в столбик. Они нужны игроку, чтобы он смог поразмышлять над следующим ходом. А вот наши ходы и ответы игрока о количестве быков и коров надо куда-то записывать. Эти сведения понадобятся для обоснованной генерации последующих ходов.

Посмотрите на три последовательно сделанных копии экрана программы, чтобы лучше понять ожидаемый результат.


Запоминать свои ходы (свои, т.е. компьютера) с результатами можно в виде фактов динамической базы данных. Факты о ходе можно записывать в базу, например, в таком виде **`x_c(li,i,i)`**, где первый аргумент это список из четырёх цифр (наш ход), второй аргумент – количество быков, третий аргумент – количество коров (не забывайте, что предварительно следует объявить такой предикат в разделе DATABASE).

Например, таблица из трех ходов:

| ход | X | X | X | X | быков | коров |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 7 | 4 | 0 | 2 |
| 2 | 2 | 4 | 6 | 5 | 2 | 2 |
| 3 | 2 | 4 | 5 | 6 | 4 | 0 |

может быть представлена тремя фактами:

```
x_c([0,2,7,4],0,2)
x_c([2,4,6,5],2,2)
x_c([2,4,5,6],4,0)
```

Итак, разделы DOMAINS, PREDICATES, DATABASE и GOAL уже есть. Осталось составить только раздел предложений. Часть из предложений я дам в готовом виде, вам останется просто их переписать в свою программу. Но оставшиеся вам придется составить самостоятельно – в этих местах я оставил соответствующие комментарии. Комментарии, присутствующие в тексте программы помогут вам разобраться с логикой предикатов. Для краткости написания быки обозначены как Б, коровы как К. Ниже приведены два текста раздела предложений с комментариями (листинг 1) и без них (листинг 2).

Текст с комментариями необходим для лучшего понимания логики программы. А для того чтобы не затягивать время вы можете набрать, дополнять и редактировать текст программы без комментариев (см. ниже).

После набора текста из листинга №2 вам останется только создать четыре предиката, причем предикат перевода числа в список цифр и предикат проверки принадлежности цифры списку цифр являются простейшими. Некоторые сложности у вас могут возникнуть только при создании предикатов генерации списка из четырех неповторяющихся цифр и проверки двух списков на количество быков и коров.

```
CLAUSES                                  % вариант программы с комментариями
start(ZC):-
   repeat,                               % повторяем пока кто-нибудь не наберет 4 Б
         xod_h(S2),                      % ждем хода от человека
         bk(ZC,S2,B,K),                  % вычисляем к-во Б и К
         write("  ",S2,"  ",B,"  ",K),   % отвечаем
         xod_c(HC),                      % ход компьютера
         write("    ",HC),               % выводим на экран
                                         % далее - ответ от человека о к-ве Б и К
         makewindow(3,111,7," НУ, ЧТО? ",17,19,6,21),
         cursor(1,1),write("Сколько быков - "),readint(VB),
         cursor(2,1),write("Сколько коров - "),readint(VK),
         assert(x_c(HC,VB,VK)),          % добавим в базу - ход компа, к-во Б и К
         removewindow,                   % удалаем окно НУ, ЧТО?
         write("  ",VB,"  ",VK),nl,      % записыв. к-во Б и К
         ost(B,VB),!,readchar(_).        % кто-то набрал 4 Б ?

ost(4,4):-nl,write("  НИЧЬЯ !"),!.
ost(4,_):-nl,write("  МНЕ ПРОСТО НЕ ПОВЕЗЛО ."),!.
ost(_,4):-nl,write("  МОЯ ВЗЯЛА !").

repeat. repeat:-repeat.                  % организуем принудительное повторение

% ______ процедура ХОД КОМПЬЮТЕРА - начало
xod_c(HC):-
   repeat,                               % пока не найдется подходящее число
         gen(HC),                        % генерируем число
         not(usl(HC)),!.                 % проверяем по всем ходам
usl(HC):-
         x_c(HCR,RB,RK),                 % берем из базы очередной ход
         not(bk(HC,HCR,RB,RK)).          % сравниваем HC и HCR по к-ву Б и К
% ______ процедура ХОД КОМПЬЮТЕРА - конец

% ______ процедура ХОД ЧЕЛОВЕКА - начало
xod_h(S2):-
         makewindow(2,110,7," ВАШ ХОД ",17,3,5,13),
         cursor(1,3),readint(M),         % ждем ввода 4-х значного числа M
         num_li(M,S2),                   % переводим его в список S2
         removewindow.                   % удаляем окно
% ______ процедура ХОД ЧЕЛОВЕКА - конец

% ______ процедура перевода 4-х значн. числа в список цифр - начало
% ______ num_li(+,–)                     num_li(integer,integer*)
% ______ эту процедуру составьте самостоятельно
% ______ процедура перевода 4-х значн. числа в список цифр - конец

% ______ процедура генерации четырех неповторяющихся цифр - начало
% ______ gen(–)                          gen(integer*)
% ______ эту процедуру составьте самостоятельно
% ______ процедура генерации четырех неповторяющихся цифр - конец

% ______ процедура проверки является ли цифра элементом списка - начало
% ______ mem(+,+)                        mem(integer,integer*)
% ______ эту процедуру составьте самостоятельно
% ______ процедура проверки является ли цифра элементом списка - конец

% ______ процедура для вычисления количества Б и К - начало
% ______ bk(+,+,–,–)                     bk(integer*,integer*,integer,integer)
% ______ эту процедуру составьте самостоятельно
% ______ процедура для вычисления количества Б и К - конец
```

листинг 2 / начало

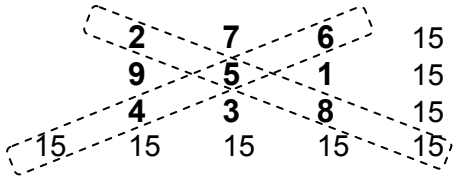
```
CLAUSES
start(ZC):-
   repeat,
          xod_h(S2),
          bk(ZC,S2,B,K),
          write("  ",S2,"  ",B,"  ",K),
          xod_c(HC),
          write("    ",HC),
          makewindow(3,111,7," НУ, ЧТО? ",17,19,6,21),
          cursor(1,1),write("Сколько быков - "),readint(VB),
          cursor(2,1),write("Сколько коров - "),readint(VK),
          assert(x_c(HC,VB,VK)),
          removewindow,
          write("  ",VB,"  ",VK),nl,
          ost(B,VB),!,readchar(_).

ost(4,4):-nl,write("  НИЧЬЯ !"),!.
ost(4,_):-nl,write("  МНЕ ПРОСТО НЕ ПОВЕЗЛО ."),!.
ost(_,4):-nl,write("  МОЯ ВЗЯЛА !").

repeat. repeat:-repeat.

xod_c(HC):-
   repeat,
          gen(HC),
          not(usl(HC)),!.
usl(HC):-
          x_c(HCR,RB,RK),
          not(bk(HC,HCR,RB,RK)).

xod_h(S2):-
          makewindow(2,110,7," ВАШ ХОД ",17,3,5,13),
          cursor(1,3),readint(M),
          num_li(M,S2),
          removewindow.
```

листинг 2 / конец

– – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – – –

## Часть 3. Задания для самостоятельного исполнения.

1. Написать программу построения магического квадрата размерностью 3x3. Магический квадрат – это квадратная таблица размерностью nxn, заполненная целыми числами от 1 до n таким образом, что сумма чисел в каждой строке, каждом столбце и на обеих диагоналях одинакова. Пример магического квадрата:




2. Написать программу в помощь кроссвордисту. Суть работы программы. Есть файл со списком русских слов (каждое слово в отдельной строке). После запуска программы пользователь получает запрос: «введите маску». Например, маска **о** означает, что из файла нужно выбрать все пятибуквенные слова, в которых на третьей позиции расположена буква «о» (знак «*» означает любой символ). Программа должна сформировать выходной файл, содержащий все слова, удовлетворяющие маске.

3. Написать программу, которая могла бы составить максимально длинное слово из списка букв. В исходном списке одна и та же буква может использоваться несколько раз, соответственно и в итоговом слове та же буква может встречаться более одного раза. Но количество таких букв в итоговом слове не должно превышать их количество в исходном списке. Так, если исходный список представлен таким образом – [`о`,`о`,`к`,`т`,`л`,`м`], то слово «молот» соответствует условиям, а слово «молоко» – нет.

В упрощенном варианте эта программа должна составлять просто слово из всех букв списка. Например, апробируйте свою программу на трех таких задачках:

| | | |
|---|---|---|
| О | Ф | И |
| С | Я | Ф |
| О | Л | И |

| | | |
|---|---|---|
| Е | Ц | С |
| П | Л | Е |
| Е | И | Н |

| | | |
|---|---|---|
| И | К | А |
| М | Н | Э |
| О | О | К |

Здесь каждая таблица содержит слово из 9 букв.