Лабораторная работа 4-5.

Основы работы с языком программирования Prolog в среде SWI Prolog.

План лабораторной работы:
1. Цель работы
2. Описание лабораторной работы (теоретическая часть)
3. Задание на лабораторную работу
4. Контрольные вопросы

Цели работы:
1) Изучить основы синтаксиса языка Пролог.
2) Выработать навыки работы с интерактивной системой SWI- Prolog.
3) Научиться оформлять отношения между данными на языке Пролог
на примере выбора станков для обработки детали.
4) Знакомство с основами логического программирования на примере языка Prolog.
5) Изучить принципы построения рекурсии и работы со списками.

Теоретическая часть

Описание лабораторной работы

Задание 1

Разработать программу, в которой в виде предикатов фактов представлены некоторые знания об обработке разных деталей на различных станках. В процессе интерпретации целевого предиката программа должна отвечать на вопросы о том, на каком станке можно обработать заданную деталь и какие детали можно обрабатывать на заданном станке.

Порядок выполнения работы:

1. Запустить программу SWI-Prolog.
2. Создать с использованием встроенного редактора файл с текстом программы для вычисления факториала (см. методические указания ниже).
3. Загрузить файл в интерпретатор с помощью команды *consult*.
4. Задать точку перехвата в отладчике (см. методические указания ниже). Запустить программу вычисления факториала и изучить, как она выполняется, используя отладчик и точку перехвата.
5. Во встроенном редакторе создать файл с программой, включающей в себя предикаты-факты, описывающие то, на каком станке обрабатывается та или иная деталь, тип обработки для деталей, и правила вывода, доказывающие можно ли обработать деталь на
заданном станке (см. методические указания).
6. Загрузить программу и поэкспериментировать с ней, задавая ей различные варианты запросов (с неопределенными переменными и со значениями вместо них).
7. Оформить файл с отчетом по работе.

*Содержание отчета***:**
1) титульный лист;
2) задание;
3) исходные тексты;
4) выводы по работе.

Описание программы SWI-Prolog

Исполняемый файл программы SWI-Prolog размещается в **..\bin\plwin.exe**. Для загрузки программы следует вызвать пункт меню “File / Consult…” и выбрать файл с текстом программы, который может быть создан в текстовом редакторе Notepad. Расширение файла по умолчанию .PL. Файл можно создавать и в оболочке программы SWI-Prolog пунктом меню “File / New…”. По умолчанию вызывается редактор Notepad

(Блокнот). В составе пакета SWI-Prolog имеется более продвинутый редактор (PCE_EMAX), анализирующий синтаксис Пролог-программы, позволяющий устанавливать точки прерываний и т.п. Чтобы включить редактор PCE_EMAX, необходимо в файле pl.ini удалить комментарии в строке

:- set_prolog_flag(editor, pce_emacs). Сделать это можно любым текстовым редактором, либо вызвав пункт меню "*Settings / User init file*…".

После запуска программы на экране появится приглашение для ввода запросов:

?-

Запрос (вопрос) вводится после приглашения и обязательно заканчивается точкой, например,

?- 5+4<3.

No

Пролог анализирует запрос и выдает ответ Yes (Да) в случае истинности утверждения и No (Нет) в противном случае или когда ответ не может быть найден.

Теперь рассмотрим работу оболочки SWI-Prolog на примере программы вычисления факториала:

```
fact(0,1).
fact(N,F) :- N1 is N-1,
fact(N1,F1),
F is N*F1.
```

Введем в основном окне программы цель: fact(4,F). Программа выдаст ответ: F = 24. В случае ошибок на этапе трансляции или выполнения Пролог в этом же окне выдает диагностические сообщения. Пользоваться таким режимом неудобно. Лучше воспользоваться отладчиком. Для этого необходимо включить опцию графического отладчика "*Debug / Graphical debugger*", затем с помощью пункта меню "*Debug / Edit spy points*…" включить точку перехвата на интересующий нас предикат, например, в нашем случае на предикат fact, и нажать кнопку с изображением шпиона. После этого нужно в окне Пролога задать цель, и появится окно отладки. Панель Bindings отображает текущие значения переменных, Call Stack – состояние стека, т.е. глубину вложенности рекурсий, наконец, нижняя панель – текст программы, где зеленым цветом выделяется выполняемый предикат. Для пошагового выполнения программы нужно нажимать на кнопку со стрелкой вправо.

После этого Пролог начинает выполнять собирать факториал из запомненных в стеке значений 1!, 2! и 3!.

Таким образом, мы можем отслеживать логику работы предикатов и выявлять ошибки.

Хранят программы на языке Пролог в текстовых файлах, чаще всего имеющих расширение pl, например, example1.pl. Для того чтобы Пролог мог оперировать информацией, содержащейся в файле, его нужно загрузить в интерпретатор. Это можно сделать несколькими способами. При использовании первого варианта в квадратных скобках записывается имя файла (без pl), например,

?- [example1].

В случае удачного завершения этой операции будет выдано сообщение, аналогичное следующему:

% example1 compiled 0.00 sec, 612 bytes

Yes

В противном случае будет выдан список ошибок (ERROR) и/или предупреждений (Warning).

Второй способ состоит в вызове встроенного предиката consult, которому в качестве аргумента передается имя файла (также без расширения), например:

?- consult(example1).

Расширение **pl** часто используется для файлов, содержащих программы на языке

программирования Perl, поэтому можно встретить и другие расширения для файлов с программами на Прологе. Для загрузки файла с расширением, отличным от pl, имя файла следует обязательно заключать в апострофы:

?- consult('example2.prolog').

?- ['example2.prolog'].

Обе эти команды добавляют факты и правила из указанного файла в базу данных Пролога. Можно загружать несколько файлов одновременно. В этом случае они перечисляются через запятую, например,

?- [example1, 'example2.prolog'].

Важно помнить, что все запросы должны заканчиваться точкой. Если вы забудете ее поставить, то Пролог выведет символ '|' и будет ожидать дальнейшего ввода. В этом случае надо ввести точку и нажать клавишу Enter.

Программирование на языке Пролог состоит из следующих этапов:

1) объявления некоторых *фактов* об объектах и отношениях между ними,

2) определения некоторых *правил* об объектах и отношениях между ними,

3) формулировки *вопросов* об объектах и отношениях между ними.

1. Факты. Предположим, надо сформулировать на Прологе факт, что «Деталь обрабатывается на станке». Этот факт включает в себя два объекта «деталь» и «станок», связанные отношением «обрабатывается». В языке Пролог используется стандартная форма записи фактов: обрабатывается(деталь,станок).

Важно соблюдать при записи фактов следующие правила:

− Имена всех отношений и объектов должны начинаться со строчной буквы. Например, обрабатывается, деталь, станок.

− Сначала записывается имя отношения. Затем в круглых скобках через запятую записываются имена объектов.

− Каждый факт должен заканчиваться точкой.

Определяя с помощью фактов отношения между объектами, необходимо учитывать, в каком порядке перечисляются имена объектов внутри круглых скобок. Этот порядок может быть произвольным, но, выбрав один раз определенный порядок, необходимо следовать ему и дальше. Например, в приведенном ниже примере на первом месте стоит объект, который обрабатывается, а на втором - который обрабатывает. Таким образом, **обрабатывается(деталь,станок)** и **обрабатывается(станок,деталь)** не одно и тоже.

Имена объектов, список которых в каждом факте заключен в круглые скобки, называют аргументами. Имя отношения, которое записывается непосредственно перед скобками, называется именем предиката. Получаем, что «обрабатывается» - это предикат с двумя аргументами. Совокупность данных в Прологе называется **базой данных.**

2. Вопросы. Имея некоторую совокупность фактов, мы можем обращаться к Прологу с вопросами о них. В Прологе вопрос записывается почти так же, как и факт, за исключением того, что перед ним ставится специальный символ « **?-** », например:

?- обрабатывается(деталь,станок).

Обращение к Прологу с вопросом инициирует процедуру поиска в базе данных, ранее введенной в систему. Пролог ищет факты, сопоставимые с фактом в вопросе. Два факта сопоставимы (или соответствуют один другому), если их предикаты одинаковы (побуквенное совпадение) и их соответствующие аргументы попарно совпадают. Если Пролог находит факт, сопоставимый с вопросом, то он отвечает **true**. Если в базе данных такого факта не существует – то **false**.

Для того, чтобы обеспечить поиск ответов на более сложные вопросы, например «Что обрабатывается на станке?» в Прологе используются *переменные*.

3. Переменные. В Прологе можно не только присваивать имена конкретным данным, но и использовать имена, подобные **X** (неизвестное), для обозначения объектов, которые должны быть определены системой. Имена такого типа называются *переменными*. В Прологе используется соглашение, согласно которому каждое имя, начинающееся с

прописной буквы, рассматривается как переменная.

При поиске ответа на вопрос Пролог организует просмотр всех фактов в базе данных, чтобы обнаружить объект, который эта переменная могла бы обозначать. Так при вопросе «Обрабатывается ли на станке X?», программа просматривается все известные ему факты для обнаружения тех вещей, которые могут обрабатываться на станке. Такая переменная, как **X**, сама по себе не является именем какого-то конкретного объекта, но она может быть использована для обозначения объектов, которым не получается пока что дать имя.

Пролог позволяет использовать переменные и задавать вопросы в виде:

?- обрабатывается(X,станок).

Возможны и другие варианты обозначения переменной, например:

?- обрабатывается(Что-то, на станке).

Рассмотрим следующую базу данных:

обрабатывается(фланец,станок_1).

обрабатывается(втулка,станок_1).

обрабатывается(колесо,станок_1).

и запрос к Прологу:

?- обрабатывается(X,станок_1).

Смысл этого запроса – «Какие детали обрабатываются на станке ‘станок_1’»

В ответ Пролог предложит вариант:

X=фланец

А затем будет ждать дальнейших приказов. Если этот вариант ответа устраивает, то после нажатия кнопки Enter, он прекратит поиск в базе данных. Если вместо этого нажать «;», то поиск продолжится, и будут выведены следующие варианты значения переменной.

X=фланец;

X=втулка;

X=колесо.

4. Правила.

Предположим, что есть утверждение, что «на данном станке могут обрабатываться все детали, полученные литьем». Один из способов сделать это заключается в записи для каждой детали отдельного факта. Но это не рационально.

Другой способ выразить факт, что на станке могут обрабатываться все детали, при условии, что они получены литьем. Здесь этот факт представлен в форме правила для определения того, какая деталь может обрабатываться на станке, а не прямого перечисления всех деталей.

В Прологе правила используются в том случае, когда необходимо выразить зависимость некоторого факта от группы других фактов. Правило – это некоторое общее утверждение об объектах и отношениях между ними.

В Прологе правило состоит из заголовка и тела правила. Заголовок и тело соединяются с помощью символа « :- ». Пример, приведенный выше, запишется следующим образом:

обрабатывается(X,станок_1) :- литье(X).

Правила также, как и факты заканчиваются точкой. Заголовком этого правила является предикат **обрабатывается(X,станок)**. Заголовок правила описывает тот факт, для доказательства которого предназначено правило.

Тело правила, в данном случае **литье(X)**, описывает цель, которая должна быть последовательно согласованна с базой данных, для того чтобы заголовок правила был истинным (был доказан). В качестве примера рассмотрим следующую базу данных:

литье(втулка).

литье(колесо).

штамповка(диск).

штамповка(фланец).

В данном случае используем правило, означающее, что на станке может обрабатываться только деталь, полученная литьем. Это правила можно записать в следующем виде:

будет_обрабатываться(X, станок_1) :- литье(X).

В результате получаем:

?- будет_обрабатываться(X,станок_1).

X = втулка ;

X = колесо.

Контрольные вопросы:

1. Как можно загрузить программу из файла?
2. Как запускается программа на Прологе?
3. Как отображаются в программе на Прологе свойства сущностей?
4. Как отображаются в программе на Прологе отношения между сущностями?
5. Как отобразить в программе на Прологе проверку конъюнкции двух предикатов?
6. Что такое унификация в Прологе?
7. Что такое правило в Прологе?

Задание 2

Составить список фрезерных станков с их параметрами, осуществить поиск станка в данном списке и вывести его параметры. Основной параметр - это ширина стола. Для данных станков ширина стола имеет следующие размеры: 200, 250, 320, 400, 500.

В таблице 1 приведены виды станков с параметрами.

Таблица 1. Перечень станков

Тип станка	*Название*	*Размер стола*
вертикально-фрезерный	6М10	200 мм
вертикально-фрезерный	6Н11	250 мм
горизонтально-фрезерный	6М80Г	200 мм
горизонтально-фрезерный	6Н81Г	250 мм
вертикально-фрезерный	6Н11	320 мм
горизонтально-фрезерный	6М82Г	320 мм
вертикально-фрезерный	6М13	400 мм
горизонтально-фрезерный	6М83Г	400 мм
вертикально-фрезерный	6Н14	500 мм
горизонтально-фрезерный	6Н84Г	500 мм

Порядок выполнения работы

1. Поэкспериментировать с правилом «*принадлежит*», описанном в методических указаниях, и изучить с помощью отладчика как работает рекурсивный перебор элементов списка.
2. Создать во встроенном редакторе программу, в которой представлена информация из выше приведенной таблицы в виде унарных предикатов станок с аргументом – списком из трех параметров (из столбцов таблицы), правил логического вывода с рекурсией для поиска параметров станка по заданному одному параметру. Предикатов «станок» должно быть 10 (по одному на каждую строку таблицы). Программа должна содержать следующие правила:

a) правило *найти*, с которого запускается программа найти(X) :- станок(L), параметры(X,L), phh(L), где: L – список параметров.

b) правило *параметры* с рекурсией для поиска заданного параметра X (написать самостоятельно)

c) правило *phh* для вывода на экран списка параметров в виде

строки (написать самостоятельно).
3. Оформить отчет по лабораторной работе.
*Содержание отчета***:**
1) титульный лист;
2) задание;
3) исходные тексты;
4) выводы по работе.

Рекурсия и работа со списками в языке Prolog

Рекурсия в Прологе. Правило является рекурсивным, если содержит в качестве компоненты само себя. Рекурсия допустима в большинстве языков программирования (например, в Паскале), но там этот механизм не является таким важным, поскольку имеются другие, свойственные процедурным языкам, механизмы – циклы, процедуры и функции.

Рассмотрим преимущества использования рекурсии на примере.

Пусть имеются следующие факты о том, какая валюта котируется выше:

```
doroje(dollar, rubl).
doroje(evro, rubl).
doroje(rubl, iena).
doroje(funt, euro).
```

Выполним запрос:

```
? doroje(evro, rubl).
Yes
```

Будет получен утвердительный ответ, поскольку такой факт явно описан в программе. Если же сделать запрос,

```
? doroje(evro, iena).
```

То ответ будет отрицательный, поскольку такой факт отсутствует. Аналогичным будет ответ на вопрос:

```
? doroje(funt, rubl).
```

Избежать таких неправильных ответов здесь можно введением правила, в котором допустимо сравнение между собой не только двух, но и трех объектов:

```
doroje1(X, Y):- doroje(X, Y). /* два объекта */
doroje1(X, Y):- doroje(X, Z), doroje(Z, Y). /* три объекта */
```

Второе правило описывает вариант, когда X>Z, а Z>Y, откуда делается вывод, что X>Y.

Однако цепочка взаимных сравнений может быть длинной. Например, при четырех сравнениях потребуется конструкция:

```
doroje2(X, Y):- doroje(X, M), doroje(M, K), doroje(K, Z), doroje(Z, Y).
```

Описывать такие длинные правила неудобно. Здесь выгоднее применить рекурсию, обратившись к правилу из самого этого правила:

```
doroje1(X, Y):- doroje(X, Y).
doroje1(X, Y):- doroje(X, Z), doroje1(Z, Y).
```

Первое предложение в этой конструкции определяет момент прекращения рекурсивных вызовов.

Второе правило описывает возможности рекурсивных вызовов, когда существуют непроверенные варианты решения. Вообще, любая рекурсивная процедура должна содержать:

1) *Нерекурсивное правило*, применяемое для завершения рекурсии (их может быть несколько, задающие разные условия для завершения рекурсии),
2) *Рекурсивное правило*, первая подцель которого вырабатывает новые значения аргументов, а вторая – рекурсивная подцель – использует эти значения.

Набор правил просматривается сверху вниз. Сначала делается попытка выполнения нерекурсивного правила. Если оно отсутствует, то рекурсивное правило может работать

бесконечно.

Использование списков. Списки – одна из наиболее часто употребляемых структур в ПРОЛОГе. *Список* – это набор объектов одного и того же типа. При записи список заключают в квадратные скобки, а элементы списка разделяют запятыми, например:

[1,2,3]

[1.1,1.2,3.6]

[“вчера”,”сегодня”,”завтра”]

Элементами списка могут быть любые термы ПРОЛОГа, т.е. атомы, числа, переменные и составные термы. Каждый непустой список может быть разделен на *голову* – первый элемент списка и *хвост* – остальные элементы списка. При этом список представляется в виде

[A|B] или [1| [2,3]] или [1| B] или [“вчера”| [“сегодня”,”завтра”]].

Это позволяет всякий список представить в виде бинарного дерева (рис.1). У списка, состоящего только из одного элемента, головой является этот элемент, а хвостом – пустой список. Для использования списка необходимо описать предикат списка.

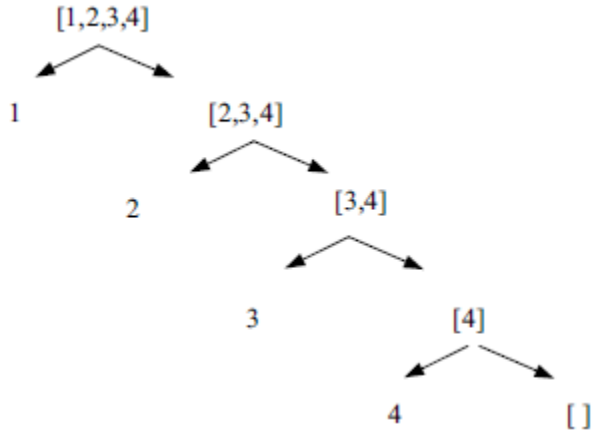

Рис. 1. Бинарное дерево списка

В следующем примере 1 - голова списка, а [2, 3, 4, 5] - хвост. Пролог покажет это при помощи сопоставления списка чисел с образцом, состоящим
из головы и хвоста.

```
?- [1, 2, 3, 4, 5] = [Head | Tail].
Head = 1
Tail = [2, 3, 4, 5]
Yes
```

Здесь *Head* и *Tail* - только имена переменных. Мы могли бы использовать X и Y или какие-нибудь другие имена переменных с тем же успехом. Заметим, что хвост списка всегда является списком. Голова, в свою очередь, есть элемент списка, что верно и для всех других элементов, расположенных до вертикальной черты. Это позволяет получить, скажем, второй элемент списка.

В Prolog используется специальный символ для разделения списка на голову и хвост – вертикальная черта |, например:

[1, 2, 3] или [1 | [2, 3]] или [1 | [2| [3]]] или [1 | [2 | [3 | [ ]]]]

Вертикальную черту можно использовать не только для отделения головы списка, но и для отделения произвольного числа начальных элементов списка:

[1, 2, 3] или [1, 2 | [3]] или [1, 2, 3 | [ ]]

Пример:

Используем анонимные переменные для головы и списка, стоящего после черты, если нам нужен только второй элемент списка:

```
?- [слон, лошадь, осел, собака] = [_, X | _ ].
X = лошадь
```

Yes

Рассмотрим несколько процедур обработки списков. Обратите внимание, что все они используют рекурсию, в которой терминальное (базовое) правило определено для пустого списка.

Пример:

Предикат «перестановка» выдает списки, полученные перестановкой элементов своего первого аргумента.

```
перестановка([],[]).
перестановка([H|L],Z):- перестановка(L,Y), место(H,Y,Z).
```

Пример использования:

```
?- перестановка([a,b,c],X).
X = [a, b, c] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [a, c, b] ;
X = [c, a, b] ;
X = [c, b, a] ;
No
```

Предположим, что имеется некоторый список, в котором **X** обозначает его голову, а **Y** - хвост списка. Такой список можно записать, как **[X|Y]**. Этот список может содержать, например, инструмент для обработки деталей:

[фреза, сверло, резец, хон]

Теперь предположим, что мы хотим определить: содержится ли некоторый инструмент в указанном списке. В Прологе это можно сделать, определив, совпадает ли данный инструмент с головой списка. Если совпадает, то наш список завершается успехом. Если нет, то мы проверяем, есть ли нужный инструмент в хвосте исходного списка. Это значит, что снова проверяется голова, но уже очередного хвоста списка. Если мы доходим до конца списка, который будет пустым списком, то наш поиск завершается неудачей: указанного инструмента в исходном списке нет. Для того, чтобы записать все это на Прологе, сначала надо установить, что между объектом и списком, в который этот объект может входить, существует отношение. Для записи этого отношения будем использовать предикат **принадлежит**: целевое утверждение **принадлежит(X,Y)** является истинным («выполняется»), если терм, связанный с **X,** является элементом списка, связанного с **Y**. Имеются два условия, которые надо проверить для определения истинности предиката. Первое условие говорит, что **X** будет элементом списка **Y**, если **X** совпадает с головой списка **Y**. На Прологе этот факт записывается следующим образом:

```
принадлежит(X,[X|_]).
```

Эта запись констатирует, что **X** является элементом списка, который имеет **X** в качестве головы. В данном случае анонимная переменная «_» для обозначения хвоста списка.

Второе правило говорит о том, что **X** принадлежит списку при условии, что он входит в хвост этого списка, обозначаемый через **Y**. Для этого используем тот же самый предикат **принадлежит** для того, чтобы определить, принадлежит ли **X** хвосту списка. В этом и состоит суть рекурсии. На Прологе это выглядит так:

```
принадлежит(X,[_|Y]) :- принадлежит(X,Y).
```

Два этих правила в совокупности определяют предикат для отношения принадлежности и указывают Прологу, каким образом просматривать список от начала до конца при поиске некоторого элемента в списке. Наиболее важный момент, о котором следует помнить, встретившись с рекурсивно определенным предикатом, заключается в том, что прежде всего надо найти граничные условия и способ использования рекурсии. Для предиката **принадлежит** в действительности имеются два типа граничных условий. Либо объект, который мы ищем, содержится в списке, либо нет. Первое граничное условие

для предиката **принадлежит** распознается первым утверждением, которое приводит к прекращению поиска в списке, если первый аргумент предиката **принадлежит** совпадает с головой списка. Второе граничное условие встречается, когда второй аргумент предиката **принадлежит** является пустым списком. Это все демонстрирует следующий пример на Прологе:

```
принадлежит(X,[X|_]).
принадлежит(X,[_|Y]) :- принадлежит(X,Y).
?- принадлежит(фреза,[фреза, сверло, резец, хон]).
true
?- принадлежит(протяжка,[фреза, сверло, резец, хон]).
false.
```

Контрольные вопросы:

1. Как представляется список в Прологе? Примеры.
2. Что такое рекурсия?
3. Для чего используется представление списка в виде головы и хвоста?
4. Какое минимальное количество предложений Пролога необходимо для программирования рекурсии?