Лабораторная работа №4
ИСПОЛЬЗОВАНИЕ ВНУТРЕННИХ БАЗ ДАННЫХ ДЛЯ НАПИСАНИЯ ЛОГИЧЕСКИХ ПРОГРАММ

Время: 180 мин.
Что нужно освоить:
1) способы работы с внутренними (динамическими) базами данных: добавление фактов в базу, удаление фактов из базы;
2) как правильно сохранять базу данных на жесткий диск и как к ней обращаться при необходимости;
3) каким образом можно повысить эффективность логической программы, используя внутренние базы данных.

Отчетность по лабораторной работе:

выполняемая часть
– по части №1 – должны быть отработаны все приводимые примеры, вы должны быть готовы ответить на вопросы: в чем преимущество внутренних баз данных и какие встроенные предикаты для работы с ними используются в Прологе;
– по части №2 – должна быть оформлена программа – база данных по оценкам студентов;

Introduction

Внутренние базы данных так называются потому, что они обрабатываются исключительно в оперативной памяти компьютера, в отличие от внешних баз данных, которые могут обрабатываться на диске или в памяти. Так как внутренние базы данных размещаются в оперативной памяти компьютера, конечно, работать с ними существенно быстрее, чем с внешними. С другой стороны, емкость оперативной памяти, как правило, намного меньше, чем емкость внешней памяти. Отсюда следует, что объем внешней базы данных может быть существенно больше объема внутренней базы данных.

Однако использование внутренних баз данных при программировании на Прологе дает ряд дополнительных преимуществ.

Внутренняя база данных состоит из фактов, которые можно динамически, в процессе выполнения программы, добавлять в базу данных и удалять из нее, сохранять в файле, загружать факты из файла в базу данных. Эти факты могут использовать только предикаты, описанные в разделе описания предикатов базы данных.

```
DATABASE [ – <имя базы данных>]
<имя предиката>(<имя домена первого аргумента>,...,
< имя домена n-го аргумента>)
...
```

Пример, показывающий как можно задать внутреннюю базу данных для работы с фактами, представляющими из себя совокупность имени и фамилии студента:

```
DOMAINS
      s = string
DATABASE
      students(s, s)
```

Если раздел описания предикатов базы данных в программе только один, то он может не иметь имени. В этом случае он автоматически получает стандартное имя dbasedom. В случае наличия в программе нескольких разделов описания предикатов базы данных только один из них может быть безымянным. Все остальные должны иметь уникальное

имя, которое указывается после названия раздела DATABASE и тире. Описание предикатов базы данных совпадает с их описанием в разделе описания предикатов PREDICATES.

Обратите внимание на то, что в базе данных могут содержаться только факты, а не правила вывода, причем используемые факты не могут содержать свободных переменных. Однако, есть существенное преимущество в использовании таких баз данных. Дело в том, что факты, использующие предикаты, заданные в разделе DATABASE, могут добавляться и удаляться во время выполнения программы. Прежде чем разбираться, в чем же состоит преимущество такой динамической базы данных, следует ознакомиться с основными способами работы с фактами внутренней базы данных.

Часть 1. Встроенные предикаты для работы с базами данных.

Давайте познакомимся со встроенными предикатами Турбо Пролога, предназначенными для работы с внутренней базой данных. Все рассматриваемые далее предикаты могут использоваться в варианте с одним или двумя аргументами. Причем одноаргументный вариант используется, если внутренняя база данных не имеет имени. Если же база поименована, то нужно использовать двухаргументный предикат, в котором второй аргумент – это имя базы.

Начнем с предикатов, с помощью которых во время работы программы можно добавлять или удалять факты базы данных.

Для добавления фактов во внутреннюю базу данных может использоваться один из трех предикатов:

assert
asserta
assertz

Разница между этими предикатами заключается в том, что предикат asserta добавляет факт перед другими фактами (в начало внутренней базы данных), а предикат assertz добавляет факт после других фактов (в конец базы данных). Предикат assert добавлен для совместимости с другими версиями Пролога и работает точно так же, как и assertz. В качестве первого параметра у этих предикатов указывается добавляемый факт, в качестве второго, необязательного – имя внутренней базы данных, в которую добавляется факт. Можно сказать, что предикаты assert и assertz работают с совокупностью фактов, как с очередью, а предикат asserta – как со стеком.

Для удаления фактов из базы данных служат предикаты:

retract
retractall

Предикат retract удаляет из внутренней базы данных первый с начала факт, который может быть отождествлен с его первым параметром. Вторым необязательным параметром этого предиката является имя внутренней базы данных.

Так, например, если база данных представлена фактами:

```
students("Сидоров","Павел")
students("Саакашвили","Михаил")
students("Ладен","Усама")
```

то удаление из базы студента Саакашвили может быть представлено предикатом:

retract(students("Саакашвили",_)),

а удаление Михаила так:

retract(students(_,"Михаил")).

Для удаления всех предикатов, соответствующих его первому аргументу, служит предикат retractall. Для удаления всех фактов из некоторой внутренней базы данных следует вызвать этот предикат, указав ему в качестве первого параметра анонимную пере-

менную **retractall(_)**. Так как анонимная переменная сопоставляется с любым объектом, а предикат retractall удаляет все факты, которые могут быть отождествлены с его первым аргументом, все факты будут удалены из внутренней базы данных. Если вторым аргументом этого предиката указано имя базы данных, то факты удаляются из указанной базы данных. Если второй аргумент не указан, факты удаляются из единственной неименованной базы данных.

Предикат retractall рекурсивно удаляет все факты из базы и, по сути, может быть реализован с использованием встроенного предикат retract. Для этого следует создать рекурсивную процедуру с использованием отката при неудаче (fail) следующим образом:

```
del_all(Fact):-
                retract(Fact),
                fail.
del_all(_).
```

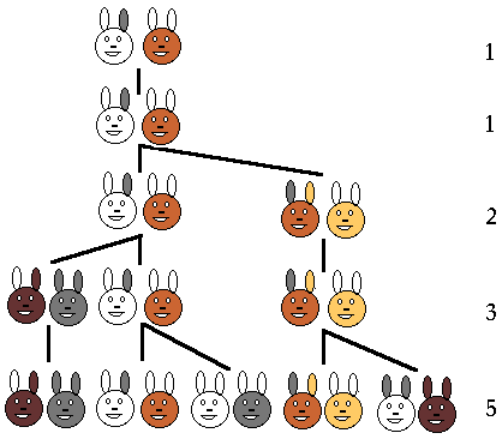
Попробуйте объяснить ***самостоятельно***, как работает данная процедура и в чем смысл второго предложения (**del_all(_).**).

Рассмотрим теперь один пример, который позволит осознать – в чем же преимущество именно ***динамической*** базы.

Эффективность программы можно поднять за счет добавления фактов в базу после их вычисления. Суть в том, что рекурсивно построенные программы зачастую производят много повторных вычислений, которых можно избежать, вычислив однажды значение и разместив его в базе как факт. Тогда, при повторной попытке унифицировать некоторую цель с аргументами уже нет необходимости производить все вычисления – Пролог просто проверит, нет ли в базе данных подходящего факта (а он там уже есть после добавления) и искомое значение просто унифицируется из факта (ранее добавленного).

Рассмотрим этот механизм на примере предиката, вычисляющего число Фиббоначи по его номеру.

Итальянский купец Леонардо из Пизы (1180-1240), более известный под прозвищем Фибоначчи, был, безусловно, самым значительным математиком средневековья. Роль его книг в развитии математики и распространении в Европе математических знаний трудно переоценить. В одном из научных трактатов Фибоначчи поместил следующую задачу: «некто поместил пару кроликов в некоем месте, огороженном со всех сторон стеной, чтобы узнать, сколько пар кроликов родится при этом в течении года, если природа кроликов такова, что через месяц пара кроликов производит на свет другую пару, а рождают кролики со второго месяца после своего рождения».



Ясно, что если считать первую пару кроликов новорожденными, то на второй месяц мы будем по прежнему иметь одну пару; на 3-й месяц - 1+1=2; на 4-й - 2+1=3 пары (ибо из двух имеющихся пар потомство дает лишь одна пара); на 5-й месяц - 3+2=5 пар (лишь 2 родившиеся на 3-й месяц пары дадут потомство на 5-й месяц); на 6-й месяц - 5+3=8 пар (ибо потомство дадут только те пары, которые родились на 4-м месяце) и т. д.

Таким образом, если обозначить число пар кроликов, имеющихся на n-м месяце через F_k , то F_1=1, F_2=1, F_3=2, F_4=3, F_5=5, F_6=8, F_7=13, F_8=21 и т.д., причем поток этих чисел регулируется общим законом:

$$F_n=F_{n-1}+F_{n-2}$$

при всех n>2, ведь число пар кроликов на n-м месяце равно числу F_{n-1} пар кроликов на предшествующем месяце плюс число вновь родившихся пар, которое совпадает с числом F_{n-2} пар кроликов, родившихся на (n-2)-ом месяце (ибо лишь эти пары кроликов дают потомство).

Числа F_n, образующие последовательность 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ... называются " числами Фибоначчи", а сама последовательность - последовательностью Фибоначчи.

Итак, в последовательности Фибоначчи начиная с 3-го каждое следующее число получается сложением двух предыдущих. Но почему эта последовательность стала столь известной?

Дело в том, что при увеличении порядковых номеров отношение двух соседних чисел последовательности асимптотически стремится к некоторому постоянному соотношению. Так, если какой-либо член последовательности Фибоначчи разделить на предшествующий ему, результатом будет величи-

на, близкая к 1.61803398875 – это отношение известно как «золотое сечение». В алгебре общепринято его обозначение греческой буквой φ (фи).

Принято считать, что объекты, содержащие в себе «золотое сечение», воспринимаются людьми как наиболее гармоничные. Пропорции пирамиды Хеопса, храмов, барельефов, предметов быта и украшений из гробницы Тутанхамона якобы свидетельствуют, что египетские мастера пользовались соотношениями золотого сечения при их создании. Начиная с Леонардо да Винчи, многие художники сознательно использовали пропорции «золотого сечения». Известно, что Сергей Эйзенштейн искусственно построил фильм Броненосец Потёмкин по правилам «золотого сечения». Он разбил ленту на пять частей. В первых трёх действие разворачивается на корабле. В двух последних – в Одессе, где разворачивается восстание. Этот переход в город происходит точно в точке золотого сечения. Да и в каждой части есть свой перелом, происходящий с учетом пропорции «золотого сечения». Эйзенштейн считал, что такие переходы воспринимаются как наиболее закономерные и естественные.

Тем не менее, существуют исследования показывающие, что значимость золотого сечения в искусстве, архитектуре и в природе преувеличена. При обсуждении оптимальных соотношений сторон прямоугольников (размеры листов бумаги A0 и кратные, размеры фотопластинок (6:9, 9:12) или кадров фотоплёнки (часто 2:3), размеры кино- и телевизионных экранов – например, 3:4 или 9:16) были испытаны самые разные варианты. Восприятие гармонии слишком индивидуально и не все принимают «золотое сечение» как оптимальное, удобное или комфортное.

Итак, последовательность чисел Фиббоначи можно задать таблицей:

порядковый номер	1	2	3	4	5	6	7	8	9	10
значение числа	1	1	2	3	5	8	13	21	34	55

или словами: первое и второе число равны единице, а каждое последующее есть сумма двух предшествующих. Формально это можно выразить так:

Число 1 = 1
Число 2 = 1
Число N = Число(N-1) + Число(N-2)

Напишем соответствующую процедуру:

```
fib(1,1):-!.                    % первое число Фиббоначи равно единице
fib(2,1):-!.                    % второе число Фиббоначи равно единице
fib(N,F) :-
        N1=N-1, fib(N1,F1),     % F1 это N-1-е число Фиббоначи
        N2=N-2, fib(N2,F2),     % F2 это N-2-е число Фиббоначи
        F=F1+F2.                % N-е число Фиббоначи равно их сумме
```

Недостаток такой организации предиката состоит в том, что при вычислении очередного N-го числа происходит многократное перевычисление предыдущих чисел Фиббоначи.

Изменим нашу программу следующим образом: добавим в нее раздел описания предикатов внутренней базы данных. В этот раздел добавим описание одного-единственного предиката **db**, который будет иметь два аргумента. Первый аргумент – это номер числа Фиббоначи, а второй аргумент – само число.

```
DATABASE
        db(integer,real)
```

Сам предикат, вычисляющий числа Фиббоначи, будет выглядеть следующим образом. Базис для первых двух чисел Фиббоначи оставим без изменений. Для шага рекурсии добавим еще одно правило. Первым делом будем проверять внутреннюю базу данных на предмет наличия в ней уже вычисленного числа. Если оно там есть, то никаких дополнительных вычислений проводить не нужно. Если же числа в базе данных не окажется, вычислим его по обычной схеме как сумму двух предыдущих чисел, после чего добавим соответствующий факт в базу данных.

Попробуем придать этим рассуждениям некоторое материальное воплощение:

```
fib(1,1):-!.                    % первое число Фиббоначи равно единице
fib(2,1):-!.                    % второе число Фиббоначи равно единице
fib(N,F):-
        db(N,F),!.              % пытаемся найти N-е число Фиббоначи в базе
fib(N,F):-                      % если в базе не нашли, то
```

```
        N1=N-1, fib(N1,F1),     % F1 это N-1-е число Фиббоначи
        N2=N-2, fib(N2,F2),     % F2 это N-2-е число Фиббоначи
        F=F1+F2,                % N-е число Фиббоначи равно их сумме
        assert(db(N,F)).        % добавляем вычисленное N-е число в базу
```

В ходе унификации предиката формируется такая база фактов:

```
fib_db(2,2)
fib_db(3,3)
fib_db(4,5)
fib_db(5,8)
fib_db(6,13)
    …
```

Испытайте оба варианта реализации предиката вычисляющего числа Фиббоначи для достаточно больших номеров и почувствуйте разницу во времени работы. Только не ставьте сразу большие значения, иначе попросту не дождётесь ответа. Попробуйте для начала вычислить число Фиббоначи с порядковым номером 30 – разница во времени вычисления будет почти незаметна. Но если вы будете постепенно наращивать значения номера числа, то вскоре вы поймете насколько могут быть полезны динамические базы данных.

Для сохранения динамической базы на диске служит предикат **save**. Он сохраняет её в текстовый файл с именем, которое было указано в качестве первого параметра предиката. Если второй необязательный параметр был опущен, происходит сохранение фактов из единственной неименованной внутренней базы данных. Если было указано имя внутренней базы данных, в файл будут сохранены факты именно этой базы данных.

Факты, сохраненные в текстовом файле на диске, могут быть загружены в оперативную память командой **consult**. Первым параметром этого предиката указывается имя текстового файла, из которого нужно загрузить факты. Если второй параметр опущен, факты будут загружены в единственную неименованную внутреннюю базу данных. Если второй параметр указан, факты будут загружены в ту внутреннюю базу данных, чье имя было помещено во второй параметр предиката. Предикат будет неуспешен, если для считываемого файла недостаточно свободного места в оперативной памяти или если указанный файл не найден на диске, или если он содержит ошибки.

Заметим, что сохраненная внутренняя база данных представляет собой обычный текстовый файл, который может быть просмотрен и/или изменен в любом текстовом редакторе. При редактировании или создании файла, который планируется применить для последующей загрузки фактов с использованием предиката **consult**, нужно учитывать, что каждый факт должен занимать отдельную строку. Количество аргументов и их тип должны соответствовать описанию предиката в разделе **database**. В файле не должно быть пустых строк, внутри фактов не должно быть пробелов, за исключением тех, которые содержатся внутри строк в двойных кавычках, других специальных символов типа конца строки, табуляции и т.д. Давайте на примере разберемся со всеми этими предикатами.

Часть 2. Создаём первую базу данных на Прологе – «журнал учебной группы».

Итак, создадим простой журнал учебной группы, в котором будут содержаться факты вида: **Фамилия Оценка**. Если бы нужно было просто перечислить факты, которые не будут со временем меняться, то можно было бы их задать в разделе предложений. Но сейчас предстоит несколько иная задача – база должна иметь возможность пополняться и изменяться. Для этого следует естественно придумать предикат и описать его в разделе DATABASE. Например так:

```
DATABASE
      ball(string, integer)
```

Предикат `ball` является двухаргументным. Предполагается, что в качестве первого аргумента будет вводиться ФАМИЛИЯ студента, а в качестве второго его ОЦЕНКА.

Кроме того, следует сразу предусмотреть, что база может быть при необходимости сохранена на жесткий диск и в последствии к ней можно будет обратиться с целью просмотра информации об оценках или их коррекции. Для этого организуем файл для хранения динамической базы данных – `student.ddb`. Чтобы к нему обратиться следует использовать предикат **`consult`**, он загрузит с диска факты из обозначенного файла.

Создайте в текстовом редакторе Пролога файл **`student.ddb`**, который содержит такие данные:

```
ball("Петров",5)
ball("Сидоров",3)
ball("Шванидзе",4)
ball("Арутюнян",2)
```

Напишите программу:

```
domains
      s=string
      i=integer
database
      ball(s,i)
goal
      consult("student.ddb"),
      ball(E,R),
      write(E," ",R).
```

Если запустить её на исполнение, то ответ будет таким:

```
Петров 5
```

И всё… Так как цель внутренняя, то Пролог найдет только первый подходящий вариант унификации цели, а для перечисления всех вариантов необходимо предпринять дополнительные шаги. К примеру, модифицировать цель таким образом:

```
goal
      consult("student.ddb"),
      ball(E,R),
      write(E," ",R),nl,fail.
```

Тогда при запуске программы получим ответ:

```
Петров 5
Сидоров 3
Шванидзе 4
Арутюнян 2
```

Давайте модифицируем цель так, чтобы была возможность добавить факт в базу и сохранить её на жесткий диск:

```
goal
      consult("student.ddb"),
      write("Введите фамилию - "),nl,readln(F),
      write("Введите  оценку - "),nl,readint(O),
      assert(ball(F,O)),
      save("student.ddb").
```

Запустите программу с такой целью и введите новую фамилию и оценку. После исполнения программы просмотрите файл **student.ddb**. Вы обнаружите, что данные, которые вы вводили, были успешно сохранены и могут использоваться в дальнейшем.

Однако, программа должна быть более универсальной и позволять пользователю самому выбирать когда и что делать – необходимо МЕНЮ.

Создадим раздел описания предикатов и добавим туда два предиката **menu** и **m(char)**. Первый будет выводить на экран меню, а второй после выбора пользователем пункта выполнять соответствующее действие. Раздел описания типов данных и описания предикатов базы данных оставим без изменений, исправим внутреннюю цель и добавим раздел предложений следующим образом:

```
CLAUSES
menu:-
    clearwindow,          % очистка текущего окна
    write("1 - Получение оценки по фамилии "),nl,
    write("2 - Добавление новой записи "),nl,
    write("0 - Выйти"),nl,
    readchar(C),          % читаем символ с клавиатуры
    m(C).                 % вызываем выполнение пункта меню
m('1'):-
    clearwindow,
    write("Введите фамилию - "), nl,
    readln(N),            % ждём ввода фамилии
    ball(N, B),           % поиск в базе данных фамилии и оценки
    write("Оценка: ",B), % если есть, выводим на экран
    readchar(_),          % ждём нажатия любой клавиши
    menu.                 % выводим на экран меню
m('2'):-
    clearwindow,
    write("Введите фамилию"),nl,
    readln(N),            % ждём ввода фамилии
    write("Введите оценку"),nl,
    readint(B),           % ждём ввода оценки
    assert(ball(N,B)),    % добавляем запись в базу данных
    menu.                 % выводим на экран меню
m('0'):-
    save("student.ddb"). % сохраняем базу данных
m(_):-
    menu.                 % если пользователь по ошибке нажал не 0,1,2,
                          % то еще раз будет отображено меню
```

Чтобы грамотно написать программу, необходимо предусмотреть две возможных ситуации:

1) если файл **student.ddb** существует, тогда загрузить его в память;

2) если файл ещё не существует, то его надо создать (то есть начать вести журнал группы).

Для описания этих исходов создадим безаргументный предикат **start** с соответствующими вариантами реализации в двух предложениях:

```
start:-
    existfile("student.ddb"),!,
    consult("student.ddb"),
    menu.
start:-
    openwrite(f,"student.ddb"),
    closefile(f),
    menu.
```

Не забываем, что процесс унификации любой цели происходит всегда последовательно сверху вниз и, если в качестве цели задать предикат **start**, то можно ожидать следующего процесса унификации.

В первом предложении проверяем существование файла **student.ddb**, если файл существует, то выполняется безусловный предикат «отсечение», который устраняет альтернативный путь – второе предложение не будет выполняться. После отсечения выполняется встроенный предикат **consult**, который загружает в оперативную память факты из базы данных. После чего управление передаётся в меню.

Если же файл **student.ddb** не существует, то первая подцель в первом предложении недостижима и первое предложение не может быть унифицировано. Это приводит к тому, что Пролог начинает искать альтернативные пути унификации цели **start** и переходит ко второму предложению. Во втором предложении создаётся файл с базой данных путём открытия файла для записи и последующего его закрытия. Такая процедура приведёт к созданию пустого файла, а в оперативной памяти не окажется никаких фактов относительной фамилий и оценок, так как фактов пока и не было. После чего управление также передаётся в меню.

Модифицируем цель программы:

```
GOAL
    start
```

Однако запустить такую программу не удастся, пока не будет определён тип переменной **f**. Следует в разделе описания типов данных (**DOMAINS**) добавить запись о том, что **f** является файловой переменной. Причем объявление файловой переменной происходит не совсем так, как объявление переменной любого другого типа – сначала назначается тип (файловый), а затем имя переменной (если их несколько, то переменные записываются через знак «**;**»):

```
    file=f
```

Теперь программа полностью работоспособна.

Испытайте её в работе – добавьте несколько записей, посмотрите как меняется файл с данными, проверьте как программа ищет ответ на запрос «найти оценку по фамилии».

ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГОИСПОЛНЕНИЯ

Модифицируйте описание предиката **`m(char)`** так, чтобы программа была способна выполнить такие запросы:

1) выдать на экран всю базу фамилий и оценок,
2) вывести оценку по фамилии,
3) вывести фамилию по оценке,
4) вывести все фамилии по заданной оценке,
5) удалить запись из базы данных по заданной фамилии,
6) вывести фамилии студентов, у которых оценка выше заданной.

ВНИМАНИЕ!!!

Программу следует оформить с использованием предиката **`makewindow`** и так, чтобы она могла быть запущена без среды программирования Пролог.